UNIVERSIDAD DE ALMERÍA

Escuela Politécnica Superior y Facultad de Ciencias Experimentales

Ingeniería en informática

Diseño de un paquete de gestión de servicios por proximidad y estudio de pasarelas de pago para dichos servicios

Autor: **Jonás Urbano Romero**

Director: **José Antonio Torres Arriaza**

Almería, marzo de 2014

Índice

1. Introducción

Este texto es la memoria de un proyecto final de carrera donde se documenta el desarrollo de dos librerías integrables en proyectos de software para la plataforma Android. Con estas librerías se pretende facilitar la implantación de pasarelas de pagos y la transformación de aplicaciones Android a servicios basados en geolocalización de forma sencilla.

Así mismo, se resume el estudio realizado sobre diferentes servicios de pago y servicios de geolocalización así como una breve descripción del protocolo que rige el funcionamiento de la moneda electrónica Bitcoin.

1. Contenido de la memoria

El contenido de la presente memoria se divide en seis puntos principales con el objetivo de hacer cómoda e intuitiva la lectura del documento. Dichos seis puntos son:

Introducción: Donde se desglosa el contenido de la memoria, se justifican las motivaciones y se plasman los objetivos así como la planificación diseñada para alcanzar estos objetivos.

Estado del arte: Donde se describen los conceptos básicos de las dos áreas estudiadas y se resume con ánimo de comparativa qué desarrollos de software y hardware se han llevado a cabo hasta el momento.

Diseño e implementación: Donde se discuten las decisiones de diseño y se explica cómo se han implementado y como usar las librerías de geolocalización y pagos que se han desarrollado en este proyecto.

Conclusiones y trabajos futuros: Donde se analiza el cumplimiento de los objetivos descritos en la introducción y se proponen futuras vías de continuación del trabajo realizado.

Referencia y bibliografía: Donde se recogen las referencias que han ayudado a la elaboración del proyecto y del documento.

Anexos: Donde se incluyen capturas de pantalla de las actividades de las librerías en la implementación sobre un paquete de gestión de servicios.

El bloque sobre el estado del arte se divide en tres puntos. El primero de ellos es Comercio electrónico desde dispositivos móviles y abarca en sus diferentes apartados la actualidad del comercio móvil, los conceptos básicos y los servicios de pago disponibles. El

segundo punto del estado del arte se titula Geolocalización en dispositivos móviles con dos apartados: Aplicaciones y servicios basados en geolocalización donde se exponen los dos servicios reconocidos que más enriquecen su producto mediante geolocalización El segundo apartado del bloque de geolocalización trata sobre las herramientas, librerías y frameworks que se pueden usar para dotar a un desarrollo de software de geolocalización. El tercer punto del estado del arte expone Bitcoin, una moneda electrónica cuyo uso se está extendiendo en internet.

El tercer bloque cubre el diseño e implementación de las librerías comenzando con el análisis del problema y la solución junto con las decisiones tomadas que desembocan en dos apartados como es la tónica de este documento.

El apartado 3.3 de geolocalización en dispositivos móviles y el apartado 3.4 de comercio electrónico desde dispositivos móviles se estructuran de la siguiente manera:

- Se presenta el diseño de la librería para el dispositivo móvil
- Se explica la implementación de la librería para el dispositivo móvil
- Se presenta el diseño de la librería para la aplicación servidor
- Se explica la implementación de la librería para la aplicación servidor
- Se describen los servicios de terceros usados para la implementación de la librería

El penúltimo apartado del tercer bloque describe qué herramientas se han utilizado desde un punto de vista personal remarcando qué características han contribuido en mejorar la productividad del desarrollo. Sin olvidar el último apartado que esboza la metodología seguida con el objetivo de realizar un buen desarrollo de software.

2. Motivación del proyecto

Las aplicaciones para dispositivos móviles han ganado mucha popularidad. Es sencillo descargar e instalarlas desde las respectivas tiendas de aplicaciones. Por lo general, la descarga de las apps es gratuita. Sin embargo, la industria del software para dispositivos móviles ha adoptado el modelo de negocio de pago in-app que es el pago desde la misma aplicación, ya sea para obtener servicios adicionales o más niveles en un juego.

Lamentablemente, las pasarelas de pago no son productos software muy flexibles. Suponen complejas y muy diferentes integraciones entre pasarelas que hacen al desarrollador declinar la implantación de estás en su software.

Por otra parte, el coste de la fabricación de sistemas de posicionamiento GPS se ha abaratado hasta tal punto que hoy en día casi todos los teléfonos móviles conocidos como

smartphones integran este dispositivo. La posición geográfica de un usuario supone una información muy valiosa para enriquecer la aplicación por lo que usarla es vital en nuevos desarrollos software para plataformas móviles.

3. Objetivos

El objetivo principal de este proyecto es el desarrollo de dos librerías para integrarlas de manera sencilla en productos de software sobre la plataforma Android de tal manera que el desarrollador no vea afectada su productividad por la integración de las características que estas librerías añaden a la aplicación.

La librería de geolocalización servirá para adaptar en un proyecto de software la arquitectura propia de un servicio basado en geolocalización de manera que la experiencia de usuario se vea enriquecida por el uso de información geolocalizada.

La librería de servicios de pago pretende resolver la problemática de integrar una o varias pasarelas de pago en un mismo producto software para que tanto desarrollador como usuarios de la aplicación tengan varias vías disponibles en cuanto a transferencias económicas se refiere.

El segundo objetivo consiste en adquirir la experiencia necesaria tras un desarrollo completo de una aplicación para el sistema operativo Android con la finalidad de entender los requisitos característicos de este tipo de aplicaciones.

El tercer y último objetivo busca conseguir el conocimiento sobre servicios de pagos y servicios de geolocalización con el que poder abarcar problemas de desarrollo de software en estos campos y encontrar la solución que más se adecúe a las restricciones del problema.

4. Planificación

El proyecto final de carrera se espera que tenga una duración de seis meses y se desarrollará en las siguientes seis fases:

Primera fase: Planteamiento general y preparación en la que se configurará el entorno y las herramientas necesarias para un proyecto de software.

Segunda fase: Estudio de servicios de pago para la integración en proyecto de software Android. Estudio del de la moneda Bitcoin.

Tercera fase: Estudio de servicios de geolocalización y de cómo aprovechar el sistema operativo Android para geolocalizar dispositivos.

Cuarta fase: Diseño e implementación de la librería de geolocalización.

Quinta fase: Diseño e implementación de la librería de pagos.

Quinta fase: Desarrollo y prueba de integración de las librerías para Android.

Sexta fase: Documentación de la aplicación y de las áreas estudiadas durante el desarrollo del proyecto.

2. Estado del arte

Muchos son los servicios, herramientas y librerías que se pueden usar desde o integrar en una aplicación móvil para ampliar los casos de uso de ésta enriqueciendo la experiencia de usuario. En este bloque se hará un repaso de los servicios y librerías más relevantes en la actualidad para tomarlos en consideración de cara al desarrollo de una aplicación móvil que integre servicios de pagos o esté basada en geolocalización.

El último apartado de este bloque trata sobre Bitcoin, una moneda electrónica muy útil en internet y de la que hacen uso algunos servicios de pago en la actualidad. Se ha decidido no analizar el estado del arte de los servicios de pago con Bitcoin en la actualidad a favor de desarrollar un breve apartado explicando los conceptos fundamentales de la moneda.

2.1. Comercio electrónico desde dispositivos móviles

El comercio móvil, conocido en inglés como **m-commerce**[1]es el mercado de compra y venta de bienes y servicios a través de dispositivos móviles, aprovechando las características que estos ofrecen para el comerciante y el comprador tales como geolocalización o servicios de fidelización. El comercio móvil, obviamente es un subconjunto del comercio electrónico.

En 1997, Kevin Duffey, director general en International SOS, acuñó el término comercio móvil definiéndolo como la dotación de las capacidades del comercio electrónico directamente a la mano del usuario en cualquier lugar y haciendo uso de tecnología inalámbrica.

El comercio móvil en la actualidad

A pesar de que este término se acuñase por primera vez en el año 1997, la tecnología no ha facilitado el comercio móvil al usuario desde el primer momento. Un informe sobre comercio móvil en 2012 realizado por la Online Business School[2] desvela que durante el año 2012, el comercio móvil supuso el 11% de las ventas de comercio electrónico y movió 71.500 millones de Euros. Centrado en España, se ha estimado que el comercio electrónico ha movido 2.500 millones de Euros suponiendo el mismo porcentaje para el comercio electrónico del país.

[1]M-Commerce, Jack Bravo Torres

[2]http://www.obs-edu.com/noticias/2012/01/30/uno-de-cada-cuatro-espanoles-compra-por-internet-un-125-mas-que-en-2010-segun-el-informe-de-online-business-school

El comercio móvil en la ingeniería del software

El comercio móvil es proporcionado al usuario de un dispositivo móvil a través de aplicaciones web accesibles desde el navegador web del dispositivo móvil o a través de aplicaciones móviles desarrolladas sobre el sistema operativo del dispositivo. Desde el punto de vista de la ingeniería del software el comercio móvil ha requerido grandes avances tecnológicos y de usabilidad a la vez que ha supuesto un gran avance económico.

Avances tecnológicos y de usabilidad

En el año 2007 se comercializó el dispositivo móvil iPhone que popularizó los smartphones en el mercado de los dispositivos móviles. Anteriormente a la popularización de los dispositivos móviles, cada fabricante dotaba al dispositivo con un sistema propio ajustado a las necesidades del terminal y sin la posibilidad de que se desarrollase una aplicación para dicho terminal. Sin embargo, los principales sistemas operativos en la actualidad: Android, iOS y Windows Phone permiten que desarrolladores de software programen aplicaciones para estos dispositivos móviles que se ejecutan sobre dichos sistemas operativos. Es casi incontable la cantidad de dispositivos móviles diferentes que cuentan con Android como sistema operativo y sobre el que se puede construir una aplicación con la seguridad de que podrá ser instalada en una variedad de dispositivos muy amplia.

Del mismo modo, cada sistema operativo proporciona una tienda de aplicaciones desde la que se puede comprar e instalar aplicaciones para la plataforma, lo que ha generado un nuevo mercado para los desarrolladores de software que tienen como modelo de negocio el desarrollo de aplicaciones para dispositivos móviles.

La increíble expansión de los smartphones ha hecho que una de las directrices más importantes en el desarrollo de aplicaciones y sistemas operativos para dispositivos móviles sea la usabilidad. Es casi imprescindible hoy en día que una aplicación para estos dispositivos sea atractiva e intuitiva. Pero no sólo de smartphones se ha inundado el mercado, si no también de relojes (conocidos como SmartWatches) hasta televisores (SmartTVs) pasando por tablets y gafas (SmartGlasses). Esta variedad enorme de tamaños de pantalla ha derivado en nuevos paradigmas de diseño donde el contenido es lo más importante y la presentación de éste se tiene que adaptar al dispositivo sea cual sea el tamaño de su pantalla. Este paradigma se conoce como Responsive Web Design[3] (RWD) y fue Ethan

[3] http://alistapart.com/article/responsive-web-design/

Marcotte quién escribió por primera vez sobre el tema en el blog sobre diseño A List Apart en Mayo de 2010.

Avances económicos

El nivel de penetración de los smartphones en la sociedad ha sido tan alto que hoy en día muchísimas empresas desarrollan su propia aplicación para dispositivos móviles desde la que poder ofertar y vender sus productos. El mercado móvil ha renovado la industria del desarrollo de software y ha establecido un nuevo canal por el que comercializar productos y servicios.

Clasificación del comercio móvil

En 2007 se publicó un documento con el título Diffusion of Mobile Commerce Application in the Market donde sus autores establecen una taxonomía de las aplicaciones de comercio móvil según los servicios que presta. Las categorías principales son: Servicios de comunicación móvil, servicios de información móvil, servicios de transacciones móvil y servicios de interacción móvil.

Ventajas para los comerciantes y compradores

Una de las ventajas principales del comercio móvil es la geolocalización. Mediante geolocalización, el comerciante y el comprador pueden aportar su posición geográfica como información para la promoción de productos y servicios y para el proceso de compra.

La geolocalización del comprador supone información muy valiosa para el comerciante de cara a la promoción de productos y servicios, a la elaboración del plan de marketing y a la generación de valor para el cliente ya que puede promover planes de fidelización y servicios personalizados según la posición del comprador.

Para el comprador, usar geolocalización en comercio móvil le permite aplicar un filtro geográfico a los productos y servicios que le son ofertados, de forma que disminuye el bombardeo promocional al que son sometidos diariamente los usuarios. Por otra parte, que los comerciantes estén geolocalizados genera confianza en el comprador ya que es consciente de que existe un lugar físicamente al que pueden acudir si ocurre algún inconveniente.

¿Qué es un sistema de pago móvil?

Los sistemas de pagos móviles, también conocidos como **M-Payment**, fueron definidos en la conferencia Mobile Payments Forum como los procesos necesarios para el

intercambio de valores financieros entre dos partes, utilizando un dispositivo móvil para la obtención de un bien o servicio.

Participantes en un sistema de pago móvil

En un sistema de pago móvil, pueden intervenir algunos de estas partes en el proceso:

1. **Consumidor**: Persona que compra algún bien o servicio desde un dispositivo móvil.
2. **Comerciante**: Persona o compañía que oferta el bien o servicio.
3. **Proveedor de pago**: Compañía especializada en pagos que se ofrece como intermediario del proveedor financiero.
4. **Proveedor de la plataforma:** Persona o compañía que desarrolla y mantiene la aplicación sobre la que consumidor y comerciante pueden realizar transacciones.
5. **Proveedor de servicios financieros:** Son los bancos, compañías de tarjetas de crédito, instituciones de cambio de moneda, etc.
6. **Agencia reguladoras**: Agencias gubernamentales que determinan cómo los sistemas de pago móvil pueden operar en cada uno de los diferentes países.

En el caso de que la unidad monetaria sea moneda electrónica descentralizada tal como Bitcoin, es posible prescindir de proveedor de pago, proveedor de servicios financieros y agencia reguladora debido a su naturaleza descentralizada.

Ciclo de vida de un sistema de pago móvil

Un pago realizado a través de un sistema de pago móvil se puede encajar en un ciclo de vida de cinco etapas:

1. **Puesta en marcha y configuración del mecanismo de pago**. Esta fase comienza en el momento en que se instala la aplicación en el dispositivo móvil y se suele dar solamente una vez.
2. **Iniciación** del sistema de pago. Esta fase comienza con la creación o recepción de los datos necesarios para el pago y el establecimiento de la conexión con el proveedor de pago si es necesario.

3. **Autenticación del usuario.** Si el sistema de pago cuenta con un proveedor de pago, en esta fase, el usuario de la aplicación inicia sesión en el sistema de dicho proveedor de pago.

4. **Terminación del pago.** El pago se acepta y se obtiene un recibo del pago

5. **Verificación del pago.** Si el sistema cuenta con el servicio de verificación de pago, la plataforma se comunicará con el proveedor de pago para verificar los datos obtenidos en el recibo de pago.

Un sistema de pago electrónico es un conjunto de componentes tanto software como hardware que unidos proporcionan el servicio de transferencia económica entre dos entidades que negocian a través de internet. Los sistemas de pago más relevantes en el comercio electrónico suelen ser las pasarelas de pago, los TPV virtuales y los monederos electrónicos.

Las pasarelas de pago son sistemas de pago útiles para el vendedor que ofrece a los compradores diferentes formas de pago desde transferencia bancaria hasta tarjetas de débito o crédito pasando por dinero electrónico de la propia pasarela. A los compradores les proporciona seguridad y facilidad en el proceso de pago.

Los TPV virtuales son sistemas de pago software proporcionados por la entidad bancaria del vendedor y permite al comprador realizar el pago con tarjeta de crédito o débito directamente con la entidad bancaria adquiriente. Es interesante para los vendedores ya que permite el ahorro de comisiones para intermediarios como podría ser un proveedor de pasarela de pagos.

Los monederos electrónicos son servicios que permiten convertir dinero de curso legal en dinero propio del monedero electrónico para pagar a otros servicios que permitan dicho dinero. Es un servicio que actúa sólo de parte del comprador y generalmente no ofrece tanta posibilidad de conversión de dinero de curso legal a dinero del monedero electrónico. Este tipo de servicio se está popularizando añadiendo características adicionales como almacenamiento de tarjetas de fidelización virtuales y promociones.

Típicamente, el mercado de los servicios de pago ha estado gobernado por grandes empresas de la talla de Paypal. Con la popularización de los dispositivos móviles han surgido muchas pequeñas empresas muy especializadas en servicios de pagos lo que ha llevado a estas empresas ofrecer servicios conocidos como PaaS o Payments As A Service.

2.1.1. Pasarelas de pago

Las pasarelas de pago (en inglés, payment gateways) son servicios que autorizan el pago con tarjetas de crédito, débito o trasferencia bancaria en comercios electrónicos. Las pasarelas de pago se pueden considerar como el equivalente a los terminales de punto de venta (TPV) que se instalan en los comercios físicos para permitir cobrar desde una tarjeta de crédito o débito. De hecho, muchas entidades financieras o proveedores de tarjetas ofertan TPV virtuales que, generalmente, ofrecen los mismos servicios que una pasarela de pago.

El rango de acción de las pasarelas de pago comienza cuando el comprador pulsa en un botón para comprar el producto o servicio hasta que la entidad financiera del comprador emite el pago a la entidad adquiriente. La labor fundamental de la pasarela de pago es facilitar este proceso a la vez que asegurar los datos que se intercambian. En cuanto a seguridad, las pasarelas de pagos deben cumplir la norma PCI DSS (Payment Card Industry Data Security Standard).

El proceso genérico de una transacción suele contener los siguientes pasos:

1. El comprador pulsa sobre el botón para comprar el pedido.

2. La página web del vendedor redirige al comprador a la web de la pasarela de pago incluyendo los datos del pedido como concepto e importe. Esta conexión es cifrada.

3. El comprador introduce los datos de su tarjeta de crédito o débito, que usualmente son el número de la tarjeta, la fecha de caducidad y los dígitos de control.

4. La pasarela de pagos envía los datos a la entidad financiera del vendedor que autorizará el pago a través del procesador de pagos.

5. La pasarela de pagos envía la petición de pago al proveedor de la tarjeta de crédito o débito (Visa, Mastercard...). Este proveedor, a su vez, envía los datos de la transacción a la entidad financiera que asumirá el pago que se haga con la tarjeta. Esta entidad acepta o rechaza el pago en función de ciertas verificaciones de seguridad contra el fraude.

6. La pasarela de pago recibe tanto la respuesta de la autorización del procesador de pagos como la respuesta de la petición (a través del proveedor de la tarjeta) de la entidad financiera que asumirá el pago. Si las dos respuestas son afirmativas, el pago ha sido aprobado y se dispone a ser liquidado repitiendo desde el paso 4 para confirmar a cada parte el acuerdo de la transacción.

7. La pasarela de pago notifica al comprador el éxito o fracaso de la transacción.

Las pasarelas de pagos más conocidas, son Paypal, Authorize.Net y Amazon FPS. Los precios que aparecen a continuación corresponden a los indicados en los sitios webs de las diferentes pasarelas de pago en diciembre de 2013.

Características de las pasarelas de pagos

1. **Confianza**: La confianza que tienen los posibles compradores con una pasarela de pago es determinante para que realicen el pago. Por lo general, la confianza se puede aproximar con la popularidad de las pasarelas de pagos. Por lo menos en España, la pasarela de pagos más conocida es Paypal.

2. **Costes**: Suele haber tres tipos de costes producidos por el uso de una pasarela de pagos en el comercio electrónico: Coste de instalación, conste por transacción y coste de mantenimiento. En el mercado de las pasarelas de pago, la competitividad suele lucharse a través del coste de transacción que calculan en función del porcentaje del importe de la transacción más una pequeña cantidad adicional.

3. **Suscripciones**: De un tiempo a esta parte se han popularizado los servicios SaaS cuyo modelo de negocio se sustenta sobre la suscripción a distintos niveles de características. En mayor o menor medida todas las pasarelas de pagos permiten realizar pagos recurrentes que facilitan al vendedor el cobro de su servicio por suscripciones.

4. **API**: Las herramientas que ofrece la pasarela de pagos para el desarrollador son una característica determinante. No sólo ofrecer un amplio rango de funcionalidades si no facilitar el desarrollo hasta el punto de que instalar una pasarela de pagos con la configuración y funcionalidad típica sea prácticamente trivial. Los desarrolladores valorarán muy positivamente aquellas pasarelas de pago que cuya documentación sea perfectamente entendible y además proporcionen una plataforma de pruebas.

5. **Seguridad**: La seguridad es la característica esencial de una pasarela de pago y la más difícil de demostrar. Es necesario cifrar las conexiones y la información y cumplir con normas ya impuestas en el mercado como AVS (Address Verification Service) que es un sistema anti fraude.

Uso de las pasarelas de pago en España

El observatorio nacional de las telecomunicaciones y de las SI publicó en 2013 el informe Estudio sobre comercio electrónico B2C 2012. De este informe se arrojan datos

como que el uso de la tarjeta de crédito o débito había aumentado del 2011 al 2012 un 3,3% alcanzando el 66.2% del total de las compras realizadas entre clientes comunes y empresas. En contra, el uso de las plataformas de pago había descendido del 14.9% al 11.6% de 2011 a 2012.

Ante estos datos, la reflexión invita a pensar que el motivo principal de que el uso de las pasarelas de pago sea menor en beneficio de las tarjetas de crédito o débito a través de un TPV virtual sea la desinformación y el desconocimiento de las ventajas que una pasarela de pago aporta a un comprador. En este ámbito, los proveedores de servicios de pago deben enfocar sus esfuerzos en campañas promocionales que pongan en valor las ventajas de usar sus servicios y sean tan reconocidos y confiables como las entidades bancarias que respaldan los puntos de venta virtuales.

De hecho, si para un comprador el único contacto con la pasarela de pagos es el botón para iniciar el proceso de pago junto al producto o servicio, el comprador no se vería aventurado a usar la pasarela de pagos porque desconoce cómo se paga realmente. En un ejercicio de información o ejemplo de caso de uso, el botón para pagar con la pasarela de pagos podría indicar que con la pasarela de pagos también se podría pagar con tarjeta de crédito o débito.

La lucha por expandir el uso de las pasarelas de pago es ardua. Hoy en día las entidades bancarias están poniendo mucho empeño y aplicando muchas medidas de seguridad visibles al usuario para evitar el fraude y aumentar la confianza en el pago con sus tarjetas de crédito o débito o transferencia bancaria. En cuanto a la transferencia bancaria en muchas bancas electrónicas se requiere una clave impresa en una tarjeta física y un código recibido por SMS, habiendo introducido anteriormente nombre de usuario y contraseña para iniciar sesión en el servicio web. En cuanto a tarjetas de crédito o débito, una medida de seguridad que hace al usuario de la entidad bancaria confiar en la seguridad de sus compras en internet es la creación de tarjetas virtuales justo antes de la compra y cuyo único uso es ésta. En cuanto al prestigio de la entidad bancaria sobre las pasarelas de pago, es obvio. El simple reconocimiento en publicidad televisiva es suficiente para desequilibrar la balanza a favor de la entidad bancaria.

Paypal

Paypal es una empresa a nivel mundial de comercio electrónico que permite realizar transferencias a través de internet. La empresa Paypal como tal no es una entidad

financiera, de hecho, en el apartado Acerca de nosotros[4] en su sitio web justifican que al ser su servicio dinero electrónico, no se considera como depósito o servicio de inversiones según la ley, con lo que sus clientes no se pueden acogerse a planes de garantía de depósitos. Aun así, Paypal está sujeta a la lista de sanciones económicas de los Estados Unidos y a otras regulaciones y leyes que el gobierno americano impone. Paypal llegó a ser muy popular como pasarela de pago entre los clientes de eBay hasta el punto que eBay compró Paypal el 3 de octubre de 2002 ante la imposibilidad de que su sistema de pagos propios compitiese con esta empresa.

Paypal permite realizar transferencias entre usuarios con correo electrónico cobrando un coste por transferencia y permite realizar pagos de productos o servicios a vendedores incluyendo un coste proporcional al importe. Los costes dependen en gran medida de la moneda con que se pague, de la opción de pago, el país del emisor y del receptor y el importe de la compra. Paypal opera en más de 190 mercados en todo el mundo permitiendo a sus usuarios intercambiar dinero entre más de 26 monedas diferentes.

Estrategia de expansión de Paypal

La empresa llevó a cabo una estrategia en tres fases para expandir su negocio a todo el mundo. En la primera fase, se esforzaron por ser la opción de pago preferida para la mayoría de los usuarios de eBay a través de una campaña de marketing agresiva en la que regalaban diez dólares por nueva cuenta. Además, Paypal ha sido un servicio muy adecuado tanto para vendedores que no disponen de pasarela de pagos para cobrar de tarjetas de crédito o débito como para compradores que no poseen tarjeta de crédito para comprar en internet debido a su bajo historial crediticio y con una cuenta Paypal evitan divulgar los números de su tarjeta de crédito.

En la segunda fase que comenzó en el 2000, la empresa ya ampliamente conocida por los usuarios de eBay se propuso afinar el modelo de negocio ya que era insostenible al cobrar sus servicios a un coste bastante bajo. En esta fase se expandirían para abarcar los usuarios de eBay fuera de Estados Unidos.

La última fase de la estrategia de Paypal se llevó a cabo fuera del ámbito de eBay. Se basó en aumentar el número de usuarios ofreciendo sus servicios para diferentes plataformas. A finales de 2003, se creó una nueva unidad de negocio: Merchant Services para proporcionar sus servicios a pequeños y grandes vendedores a través de comercio electrónico. Estos servicios para comerciantes contaban con las siguientes características:

[4]https://www.paypal.com/es/webapps/mpp/about

1. Disminuir los costes por transacción de grandes volúmenes.
2. Animar a sus usuarios a atraer a nuevos usuarios con recompensas más suculentas.
3. Convencer a otras compañías de pasarelas de pagos como Cybersource a incluir Paypal entre sus ofertas a comerciantes por internet.
4. Contratar un nuevo departamento de ventas para adquirir grandes mercados como Dell, iTunes de Apple o Yahoo! Stores que alojaba a miles de comerciantes.
5. Reducir costes en micropagos.
6. Lanzar Paypal Mobile que permite a sus usuarios realizar compras a través de SMS.

Paypal para los compradores

Las principales ventajas que ofrece Paypal para los compradores son:

1. El pago que realiza el comprador es recibido al instante por el vendedor.
2. Seguridad ya que el vendedor no tiene acceso a los datos bancarios del comprador.
3. Confianza: Paypal es la pasarela de pagos más conocida y usada en internet. Se ha convertido en un servicio de confianza total para el usuario.
4. Flexibilidad ya que el comprador elegirá cómo prefiere pagar: Cuenta bancaria o tarjeta de crédito.
5. Comodidad: Los datos bancarios son introducidos sólo una vez en la pasarela de pagos.
6. Protección: Si el producto comprado no es el esperado, Paypal intentará hacer llegar a un acuerdo a las partes.

Paypal para los vendedores

La tarifa estándar de Paypal suele ser de un 3.4% del importe de la transacción además de 0.35 € por transacción. Conforme el volumen de ventas a través de Paypal se incrementa, el porcentaje que adquiere Paypal por la transacción se reduce. Las características que ofrece Paypal para los vendedores son:

1. Proporciona diferentes formas de pago al comprador para que pague por sus productos.

2. Posibilidad de elegir entre diferentes formas de integración y de cobro con Paypal. Es posible recibir pagos enviando una factura por correo electrónico o incluir sencillos botones para iniciar el pago con Paypal en la web.

3. Paypal ofrece un sistema llamado Pagos Express para reducir los pasos del proceso de compra -conocido como conversión porque transforma un usuario normal en cliente- evitando así el abandono del cliente antes de finalizar el proceso.

Soluciones de Paypal

Pago exprés

Servicio con el que el vendedor podrá incrustar un botón Pagar con Paypal en su página web que redirigirá a la pasarela de pagos con la que se podrá comprar el producto en 3 clicks. Permite al comprador comprar con tarjeta o cuenta de Paypal. Añadir el servicio de Paypal permite al vendedor recibir pagos de 25 divisas posibles.

Pago estándar

El servicio de pago estándar extiende las ventajas del pago exprés. Si el vendedor no cuenta con página web, cabe la posibilidad de que el comprador reciba un email con la factura electrónica y un enlace que le redirija a la pasarela de pagos y pueda pagar el producto. Además, adquiriendo Pago estándar, el vendedor podrá utilizar Paypal como medio de cobro para la venta de sus productos en eBay.

Pasarela integral

El servicio Pasarela integral extiende las ventajas del pago estándar. Con Pasarela integral, el vendedor puede personalizar la pasarela de pago para adaptarla al diseño de la página web del vendedor. La característica más importante de Pasarela integral es que el vendedor recibirá protección para sus ventas en caso de que el comprador haga una devolución de cargo. Una devolución de cargo se puede producir porque el comprador solicita la cancelación del pago a la entidad emisora de la tarjeta. Esto es muy común ya que se realiza justo cuando se advierte el robo de la tarjeta. Paypal solicita el cumplimiento de varios requisitos y un cierto historial de transferencias para proteger al vendedor de las pérdidas potenciales.

Para disfrutar de estos servicios, el vendedor afrontará un coste de 15 € mensuales por el servicio de Paypal Pasarela integral.

Otros productos de Paypal

Paypal Here

Es una aplicación móvil y un lector de tarjetas en dos modelos diferentes: Para Europa, el lector incorpora un teclado numérico para introducir el número PIN de la tarjeta, en Estados Unidos es sólo el lector con forma de triángulo. La aplicación móvil permite realizar cobros de la tarjeta que se depositan en la tarjeta del vendedor asociada a su cuenta de Paypal. La comisión por transacción es del 2.7% que se puede reducir a 1,7% si se realizan compras con dicha tarjeta.

Paypal Beacon

Paypal Beacon es un dispositivo que se conecta a través de USB e implementa la tecnología Blutooth Low Energy para facilitar el proceso de pago en un establecimiento. La tecnología BLE también conocida como Bluetooth 4.0 permite la transmisión de datos entre dispositivos a menos de 50 metros de distancia y su consumo de energía es muy bajo lo que convierte a esta tecnología en una vía muy adecuada para proporcionar información del entorno a dispositivos que se acercan. BLE es la alternativa a NFC, apuesta de Google cuyo alcance es apenas de 20 centímetros de radio.

Cuando el comprador con la aplicación de Paypal instalada (pero no necesariamente abierta) entra en un establecimiento con este dispositivo, el dispositivo móvil recibe una notificación para que el usuario sepa que puede utilizar dicha tecnología para comprar y permita ser cobrado posteriormente. Esta notificación hace que el móvil vibre o suene de manera que el comprador no tiene que coger el móvil que es el objetivo del producto. Para mayor seguridad, empleados de Paypal se encargarán de confirmar la información de los establecimientos.

El vendedor a partir de ese momento podrá cargar el importe de los productos al comprador necesitando sólo la confirmación verbal del comprador siempre y cuando la tecnología BLE detecte el dispositivo móvil del comprador cerca, confirmando así su proximidad física. La foto del comprador en el terminal punto de venta del comprador permite al vendedor identificarle y cargar el importe de la compra.

La aplicación móvil permite que el vendedor notifique ofertas al comprador y oferte promociones personalizadas. Por otra parte, el comprador podrá comprar productos desde la aplicación fuera del establecimiento y pasarse a recogerlos más tarde.

Authorize.Net

Autorhize.Net es una pasarela de pagos que fue fundada por Jeff Knowles en 1996. Ha tenido un enorme crecimiento: En el año 2004 fue comprada por Lightbridge que pagó 82 millones de dólares. Posteriormente fue comprada en 2007 -en menos de cuatro años- por Cybersource, adquisición por la que se pagó 565 millones de dólares. En 2011, Cybersource fue adquirida por Visa.

En la página web de Authorize.Net incluyen una sección How it works[5] con un diagrama de funcionamiento de la pasarela de pagos muy parecido al proceso descrito anteriormente.

El cobro del servicio por parte de Authorize.Net es ligeramente diferente a Paypal. El cobro y los servicios que ofrece también varían según la localización geográfica. Para Europa sólo ofrecen la tarifa Only gateway. Sin embargo, para Reíno Unido, Australia, EE. UU. y Canadá, el abanico de servicios es mayor. El servicio Only gateway no tiene coste de instalación. El coste por transferencia es de 0.1 libra a partir de las 350 transacciones con la pasarela de pagos. A diferencia de Paypal. Authorize.Net no cobra una parte proporcional de la transferencia.

Adicionalmente, se pueden contratar otros servicios de Authorize.Net como son:

1. Advanced Fraud Detection Suite por $9.95 al mes.
2. Automated Recurring Billing por $10 al mes.
3. Customer Information Manager por $20 al mes.
4. eCheck.Net que permite al comprador adquirir el producto por transferencia bancaria salvando la necesidad de una tarjeta. La cuota de este servicio depende de las necesidades del vendedor y es necesario contactar con el departamento de ventas de Authorize.Net.

Authorize.Net ha desarrollado un programa de afiliación de distribuidores para propagar el uso de la pasarela de pagos y conseguir comisión por transferencias. Así por ejemplo, del coste de 0.15 libras por transacción con Authorize.Net, el distribuidor obtendría 0.05 libras.

[5] http://www.authorize.net/resources/howitworksdiagram/

Amazon Flexible Payments Service

La empresa Amazon, creó la filial Amazon Payments en 2007. Amazon Payments se encarga de la infraestructura de las transferencias entre Amazon, sus proveedores y sus clientes. La enorme masa de usuarios que utilizaba Amazon para comprar productos les llevó a crear su propia pasarela de pagos Amazon FPS.

La sección Frequently Asked Questions[6] del sitio web de Amazon FPS contiene la información necesaria para que vendedores conozcan el servicio y decidan integrar Amazon FPS en sus tiendas electrónicas.

La pasarela de pagos Amazon FPS acepta las tarjetas de crédito Visa, MasterCard, American Epxress, Discover, Diners Club y JCB. También se puede pagar con dinero electrónico de Amazon y sólo en Estados Unidos es posible pagar a través de transferencia bancaria. Todas las transacciones se llevan a cabo con dólares americanos.

Amazon por su servicio Amazon FPS sólo cobra tasas por transferencia. Éstas dependen del importe de la transferencia, estableciendo rangos y distinguiendo según la forma de pago. Las siguientes tasas son para EE. UU. Es posible que se apliquen diferentes en España.

Si se paga con tarjeta de crédito y el importe es mayor o igual a $10, la tarifa es del 2.9% del importe y $0.3. Si el importe es menor a $10 la tarifa es 5% y $0.05. En cambio, si se paga con transferencia bancaria la tarifa por transferencia es 2% y $0.05.

La opción más económica es utilizar dinero de Amazon (Amazon Payments Balance) para pagar por transferencia ya que si el importe es mayor a $0.05, la tarifa es 1.5% + $0.01, si el importe es menor la tarifa es 20% y $0.0025.

Amazon FPS tiene en cuenta los vendedores con un alto volumen mensual de transacciones con tarjeta de crédito. Para el volumen de pago entre $3K y $10K, Amazon aplicará por transacción un coste de 2.5% y 0.3. Entre $10K y $100K el porcentaje se reduce a 2.2 y cuando se alcancen los $100K el porcentaje del importe por transferencia se reduce a 1.9%.

Google Wallet

Google Wallet es la pasarela de pagos que proporciona la empresa Google en la actualidad. En junio de 2006, Google lanzó Checkout, el sistema de pagos que ha dejado de

[6] http://aws.amazon.com/es/fps/faqs/#a2

estar disponible desde noviembre de 2013 y ha sido sustituido por Google Wallet. Oficialmente, Google Wallet fue presentada en mayo de 2011.

Desde agosto de 2012, Google Wallet permite añadir a la cuenta de Google diferentes tarjetas de crédito o débito incluyendo Visa, MasterCard, American Express y Discover.

Google Wallet está muy enfocada al pago desde dispositivos móviles y permite que el comprador almacene tarjetas de fidelización, descuentos y ofertas aprovechando la movilidad y evitando engorrosas tarjetas de fidelización físicas. Desde el dispositivo móvil con Android y tecnología NFC se pueden realizar compras en establecimientos físicos que acepten pagos a través de la pasarela Google Wallet, aunque por el momento la aplicación de Google Wallet para pagar en establecimientos sólo se puede usar en Estados Unidos.

Los dos productos o características principales de Google Wallet son Instant Buy y Objects. Instant Buy es el software, el hardware y el proceso que posibilita el pago con la pasarela de pagos de Google. Wallet Objects es el software que permite al desarrollador crear objetos que serán las ofertas, descuentos, promociones y tarjetas de fidelización del comercio.

A finales de 2013, Google ha enviado a los usuarios estadounidenses verificados de Wallet la tarjeta Google que no es más que una tarjeta Mastercard con la que se puede pagar en tiendas físicas de los Estados Unidos con dinero Wallet Balance.

La empresa Google se está esforzando por renovar el proceso de compra de los clientes y aumentar la masa de clientes de los compradores con sistemas de fidelización y promociones. Sin embargo, **entre los casos de uso para los que su pasarela se ha diseñado no se encuentran los mercados basados en su tecnología**. Los tipos de usuarios de Google Wallet son comprador y vendedor pero no encaja una tercera persona que proponga una plataforma que incluya la pasarela de pagos de Google en la que los compradores y los vendedores son otros.

La sección Frequently Asked Questions incluye un apartado donde indica la tarifa por el uso de la pasarela Google Wallet. Para el comprador, hacer una compra tanto a través de internet como en una tienda física no supone ningún cargo. Ingresar y retirar fondos entre la cuenta bancaria y Wallat Balances tampoco supone ningún cargo. Sin embargo, si se usa la tarjeta de crédito o débito para obtener Wallet Bllance sí que se cobra un 2.9% de la transferencia y $0.3.

2.2.2. Otros servicios de pago

LifeLock Wallet

LifeLock Wallet[7] es una aplicación para dispositivos móviles con Android o iOS que permite al usuario almacenar la información de los documentos que podría guardar en la cartera, tanto tarjetas de crédito, débito, permiso de conducir, tarjetas de descuento, etc. Con el objetivo de sustituir la cartera y garantizando seguridad en cuanto al almacenamiento de la información han creado un mercado para usuarios de la plataforma LifeLock Wallet permitiendo el pago a través de códigos QR con la información de la tarjeta de crédito o débito. La aplicación móvil permite al comprador administrar y elegir la tarjeta con la que quiere realizar el pago y evita al vendedor almacenar información financiera del comprador.

Como pasarela de pago, LifeLock Wallet también ofrece una vía a través de la aplicación para discutir reclamaciones sobre los pagos que se han efectuado en la plataforma. En su versión Premium ofrece características adicionales a los compradores como administración y supervisión de los gastos escaneando los tickets de compra.

En diciembre de 2013, justo después de que Lemon haya sido comprada por LifeLock cuentan con más de tres millones de usuarios en cien países y como consejeros cuentan con cargos de grandes entidades financieras como citi, Bank of America, Visa y Paypal

iZettle

iZettle[8] es un servicio de pago que combina una aplicación Android o iOS con un lector de tarjetas de crédito o débito para que el vendedor pueda crear un catálogo de productos y cobre a través del lector de tarjetas que se conecta al dispositivo (teléfono móvil o tablet) a través de bluetooth. Las tarjetas con las que se pueden realizar pagos a través de iZettle son Visa, MasterCard y American Express.

El lector de tarjetas tiene un precio de 99 € y dispone de un teclado numérico para introducir el PIN, esta acción supone por sí misma la confirmación del pago con la tarjeta. Desde el punto de vista de la seguridad, sería deseable que la información financiera no circulase a través del dispositivo del vendedor. De la misma manera, sería deseable que el lector de tarjetas cubriese la mano del comprador mientras teclea el código PIN. No obstante, iZettle asegura que cumplen con los estándares EMV y PCI DSS de operatividad con tarjetas de crédito y débito.

En cuanto a las comisiones, iZettle pone en valor la simplicidad en el cobro y sólo cobran comisión por transacción que es del 2.75%. En Reino Unido la comisión es variable

[7] http://www.lemon.com
[8] https://www.izettle.com

dependiendo de los ingresos mensuales a través de la plataforma. Esta comisión varía del 2.75% hasta el 1,5% a partir de las trece mil libras.

Square

Square[9] es una empresa americana con el objetivo de facilitar la compra en comercios. La empresa proporciona de forma gratuita un lector de tarjetas que se conecta a través de la salida de audio del dispositivo móvil. El PIN es introducido directamente en el dispositivo móvil del vendedor donde está instalada la aplicación Square por lo que el problema en cuanto a la confidencialidad se acentúa con respecto a Kuapay. El vendedor también puede listar sus productos en la web de Square para vender a través de internet sin necesidad de comercio físico.

Es posible pagar a través de la plataforma Square con Visa, MasterCard, American Express y Discover aunque el servicio sólo está disponible en Estados Unidos, Canadá y Japón. Square cobra una comisión del 2.75% por transacción realizada sobre su plataforma con la excepción de los pagos realizados introduciendo manualmente los datos de la tarjeta a los que carga una comisión del 3,5% y 15 centavos. Los pagos son ingresados al día siguiente en la cuenta bancaria del vendedor.

Como segundo planteamiento estratégico tras ofrecer una solución a los vendedores facilitando el cobro, Square ha desarrollado una aplicación para facilitar el pago llamada Square Wallet. Los vendedores registran su comercio, localización y los productos en venta. El comprador a través de la aplicación en la que previamente ha registrado su tarjeta de crédito o débito, su nombre y su foto confirma que se encuentra en el comercio (en inglés: Check in). Desde ese momento, el vendedor puede vender el producto seleccionando desde su dispositivo móvil al comprador que identifica por el nombre y su foto. Aprovechando el sensor de geoposicionamiento del dispositivo móvil, Square Wallet se puede configurar para que el dispositivo móvil sea el que notifique al comprador cuando detecte que ha entrado en el área del establecimiento.

Otra aplicación desarrollada por Square es Square Cash con la que es posible enviar dinero a otras personas. Primero, el emisor envía un email al receptor del dinero incluyendo en el campo Cc cash@square.com. El emisor recibirá un email de Square con un enlace web a un formulario donde introducirá la información de su tarjeta. Posteriormente, el receptor recibe un email para que rellene la información de la tarjeta asociada a la cuenta bancaria en la que desea recibir el dinero. El envío de dinero es gratis.

[9] https://squareup.com/

Stripe

Stripe[10] es un servicio de pagos diseñado específicamente para desarrolladores y para ser integrada en proyectos software. Comprende un conjunto de librerías para una gran cantidad de lenguajes de programación y una API. Stripe proporciona la infraestructura necesaria para permitir realizar pagos a través de la aplicación con tarjetas Visa, MasterCard, American Express, Discover, JCP y Diners Club. El servicio de pago es completamente funcional en Estados Unidos, Canadá, Irlanda y Reino Unido, además en Australia, Bélgica, Francia, Alemania, Luxemburgo, Países Bajos y España se encuentra en fase beta.

En España, el conjunto de tarjetas aceptadas se reduce a las Visa, MasterCard y American Express. La única comisión que Stripe cobra es por transferencia recaudando un 2.9% del importe y 30 céntimos. Sin embargo, en España, recibir pagos en euros, libras y dólares pero la conversión supone una comisión del 2% que Stripe cobrará automáticamente.

SumUp

En la lucha por cubrir las necesidades de un punto de venta virtual, SumUp[11] es un producto español que también comercializa un lector de tarjetas, pero lamentablemente, sólo se puede utilizar con tarjetas MasterCard. No obstante, SumUp ofrece una forma alternativa de uso con la que se puede enviar un SMS al cliente con un enlace a la web de SumUp para realizar el pago desde su dispositivo móvil. Esta funcionalidad requiere que el comprador tenga conexión a internet en su dispositivo móvil pero mantiene los datos de la tarjeta entre comprador y servicio de pago sin pasar por el dispositivo móvil del comprador. A través del enlace a la web, el comprador puede introducir los datos de su tarjeta Visa o Mastercard.

Donde SumUp es más competitiva es en la comisión por transferencia entre sus usuarios. Esta es del 1,5% del importe de la transacción. Por otra parte, la empresa es completamente española, aspecto que en temas financieros es muy valioso sobre todo porque al final de la cadena del pago se encuentra una entidad financiera española donde se deposita el importe de la transacción.

Coin

[10] https://stripe.com
[11] https://sumup.es/

Coin[12] es un chip que simula una tarjeta que junto a la aplicación móvil permite al comprador realizar pagos en los TPVs que leen la banda magnética. En el dispositivo Coin se pueden almacenar hasta 8 tarjetas -de crédito, débito, fidelización...-, en cada momento se puede escoger pulsando un botón que incorpora el dispositivo la tarjeta con la que se desea pagar. Obviamente, Coin dispone de una pequeña pantalla que muestra los últimos 4 dígitos del número de la tarjeta para saber cuál se escoge. Por otra parte, dispone de conexión bluetooth para comunicarse con el dispositivo móvil. Esta conexión permite desactivar la tarjeta Coin o que el dispositivo móvil avise si se aleja una cierta distancia de la tarjeta. Se espera que Coin se ponga en funcionamiento en el verano de 2014. Por lo pronto, se puede comprar el dispositivo desde la página web de la empresa.

Por ahora, Coin no cumple el estándar EMV y no soporta la funcionalidad chip & pin que confirma el uso de la tarjeta de crédito o débito con el código PIN. Esto hará muy difícil su uso en Europa. Tampoco cumplen con el estándar PCI DSS. De hecho, la implementación de Coin hará complicado que se pueda cumplir:

Con un lector de bandas magnéticas (que también hay que comprar a la empresa Coin) se clonan los datos de la banda magnética de la tarjeta al dispositivo Coin. El código CVV se almacena tanto en la tarjeta (además se muestra por la pequeña pantalla) como en los servidores de la empresa Coin. El reglamento PCI DSS especifica que el código CVV sólo se puede usar en la transferencia y no se debe almacenar en ningún lugar. Otro aspecto de usabilidad interviene en la compra con Coin ya que el vendedor puede desconfiar de una tarjeta oscura y no permitir el pago con el dispositivo.

[12] https://onlycoin.com

2.2. Geolocalización en dispositivos móviles

Se atribuyen las siglas **LBS (Location Based Services)** a los servicios que aprovechan la información geográfica para enriquecerse y aportar mejores características e información a los usuarios. Otra denominación es Location-Dependent Information Services (LDIS).

Con la proliferación de dispositivos móviles se ha visto impulsado el desarrollo de aplicaciones que haciendo uso de la posición geográfica del dispositivo ofrecen servicios en base a dicha información. Estas aplicaciones contextualizan al usuario en el mapa y proporcionan servicios personalizados.

Las aplicaciones que proporcionan servicios basados en localización se ven respaldadas por los sistemas de localización que son las infraestructuras que acumulan la información geolocalizada, la relacionan y obtienen de ella más información para proporcionar a los usuarios del sistema los servicios basados en posicionamiento a través de la aplicación. Los componentes de un sistema de localización son:

- **Dispositivo móvil:** Dispositivo del usuario en el que se ejecuta la aplicación que proporciona los servicios.
- **Aplicación LBS:** Aplicación instalada en el dispositivo móvil que recoge la posición del dispositivo, la comunica al sistema de posición geográfica y recibe o hace uso de los servicios basados en posicionamiento.
- **Proveedor de posicionamiento:** Proporciona la posición geográfica del dispositivo móvil. Puede ser un sensor -como el GPS-, un conjunto de sensores o una aplicación como la aplicación que estima la posición del dispositivo haciendo cálculos trigonométricos entre las antenas de telefonía y los puntos de acceso Wifi.
- **Sistema operativo:** El sistema operativo del dispositivo móvil generalmente controla los proveedores de posicionamiento a bajo nivel para reducir la complejidad del uso de éstos y aplicar buenas prácticas como por ejemplo optimizar el consumo de batería.
- **Infraestructura de red:** El hardware y software necesario para que la aplicación se comunique con el sistema de información geográfica. Generalmente, internet haciendo uso del protocolo IP ya a través de redes Wifi o móviles como 3G.
- **Sistema de información geográfica** o **GIS** por sus siglas en inglés es el servidor que recoge la información geográfica enviada por la aplicación, procesa esta información y obtiene nueva información en base a otros datos recabados por otras aplicaciones. Finalmente, se comunica con la aplicación u otras aplicaciones para proporcionar los servicios basados en posicionamiento.

El siguiente apartado trata sobre aplicaciones muy conocidas que ejemplarizan la integración y el uso de geolocalización en dispositivos móviles. Para finalizar con el estado de arte de la geolocalización, se exponen algunas herramientas, librerías y servicios que se pueden integrar o usar desde aplicaciones móviles para convertirlas en aplicaciones basadas en posicionamiento geográfico.

2.2.1. Aplicaciones y servicios basados en geolocalización

Dos de los servicios más conocidos que hace uso intensivo de la información de geolocalización del usuario son Foursquare y Waze. Los dos servicios fueron desarrollados principalmente para dispositivos móviles y ambos aprovechan las características sociales disponibles desde internet y enriquecen su contenido animando a los usuarios a colaborar para el bien de la plataforma.

Foursquare

Foursquare[13] es un servicio basado en geolocalización y pensado para su uso en dispositivos móviles creado en el año 2009 por Dennis Crowley y Naveen Selvadurai.

El caso de uso principal para un usuario de Foursquare se conoce como **hacer check-in** y consiste en confirmar que se encuentra en un determinado punto geográfico como puede ser un edificio, un aeropuerto, etc. Cuando el usuario quiere hacer check-in la aplicación le muestra una lista de lugares cercanos a su posición, éste elije en qué lugar se encuentra y tiene la libertad de publicar en redes sociales como Twitter y Facebook que ha hecho check-in de ese lugar. Se puede configurar la publicación automática en redes sociales pero resulta realmente molesto para los usuarios que continuamente reciben esta información. Se han realizado estudios en busca de los motivos que llevan a unos usuarios dejar de seguir a otros en redes sociales[14] y se ha determinado que una de las principales causas es la comunicación automática de los check-ins a través de la red social.

Por el hecho de hacer check-in en determinados lugares y con determinada frecuencia, el usuario consigue medallas e incluso títulos como ser el alcalde de un lugar. Además, el equipo de Foursqaure puede otorgar a un usuario el estado de **superusuario** por sus contribuciones y el uso de la aplicación. Siendo superusuario, es posible revisar lugares y editar la información para corregirla ya que otros usuarios han podido cometer errores al introducir nuevos lugares. De esta manera, la información geoposicionada es con el paso del tiempo mayor y más rica en contenido.

[13] https://es.foursquare.com/
[14] Fragile Online Relationship: A First Look At Unfollow Dynamics in Twitter

La aplicación de mecánicas de grupo en la que los usuarios se ven incentivados a realizar ciertas tareas como la revisión de una nueva localización para conseguir bonificaciones como las medallas compitiendo con otros usuarios se conoce como **gamificación** y permite que plataformas sociales como Foursquare almacenen información geográfica veraz y actualizada.

Por otra parte, la plataforma aprovecha la información geolocalizada para aportar al usuario otros beneficios como la posibilidad de elaborar listas de tareas por realizar cuyas tareas estén geolocalizadas y el usuario sea notificado con un recordatorio de la tarea cuando se encuentre cerca del lugar. También es posible dejar sugerencias en los lugares en los que se hace check-in para otros usuarios cuando se encuentren en el lugar.

Con toda la información geolocalizada y las sugerencias de los usuarios de Foursquare, el equipo ha desarrollado un buscador de lugares que permite realizar búsquedas de lugares por categorías como son Comida, Artes, Vida nocturna, Aire libre dentro del sector rectangular del mapa que el usuario está consultando.

Waze

Waze[15] es una aplicación móvil disponible para multitud de plataformas que proporciona información en tiempo real del tráfico y guiado en rutas. Waze es el servicio de una empresa con el mismo nombre que se fundó en el año 2008 en Israel. Siendo una empresa muy pequeña con un producto muy característico fue comprada en junio de 2013 por Google en sintonía con la compra de pequeñas empresas con productos muy enfocados y personal muy especializado cuya integración ayudará a enriquecer los propios productos.

La aplicación mantiene un grafo de vías continuamente actualizado por la comunidad de usuarios ya que el dispositivo móvil proporcionará información geográfica a la plataforma sobre la ruta que está tomando el vehículo. Además, la empresa otorga un gran valor a la comunidad, animándola a editar las vías para ayudar a otros usuarios.

Más allá de las características usuales que un sistema de guiado tiene, los usuarios pueden informar en tiempo real sobre incidencias en el tráfico como retenciones o desperfectos en la vía lo que se convierte en una información de gran valor para otros usuarios que podrán tomar una ruta alternativa.

La aplicación incluye otros servicios como un sistema de proximidad para encontrar la gasolinera más cercana con los precios en carburantes más económicos que aportan los

[15] https://www.waze.com/es/

usuarios. Este servicio fue añadido en 2012. Previamente, en el año 2011, se había integrado en la aplicación la posibilidad de que los usuarios pudiesen informar sobre puntos de interés y eventos locales como ferias y manifestaciones.

También desde 2012, con el objetivo de adaptar otros modelos de negocio en el servicio Waze, la empresa ofrece a anunciantes una interfaz web para gestionar publicidad basada en la ubicación geográfica. Esta publicidad aparece en el mapa de la aplicación a través de un pequeño icono. Para agencias de noticias, Waze proporciona una interfaz web desde la que pueden obtener informes del tráfico en tiempo real y alertas generadas por el servicio sirviéndose de los datos que proporcionan sus usuarios.

La compra de Waze por parte de Google también ha beneficiado al propio servicio sirviéndose de las imágenes de satélite de Google Maps y de las imágenes a pie de calle de Google Street View. Por otra parte, el equipo de Google Maps ha integrado la información del tráfico en tiempo real generada en la plataforma Waze en la aplicación móvil de mapas Google Maps.

2.2.2. Frameworks, librerías y herramientas

En esta sección se han contemplado tres piezas de software centradas en servicios de geolocalización que se pueden utilizar desde aplicaciones en dispositivos móviles. La primera es una librería para integrar en proyectos Java, la segunda es un PaaS especializado en el almacenamiento de información geográfica y la visualización de mapas y la tercera es la aplicación de un PaaS para almacenamiento de información a partir del cual se ha desarrollado una librería menos funcional que la segunda pero muy práctica.

GeoTools y Geomajas

GeoTools[16] es una librería diseñada para el apoyo en sistemas que tratan con información espacial –conocidas como GIS, Geographic Information Systems-. Esta librería proporciona bastantes implementaciones de las especificaciones redactadas por el Open Geospatial Consortium (OGC). Además, GeoTools es un proyecto Open Source cuya licencia es LGPL.

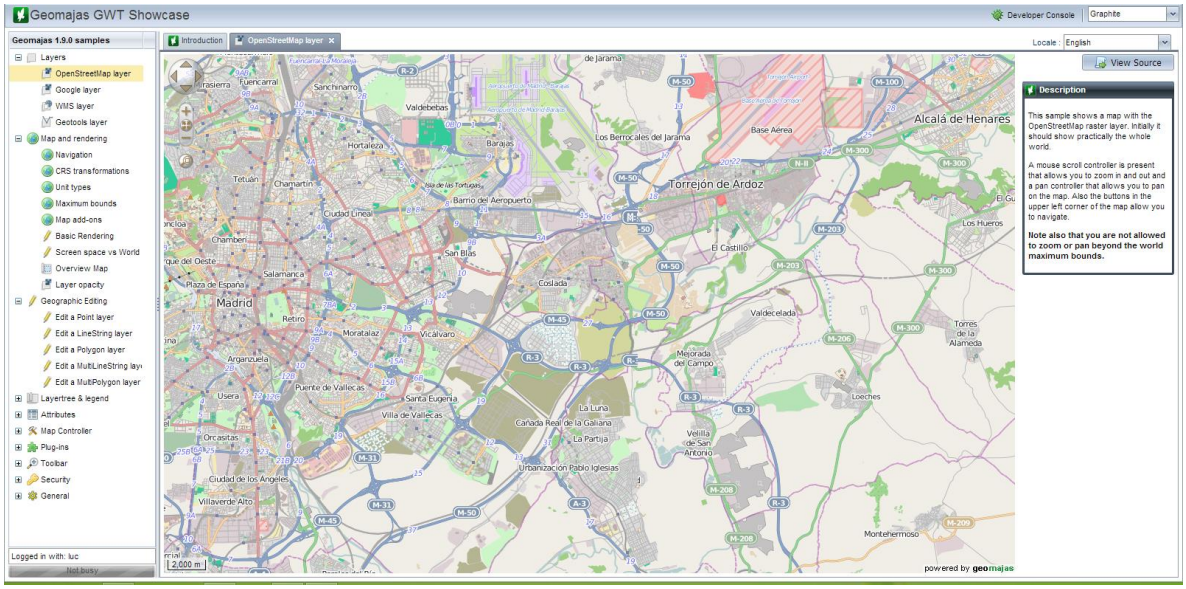
Como librería, GeoTools proporciona el marco de desarrollo para utilizar un sistema de coordenadas y herramientas para convertir fácilmente este sistema de coordenadas en otros más útiles para determinadas tareas. Soporta la especificación OGC Styled Layer

[16] http://geotools.org/

Description para la definición de simbología sobre los mapas y la Filter Encoding Specification para aplicar filtros que dependan de información geoespacial.

La fundación Open Source Geospatial Foundation está desarrollando otro proyecto denominado Geomajas[17] que es un framework de desarrollo de aplicaciones cliente-servidor en las que se representa y es posible editar información geográfica. El proyecto está especialmente diseñado para cubrir las exigencias en cuanto a seguridad y escalabilidad. Por supuesto, aplican las recomendaciones que la misma fundación define para un sistema de información geográfica.

La funcionalidad de Geomajas puede ser ampliada usando plugins. En la aplicación cliente, el desarrollo se apoya en el framework GWT para desarrollar vistas con Java y compilan a Javascript ejecutándose en el navegador del usuario y manteniendo un buen rendimiento.



Captura de pantalla de Geomajas

En esta captura de pantalla de la aplicación cliente Geomajas dentro del navegador, se ha seleccionado la capa de renderizado del mapa de OpenStreetMap. Como se aprecia en la parte superior izquierda, otra capa disponible es la de Google Maps. Téngase en cuenta que se está usando la aplicación para representar información geográfica y editarla. Sin embargo, la arquitectura del framework Geomajas se ha desarrollado para soportar otros casos de uso como el de intercambio de información geográfica entre aplicaciones clientes y aplicación servidor, que es una característica muy útil para el desarrollo de aplicaciones móviles.

[17] http://www.geomajas.org/

CartoDB

CartoDB[18] es una plataforma como servicio dedicada específicamente a información geográfica. Este producto que fue lanzado en 2012, consiste, a grandes rasgos, en una base de datos optimizada para el almacenamiento de datos geográficos y el software que permite al usuario almacenar dicha información, analizarla y visualizar los datos en un mapa. Desde el punto de vista del desarrollador, CartoDB es la plataforma en la que se puede almacenar información geográfica, analizarla y elaborar mapas para desarrollar servicios basados en posicionamiento.

En primer lugar, desde la aplicación web se pueden introducir datos con referencias geográficas ofreciendo diversas opciones de importación en tablas que almacena el propio servicio. A partir de información, se pueden crear visualizaciones de los datos contando con amplias opciones de personalización de los mapas. El mismo equipo ha desarrollado una librería en Javascript llamada Torque para visualizar los datos geográficos sobre un mapa animado en función de la marca de tiempo de estos datos. Creadas las visualizaciones de los datos sobre los mapas, la aplicación web permite publicarlas y compartirlas.

Por otra parte, CartoDB permite simplificar el desarrollo de aplicaciones basadas en información geográfica ya sean web o móviles proporcionando en sus propios servidores el respaldo y almacenamiento de dicha información, el renderizado de los mapas y funcionalidad como seguimiento de la posición o cálculo de distancias. Para aplicaciones web, se ha desarrollado una librería denominada cartodb.js en Javascript desde la que se puede usar las funcionalidades de CartoDB.

Alternativamente, para aplicaciones móviles, la comunicación con CartoDB se efectúa a través de una API REST en la que se incluyen consultas de datos al estilo del lenguaje SQL También ofrecen una API para la visualización de mapas llamada Maps API que permite visualizar los datos sobre mapas de Google, Bing o cualquier otro servicio.

```
http://username.cartodb.com/api/v2/sql?
q=SELECT cartodb_id, the_geom, the_geom_webmercator, name FROM
wifi_hotspot_locations ORDER BY the_geom <-> ST_SetSRID(ST_MakePoint(-
73.98056030, 40.76390128),4326) ASC LIMIT 40&format=GEOJSON
```

Este es un ejemplo de petición GET a la API de CartoDB en el que se incluye una consulta para obtener datos en formato JSON directamente de las tablas alojadas en el servicio. Las bases de datos usadas para respaldar el servicio son PostgreSQL con la

[18] http://cartodb.com/

extensión PostGIS que permite almacenar objetos con información geográfica y hacer consultas en la base de datos incluyendo operaciones geoespaciales.

La aplicación de backend que recibe las peticiones de la aplicación cliente y gestiona la base de datos es código libre pudiéndose encontrar en Github desde donde se puede descargar para instalarlo localmente y gestionar personalmente un servidor CartoDB para la aplicación.

Geofire

Firebase[19] es un servicio que permite almacenar y obtener datos a través de una API. Cumple las funciones de backend para la mayoría de aplicaciones móviles y web que no necesitan procesamiento centralizado. Para aprovechar este servicio y ejemplarizar casos de uso, el equipo de Firebase ha desarrollado Geofire[20] que es una librería Javascript desarrollada por el equipo de Firebase para utilizarla contra su servicio.

La librería Geofire para las aplicaciones móviles cubre dos casos de uso relacionados con geolocalización: El primero lo denominan búsqueda geolocalizada que consiste en obtener elementos en un determinado radio. El segundo caso de uso consiste en asegurar que los elementos geolocalizados son actualizados constantemente para proveer una posición geográfica correcta. El desarrollador tiene disponibles las siguientes funciones para utilizar dichos casos de uso en su aplicación:

- insertByLocWithId(latLon, id, data, [onComplete]) es la función para almacenar el identificador de un elemento data geolocalizado. Permite consultar dicho elemento posteriormente a través de su posición o de su identificador. La función onComplete es opcional y se ejecuta al obtener respuesta satisfactoria del servicio.
- updateLocWithId(latLon, id, [onComplete]) permite actualizar la posición geográfica del elemento identificado con id.
- onPointsNearLoc(latLon, radius, callback) es la función para encontrar los elementos que están dentro de radius en kilómetros desde la posición latlon. La respuesta a esta llamada no devuelve nada pero se devuelve un array con elementos en el callback. Cada vez que este array sea modificado, el servicio Firebase llamará al callback definido.

No sólo se almacenan las coordenadas de la posición del elemento en el servicio Firebase, también se almacena su geohash[21]. El geohash es una cadena alfanumérica que se

[19] https://www.firebase.com/
[20] https://www.firebase.com/blog/2013-09-25-location-queries-geofire.html
[21] http://en.wikipedia.org/wiki/Geohash

obtiene a partir de la latitud y la longitud e identifica la posición geográfica. Esta cadena tiene dos propiedades muy adecuadas:

- Casi siempre, cuanto más largo es el prefijo (x primeros caracteres) que comparten dos geohashes, las posiciones geográficas se encuentran a menos distancia.
- Cuanto más larga es la cadena alfanumérica, la posición geográfica codificada es más precisa. Esto permite eliminar caracteres de la cadena disminuyendo la precisión a favor de la reducción del tamaño de la cadena.

Este código alfanumérico fue desarrollado por Gustavo Niemeyer y la conversión de las coordenadas supone intercalar la representación binaria de latitud y longitud y posteriormente codificar el resultado a Base32.

Geofire aprovecha los geohashes para la detección de elementos cercanos, haciendo búsquedas de prefijos cuya longitud depende del radio de la distancia y de la resolución del zoom en una representación sobre un mapa.

2.3. La moneda electrónica Bitcoin

Bitcoin es una moneda electrónica descentralizada y el protocolo para preservar las características de dicha moneda que fue propuesto en un artículo técnico[22] en el año 2008 bajo el pseudónimo de Satoshi Nakamoto. Los símbolos de la moneda son BTC y ฿.

Con este algoritmo se propone una nueva solución al problema de los pagos por internet que es el gasto doble. Actualmente, es necesaria la confianza en un intermediario que valida la transacción, modelo de negocio de los servicios de pago y de las entidades financieras. Satoshi Nakamoto o el grupo de investigadores bajo ese pseudónimo utilizan la arquitectura P2P y las técnicas criptográficas para evitar la necesidad de confianza y permitir transacciones entre dos personas que se verifican por toda la red.

Utilizar Bitcoin conlleva varias ventajas. La más conocida es el anonimato ya que el usuario puede decidir el grado de anonimato con el que realizar un pago. Las transferencias son públicas pero es libertad del usuario relacionar la dirección de la transferencia consigo mismo o no.

Por otra parte, al ser un protocolo distribuido sin necesidad de intermediarios, las transacciones se realizan de manera instantánea y no hay que sufrir comisiones de terceros que proporcionen seguridad a la transferencia. Con el protocolo Bitcoin, cada usuario tiene la libertad de administrar la seguridad de sus carteras electrónicas de manera fácil sin requerir servicios externos como bancos que protejan sus bienes económicos.

Además, debido a la naturaleza de sus transacciones, los pagos son irreversibles. Ningún usuario puede cancelar un pago a mitad como ocurre en otros servicios web. Si se desea recuperar el dinero se requerirá una transacción en sentido contrario.

En cuanto a moneda se refiere, Bitcoin es escaso lo que conlleva que su valor aumentará conforme se extraen bitcoins y se ponen en circulación. El hecho de que sea escaso y no se pueda aumentar su cantidad, permite que se conozcan de antemano las deflaciones e inflaciones que sufra la moneda y que su valor no caiga por intervención humana.

La moneda tiene también sus inconvenientes. Como toda buena herramienta, atrae tanto a usuarios honrados como usuarios que la utilizan para fines ilegales. Lamentablemente, Bitcoin ha conseguido mucha popularidad a partir de sucesos relacionados con ilegalidades cometidas por usuarios.

[22] Bitcoin: A Peer-to-Peer Electronic Cash System en http://bitcoin.org/bitcoin.pdf

Actualmente, la masa monetaria no es lo suficientemente grande como para evitar que sea afectada por el mercado. En noviembre de 2013 buenas noticias provenientes de Estados Unidos y China hicieron que el cambio de Bitcoin a Dólares llegase a $ 1000 por Bitcoin. Posteriormente, a raíz de noticias negativas y de la propia naturaleza de una moneda cuyo valor depende fuertemente del mercado, volvió a casi la mitad del valor alcanzado en el pico. Cuando la masa monetaria en circulación sea mucho mayor, el cambio de Bitcoin no se verá tan afectado.

Las fluctuaciones de la moneda no la convierten en mala moneda para transferencias y para aprovecharse de las ventajas comentadas anteriormente. Sin embargo, no sería recomendable por el momento, el uso de Bitcoins para obtener rentabilidad ni para especulaciones. Cabe aclarar que en España el uso de Bitcoin no es ilegal, pero tampoco está regulado. No hay noticias hasta la fecha de que pretenda ser regulado.

Una buena forma de introducirse en el protocolo BItcoin es conocer primero qué elementos intervienen en la red distribuida que forma el protocolo. Posteriormente se explica cuál es el procedimiento de extracción de bitcoins.

Nodos y nodos generadores

Un nodo es la ejecución de una aplicación en la que se ha implementado el protocolo Bitcoin total o parcialmente. Se denomina nodo porque el dispositivo pasa a formar parte de la red Bitcoin que es una red distribuida. Hay implementaciones con las que se puede descargar la cadena de transacciones completa, que puede demorarse varios días, y otras con las que se puede minar bitcoins. Los nodos que ejecutan implementaciones del protocolo Bitcoin para minar monedas se conocen como nodos generadores.

Direcciones

La dirección Bitcoin es la dirección necesaria para enviar un pago o para recibirlo. Concretamente, la dirección es una secuencia alfanumérica de 33 caracteres que se obtiene a partir de un par de claves pública-privada en el nodo de Bitcoin. Cada carácter de esta secuencia está codificado en Base58 que comprende los 64 caracteres de Base64 exceptuando 0, O, I, L, + y /. Se excluyen estos seis para evitar que haya confusiones entre caracteres en una dirección Bitcoin. Además, las direcciones contienen 32 bits de checksum. El nodo puede crear tantas direcciones como necesite, éstas no incluyen información sobre el nodo ni necesitan de la intervención de otros nodos para ser creadas. Para generar una dirección primero hay que generar un par de claves pública-privada.

La dirección Bitcoin es la dirección necesaria para enviar un pago o para recibirlo. El nodo utiliza la clave privada de alguna de sus pares de claves criptográficas almacenadas en la cartera para autorizar el pago sólo a otro nodo de la red.

Transacciones

Una transacción es la transferencia de una determinada cantidad de bitcoins de un usuario a otro. Para crear una transacción, el nodo emisor firma con su clave privada la cantidad, el identificador de la transacción y la clave pública del usuario receptor. Entonces difunde la transacción en la red donde otros nodos validan las firmas criptográficas y el valor de la transacción. Por tanto, las transacciones son públicas para evitar el problema del gasto doble ya que no puede haber dos transacciones que transfieras los mismos bitcoins desde el mismo dueño. Si las hubiese, sólo sería válida la primera transacción, ningún nodo de la red aceptaría la segunda.

Cadena de bloques

Un bloque es un conjunto de transacciones con una marca de tiempo, un hash que lo identifica y el identificador del bloque anterior. Con este identificador en la red se mantiene una lista enlazada de bloques conocidos como la cadena de bloques que es para la red Bitcoin lo que la confianza es para el dinero fiduciario. Es así porque en la cadena de bloques están todas las transacciones válidas en ese momento. Si una transacción no está en la cadena de bloques, no es válida.

Minar bitcoins

Los nodos generadores son los encargados de la seguridad de la red aportando nuevos bloques válidos. Nótese que al aportar un nuevo bloque se confirma además la validez de los bloques anteriores. Para mantener una cadena de bloques honrada, los nodos generadores son recompensados por la creación de nuevos bloques.

Cuando un nodo realiza una transacción y la difunde a la red, ésta no es inmediatamente válida. En la red, un nodo generador recibirá la transacción, la añadirá a un conjunto de transacciones y tratará de crear un bloque de transacciones nuevo para añadirlo a la cadena de bloques. Esto se conoce como minado de bitcoins. Una vez que esté el bloque añadido a la cadena de bloques y la transacción esté incluida en él, la transacción será considerada por todos los nodos de la red como válida.

Para crear un nuevo bloque, el nodo generador tiene que encontrar un hash del bloque que contiene las transacciones que no es más que un código aleatorio que lo

representa. Sin embargo, ya que crear bloques nuevos y añadirlos a la cadena de bloques aporta seguridad a la red Bitcoin y esto se recompensa a los mineros de los nodos generadores, el hecho de encontrar un hash se dificulta de manera controlada por el protocolo. Por ello, esto se conoce como **proof of work**, denotando que se ha realizado esfuerzo computacional para crear el nuevo bloque.

Para encontrar el hash correcto se requiere realizar varios intentos por fuerza bruta de manera repetitiva buscando una solución no determinista lo que asegura que todos los nodos, independientemente de su capacidad de procesamiento, tienen la misma probabilidad de encontrar el hash que identifique al bloque y, por lo tanto, de crear un bloque. Así, se asegura que la generación de bloques se hace de forma distribuida y no haya nodos que la controlen, ya que en caso contrario, no sería complicado introducir de forma malintencionada transacciones falsas en bloques y enterrarlos en la cadena de bloques generando bloques nuevos.

El protocolo ha sido diseñado para que la frecuencia de creación de nodos siga una distribución de probabilidad de Poisson haciendo que se creen bloques aproximadamente cada 10 minutos y la dificultad para encontrar el hash se reajusta tras cada generación de 2016 bloques.

Como se mencionó anteriormente, el minero que crea un bloque es recompensado con un máximo de 25 bitcoins aunque la recompensa disminuye conforme las monedas son extraídas. En la red Bitcoin ocurre que los nodos generadores compiten entre sí para generar bloques con las transacciones que reciben y ganar la recompensa. La recompensa incentiva a los nodos generadores que convierte la red Bitcoin en una competición que contribuye en una red segura ya que los mineros se esfuerzan por verificar transacciones para posteriormente crear nuevos bloques.

La extracción de monedas tiene un máximo en cantidad que es 21 millones de bitcoins. La cifra es arbitraria, sin embargo, lo primordial es que haya un límite lo que convierte a la moneda en escasa como el oro. Asegurando la escasez de la moneda, se pueden controlar las inflaciones y las deflaciones por toda la red e incluso pronosticarse en el tiempo. Al contrario que el dinero fiduciario que no es escaso porque el papel sobre el que se imprime tiene menos valor que el que representa y, además, autoridades bancarias pueden incrementar o disminuir el valor a voluntad poniendo en circulación nuevas monedas y billetes.

Conforme la cantidad de monedas extraídas alcanza el máximo, disminuye la recompensa, en el futuro será necesario que se incentive la creación de nuevos bloques

ofreciendo una recompensa adicional por parte de los integrantes de la transacción. Uno o los dos integrantes pagarán la recompensa con el objetivo de acelerar la validación de la transacción. Si se recompensa al minero que firma un nuevo bloque, la primera transacción tendrá como destinatario el minero con la cantidad recompensada.

3. Diseño e implementación

3.1. Análisis del problema

La cantidad de información y servicios que se le ofrece al usuario es tan enorme que requiere sesgar tal ingente cantidad de estímulos para no provocar rechazo por parte del usuario. De todos los datos que se pueden recabar de un usuario para personalizar la información que recibe, la posición geográfica de éste es un dato muy valioso porque permite personalizar el servicio que se le ofrece atendiendo al lugar en el que se encuentra, la posible actividad que esté realizando y el entorno que le rodea.

Desde el punto de vista de un comerciante, las vías de comunicación y de presentación de productos al usuario se han incrementado, expandido e integrado tanto en la vida del usuario que se ha convertido en necesario ofrecer al usuario la posibilidad de comprar el producto en cualquier momento, en cualquier lugar y de una forma casi trivial e impulsiva.

Teniendo en cuenta las circunstancias anteriores, desde el punto de vista de la ingeniería del software, es primordial que un producto software tenga la capacidad de proporcionar información y servicios en base a su posición geográfica y además que se permita pagar por los productos de pago de la forma más fácil, rápida e intuitiva posible.

Los desarrolladores, por su parte, deberían percibir la obtención de la posición geográfica y la integración de formas de pago en una aplicación como funcionalidad intrínseca y fácil de proveer, para así centrarse en el problema global que intentan resolver.

En consecuencia, el paquete de servicios por proximidad sobre el que trata este documento trata de resolver la problemática de integrar una o varias pasarelas de pagos y la obtención y gestión de la posición geográfica de los usuarios en un desarrollo software.

3.2. Solución y decisiones tomadas

Atendiendo a las necesidades, el objetivo ha sido desarrollar un paquete con dos librerías: Una librería para convertir una aplicación en un servicio basado en posicionamiento y otra librería que permita integrar fácilmente pasarelas de pago en la aplicación.

Existen bastantes sensores para tomar la posición del usuario pero ningún dispositivo ha tenido tanto calado en la sociedad y en la vida de un usuario como el dispositivo móvil. En 2013 se estimó que el porcentaje de dispositivos móviles por persona en todo el mundo era

un 87%. Además el dispositivo móvil se ha convertido en el objeto que más tiempo se encuentra cerca de su propietario, esto convierte a los dispositivos móviles en la plataforma idónea sobre la que desarrollar servicios basados en posicionamiento y, por tanto, fue elegida como la plataforma sobre la que desarrollar el paquete de servicios por proximidad.

La decisión posterior a la plataforma es elegir sobre qué sistema operativo se desarrollará el paquete de servicios por proximidad. Más del 80% de la cuota de mercado de sistemas operativos móviles es propiedad de iOS, el sistema operativo de Apple, y Android, teniendo el primero una cuota de más del 55% a finales de 2013. Sin embargo, para el desarrollo de las librerías de geolocalización y de la integración de pasarelas de pagos, se ha elegido Android por varios motivos:

- Android es instalado en un rango muy amplio de dispositivos móviles desde aquellos con prestaciones más altas hasta los más modestos cuyo precio es más asequible.

- El código fuente de Android es abierto y para utilizar el SDK de Android no es necesario ningún IDE concreto ni el pago por una licencia de desarrollador hasta el momento de incluir la aplicación en la tienda de aplicaciones Google Play.

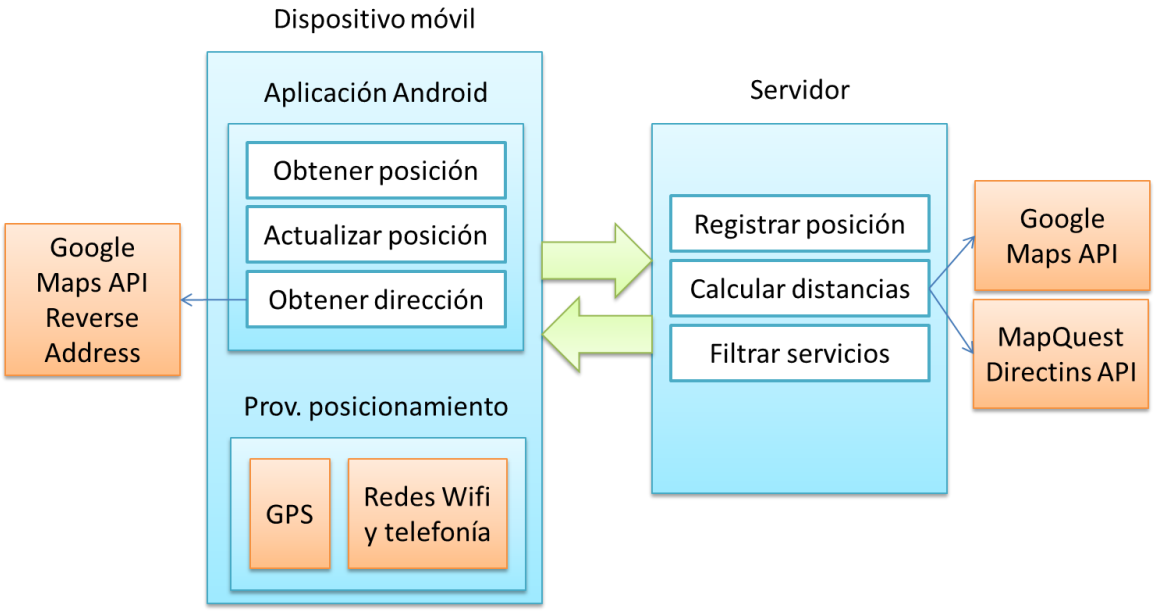
- El lenguaje de programación para el que ha sido desarrollado el SDK de Android es Java. Este lenguaje es ampliamente conocido, colocándose en la segunda posición del índice TIOBE de uso de lenguajes de programación por la comunidad.

Con el objetivo de mostrar y concretar el uso de las librerías desarrolladas, éstas han sido implementadas sobre una plataforma de oferta y consumo de servicios a través de una aplicación móvil en Android. Las librerías dotan a la plataforma de la capacidad para filtrar los servicios en base a la posición geográfica del usuario y la capacidad para pagar y cobrar por ellos. Esta plataforma comprende dos desarrollos paralelos: La aplicación móvil sobre Android y la aplicación servidor implementada en Java haciendo uso del framework Spring.

La librería de geolocalización del lado del servidor hace uso de diferentes APIs externas para calcular distancias entre dos puntos geolocalizados. La librería está diseñada para que se puedan implementar nuevas conexiones con otras APIs, no obstante, se ha implementado la conexión con la API de Google Maps y la conexión con la API de MapQuest. La primera API se ha elegido por ser la que ofrece información más precisa pero tiene un límite de conexiones sin coste y términos legales como que se requiere visualizar la información obtenida. Por estas restricciones, también se ha implementado la conexión con la API de MapQuest cuya información geográfica es libre además de ser creada y mantenida por la comunidad, aunque esta API no ofrece información tan precisa como la API de Google.

Del mismo modo, la librería de pagos del lado del dispositivo móvil hace uso del SDK de la pasarela de pagos de Paypal para Android y de la API REST del servicio de pagos con bitcoins Coinbase. Se ha decidido integrar la pasarela de pagos de Paypal porque es el servicio de pagos más conocido y, por lo tanto, más confiable. A pesar de ello, la comisión por transacción es de casi un 3% del importe de ésta. En contraposición de Paypal se ha elegido usar Coinbase ya que no hay ningún cargo por transferencia debido a la naturaleza distribuida de la moneda electrónica.

3.3. Geolocalización en dispositivos móviles



Sistema de geolocalización de la librería

3.3.1. Diseño de la librería para la aplicación móvil

La clase central de la librería de geolocalización en el dispositivo móvil es la interfaz Geolocalizador. Esta interfaz define los métodos de los que hará uso la aplicación para obtener el posicionamiento geográfico del usuario. La implementación de la interfaz para dispositivos Android es GeolocalizadorAndroid donde se hace uso de la API android.Location para gestionar los proveedores de posicionamiento a través de la clase LocationManager.

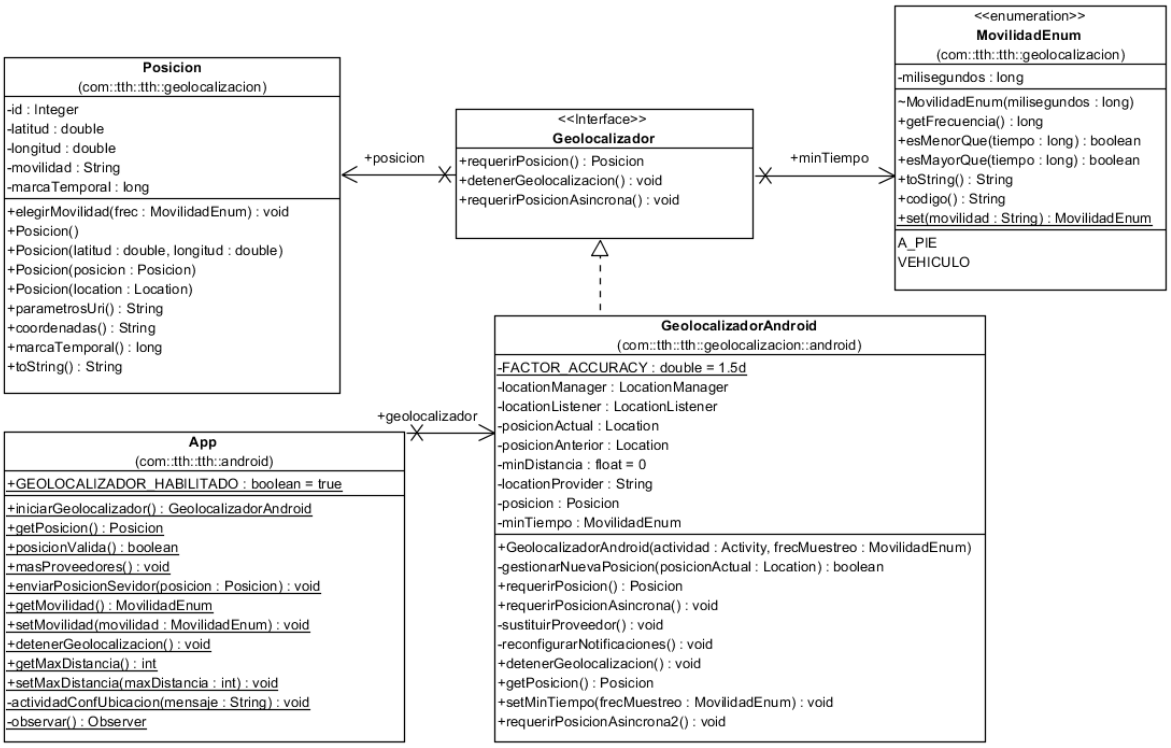


Diagrama de clases de la interfaz Geolocalizador

Para facilitar el uso de la librería de geolocalización en aplicaciones Android, se ha incluido la clase App que entre otros beneficios gestiona la clase GeolocalizadorAndroid tomando la responsabilidad de asegurar que sólo se crea una única instancia de esta clase que se puede usar entre las distintas actividades de la aplicación Android. La clase App ha sido diseñada con los siguientes métodos útiles para la librería de geolocalización:

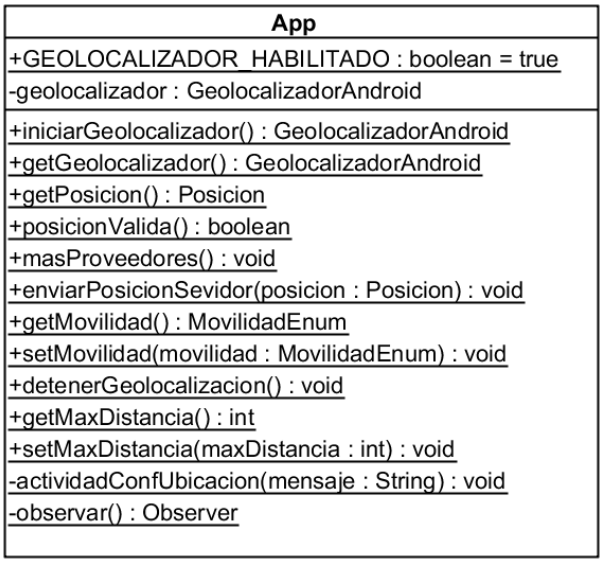


Diagrama UML de la clase App

El método iniciarGeolocalizador() instanciará un objeto de la clase GeolocalizadorAndroid comprobando antes qué proveedores de posicionamiento están

habilitados. Si no hay ninguno, lanzará la actividad de configuración de servicios de ubicación de Android para que el usuario habilite alguno de los proveedores disponibles. Tras iniciarGeolocalizador(), se podrá utilizar la instancia de tipo GeolocalizadorAndroid desde cualquier actividad de la aplicación a través de getGeolocalizador() y se obtendrá la posición geográfica del usuario con getPosicion(). Cuando se desee terminar con el servicio de geolocalización, se debe llamar a detenerGeolocalizacion(). Por ejemplo, al cerrar la aplicación con el objetivo de no malgastar la carga de la batería del dispositivo móvil.

El método posicionValida() permite verificar si el proveedor está funcionando correctamente. En caso de que no sea correcta, se puede solicitar al usuario que habilite más proveedores de posicionamiento a través del método masProveedores() que lanzará la actividad de configuración de los servicios de ubicación que proporcional el sistema operativo Android.

En la clase App se incluyen otros métodos como enviarPosicionServidor() que usará la clase GeolocalizadorAndroid para informar periódicamente al servidor de cual es lo posición del dispositivo. La clase GeolocalizadorAndroid extiende el tipo Observable. El mecanismo para que enviarPosicionServidor() sea llamado cuando se actualice la posición consiste en implementar un objeto Observer dentro del método observar() que se añade a la lista de observadores de GeolocalizadorAndroid. Este objeto ejecutará enviarPosicionServidor() cuando reciba la notificación desde GeolocalizadorAndroid.

También se encuentran en App los métodos getter y setter para la movilidad y la distancia máxima que están diseñados para aprovechar la interfaz SharedPreferences de Android que facilita el almacenamiento de la configuración elegida por el usuario. La distancia máxima es el parámetro que se pedirá al usuario para filtrar los elementos geoposicionados cercanos.

Nótese que todos los métodos comentados de la clase App son estáticos para que se pueda acceder desde cualquier actividad sin necesidad de instanciar la clase App ni mantener una referencia a esa instancia. Esta clase no es indispensable en la librería pero se ha diseñado como ejemplo de casos de uso de la librería y para facilitar la integración de ésta en Android. Sim embargo, se puede utilizar directamente la especialización GeoocalizadorAndroid de la interfaz Geolocalizador. Esta interfaz se ha diseñado con los siguientes métodos

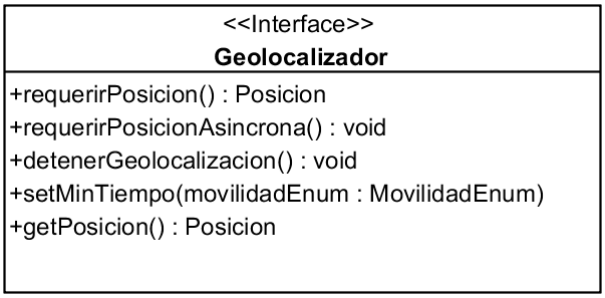


Diagrama de clase de la interfaz Geolocalizador

Teniendo en cuenta que la obtención de la posición a través de los diferentes proveedores de posicionamiento no es inmediato, es necesario contemplar en el diseño de la librería de geolocalización la necesidad de obtener una posición inmediata y la posibilidad de obtener la posición de manera asíncrona que es solicitar la posición y continuar con el flujo de ejecución de la aplicación; en algún momento posterior el proveedor de posicionamiento proporcionará a la librería la posición. Por ello, se ha diseñado requerirPosicion() que devuelve inmediatamente una posición geográfica que puede estar desactualizada y requerirPosicionAsincrona() que fuerza el uso de algún proveedor de posicionamiento para devolver a través de un evento la posición obtenida a partir de la llamada. Generalmente, se implementará este método haciendo que el sistema operativo obtenga actualizaciones periódicas de la posición y actualice ésta en la librería de forma que a partir de ejecutar requerirPosicionAsincrona() sólo será necesario obtener la posición con getPosicion() cuando se desee.

Las implementaciones de la interfaz Geolocalizador, deben mantener una estructura de datos que contenga la posición geográfica y la movilidad del usuario. Esta información se encapsula en objetos de tipo Posicion y se obtiene a través del método getPosicion() del geolocalizador. La clase Posicion es la siguiente:

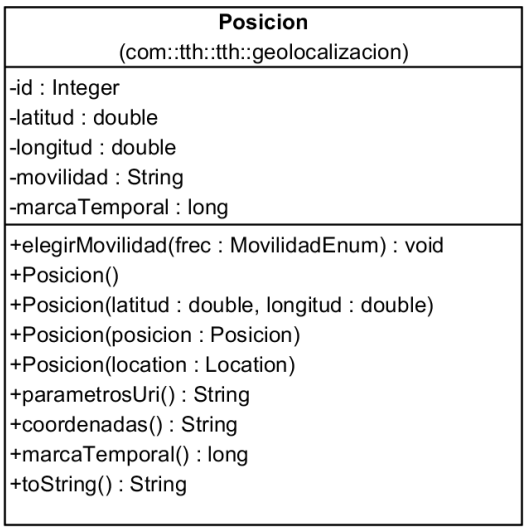


Diagrama de clase de Posicion

La clase Posicion contiene las propiedades id, latitud, longitud, marcaTemporal y movilidad. Se completa con los métodos constructores (vacío, con latitud y longitud, el constructor copia y el constructor a través de una instancia Location) y tres métodos para serializar un objeto Posicion toString(), parametrosUri(), -que es útil para el envío de la posición al servidor- y coordenadas(), -que devuelve un String de la forma latitud,longitud-.

La latitud y la longitud son los valores que obtiene el proveedor de posicionamiento en grados. Movilidad es la serialización del tipo enumerado MovilidadEnum; este tipo indica la actividad que realiza el usuario, que es caminar o moverse en vehículo. Esta información la proporcional el usuario a través de la interfaz gráfica y tiene utilidad doble: Dependiendo de la velocidad del usuario, permite ajustar la frecuencia de actualizaciones de la posición (cuanta menos velocidad, menos frecuencia de actualización y más se preserva la carga de la batería) y permite afinar el cálculo de la distancia entre dos posiciones dependiendo si se recorrerá andando o en vehículo. El diagrama de clases del enumerado MovilidadEnum es como sigue:

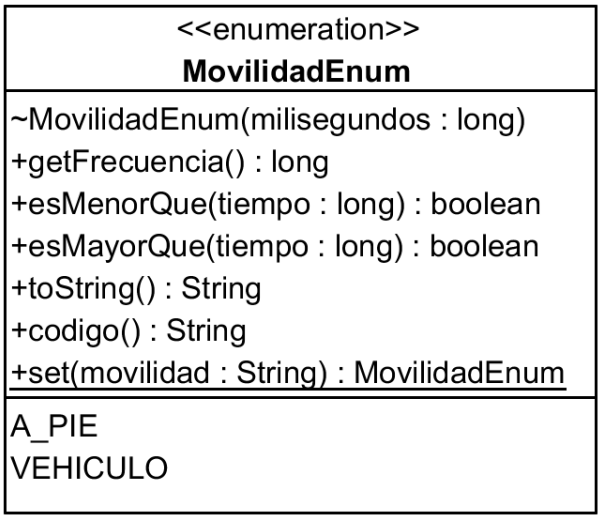


Diagrama de clase del enumerado MovilidadEnum

El enumerado contiene los dos estados A_PIE y VEHICULO con sus correspondientes frecuencias en milisegundos. Con la frecuencia del enumerado se configurará el mínimo tiempo que transcurrirá entre dos actualizaciones de la posición. En el caso de A_PIE la frecuencia es 600000 (diez minutos) y en el caso del estado VEHICULO la frecuencia es 120000 (dos minutos). Si el usuario se desplaza en vehículo y así lo ha indicado, se configurará el proveedor de posicionamiento para que proporcione la posición sólo pasados dos minutos. Si el usuario se mueve a pie, sólo proporcionará la posición pasados 10 minutos. De esta manera, se obtendrán actualizaciones de la posición con una diferencia razonable entre ellas y se optimizará el consumo de la carga de la batería.

De cara al desarrollador que quiera usar la librería de geolocalización, sea para extenderla y añadir una implementación de la interfaz Geolocalizacion o sea para usarla y obtener la posición geográfica, el diagrama se secuencia para obtener la posición arrojará luz sobre las posibles dudas:

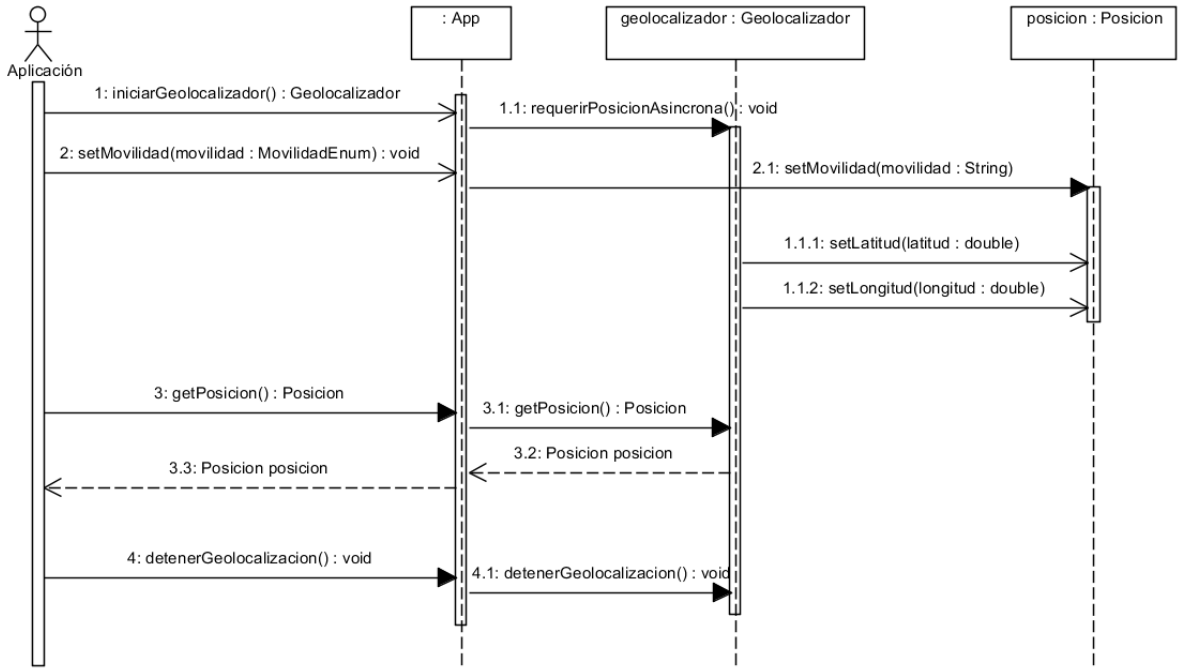


Diagrama de secuencia del uso del geolocalizador

Como se aprecia en el diagrama de secuencia, desde la aplicación bastan sólo cuatro llamadas a la librería para obtener la posición geográfica del usuario:

- 1 es la llamada para que App instancie e inicie la geolocalización.
- 2 configura la movilidad del usuario
- 3 es la llamada para obtener la posición en el momento en que es necesaria.
- 4 es la llamada para detener la geolocalización y optimizar el uso de la carga de la batería.

Las llamadas 1.1.2 y 1.1.3 que corresponden a la actualización de la posición recibida desde el proveedor de posicionamiento son asíncronas y periódicas. Estas llamadas se repetirán con la frecuencia mínima indicada con la llamada 2 para configurar la movilidad del usuario.

La implementación GeolocalizacionAndroid de la interfaz Geolocalizacion hace uso de la API Android Location para obtener la posición geográfica del dispositivo móvil usando o el dispositivo GPS o un algoritmo que calcula la posición a través de operaciones trigonométricas con las distancias entre el dispositivo móvil y las distintas antenas de

telefonía y puntos de acceso Wifi. La representación UML de la clase GeolocalizacionAndroid es la siguiente:

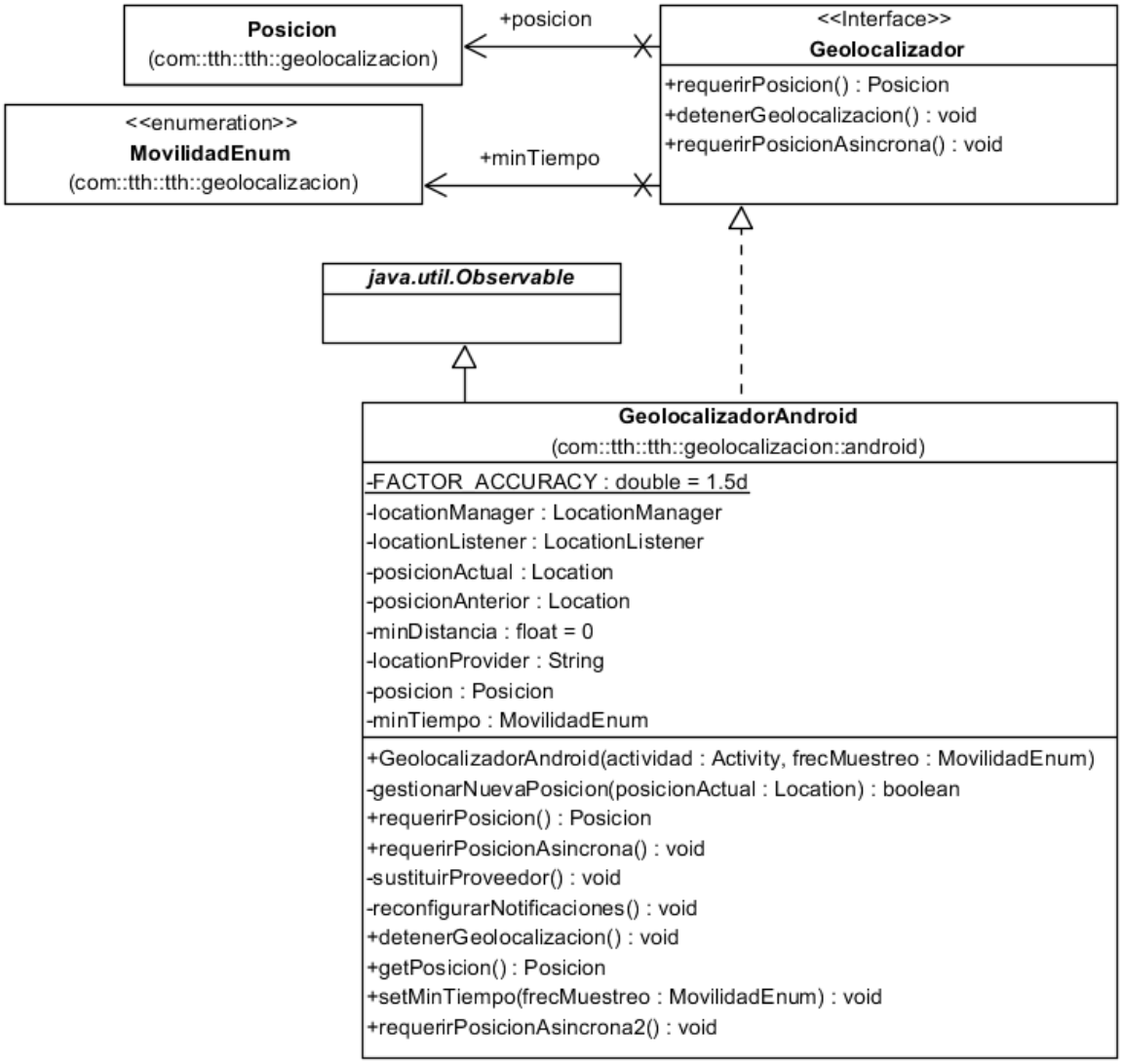


Diagrama de clase de GeolocalizadorAndroid

Las propiedades locationManager, locationListener y locatinProvider son necesarias para el paso de mensajes entre la librería y la API Android Location. La primera gestiona los proveedores de posicionamiento, la segunda es una implementación de la interfaz LocationListener para recibir actualizaciones de la posición y la tercera mantiene en memoria el identificador del proveedor de posicionamiento que se está usando en cada momento.

Al igual que minTiempo, minDistancia permite configurar la API Android Location para recibir actualizaciones de la posición tras recorrer una distancia mínima en metros. Sin embargo, en la librería no se requiere tal restricción así que esta propiedad tendrá el valor 0.

Para gestionar la posición geográfica del dispositivo se mantendrán en memoria dos objetos: posicioActual y posicionAnterior de tipo Location, que es la estructura de datos que proporciona Android y el objeto Posicion que será usado por la librería para devolver la posición geográfica en grados junto con la movilidad del usuario. Tener dos objetos Location permitirá comparar la precisión de las actualizaciones de la posición cuando el proveedor proporciona una nueva muestra. Se ha diseñado la constante FACTOR_ACCURACY con valor 1,5 para desechar una nueva muestra si es como mínimo un 50% menos precisa. Esta validación se puede realizar gracias a que la clase Location contiene un valor que indica la precisión en metros de la muestra tomada por el proveedor.

El método requerirPosicionAsincrona() registrará en el sistema operativo Android que locationListener solicita actualizaciones periódicas de la posición para un determinado proveedor de posicionamiento indicado con la propiedad locationProvider. La instancia locationListener de manera asíncrona recibirá un objeto Location y llamará a gestionarNuevaPosicion(), método en el que se comprobará la precisión de la nueva posición y si no es lo suficientemente precisa, decidirá cambiar el proveedor de posicionamiento a través de sustituirProveedor().

Para sustituir el proveedor, la librería elegirá entre los proveedores habilitados que proporciona la instancia LocationManager. Si elige uno nuevo, será necesario modificar los parámetros con los que se registra el objeto locationListener. Para ello, se llama al método reconfigurarNotificaciones(). Si, en cambio, la precisión es adecuada, se actualiza el objeto Posicion para que posteriormente cuando la aplicación haga uso de este objeto acceda a la posición actualizada del dispositivo.

GeolocalizadorAndroid extiende Observable utilizando así la implementación del patrón Observer de la API de Java. Los observadores se pueden suscribir a las notificaciones de actualización de la posición a través del método addObserver(). Serán notificados cuando se ejecute notifyObservers() pasando como argumento el objeto posicion.

3.3.2. Implementación de la librería para la aplicación móvil

A continuación, se detalla la implementación del geolocalizador para dispositivos con el sistema operativo Android.

```
public GeolocalizadorAndroid(Activity actividad,MovilidadEnum frecMuestreo)
```

Es el constructor de la clase. Instancia un objeto LocationManager y un objeto Posicion al que le pasa la movilidad del usuario que se recibe a través del argumento frecMuestreo.

Comprueba si está habilitado el proveedor GPS y lo elige como proveedor de posicionamiento actual. En caso contrario, establece como proveedor NETWORK_PROVIDER. Es requerido que antes de instanciar GeolocalizadorAndroid se pida al usuario habilitar alguno de estos dos proveedores de posicionamiento.

Por último ejecuta locationManager.getLastKnownLocation(locationProvider) para obtener la última posición que se solicitó desde alguna otra aplicación y que almacena el sistema operativo.

`private boolean gestionarNuevaPosicion(Location posicionActual)`

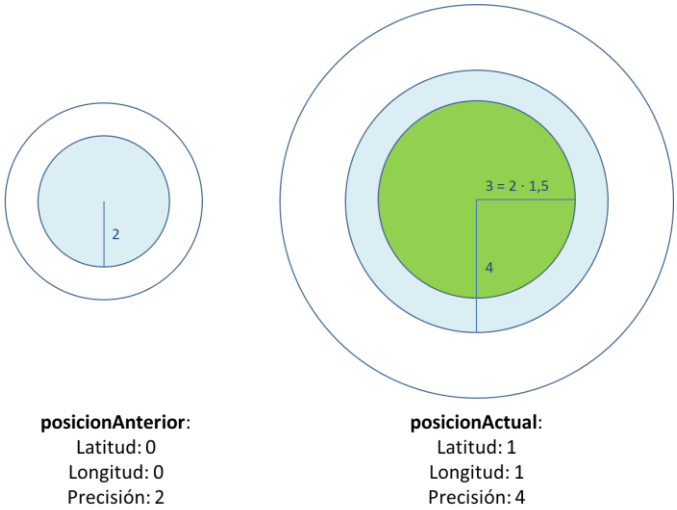
Este método se ejecuta tras recibir una actualización de la posición del dispositivo. Comprueba la precisión de la muestra y si acepta y se actualiza el objeto posición, devuelve true. El código es el siguiente:

```
private boolean gestionarNuevaPosicion(Location posicionActual) {
    if (posicionActual == null) return false;

    posicionAnterior = this.posicionActual;
    this.posicionActual = posicionActual;

    // getAccuracy() cuanto menor, mayor precisión
    if (posicionAnterior != null &&
        posicionActual.getAccuracy() > posicionAnterior.getAccuracy() * FACTOR_ACCURACY) {
        sustituirProveedor();
        return false;
    }

    posicion.setLongitud(posicionActual.getLongitude());
    posicion.setLatitud(posicionActual.getLatitude());
    posicion.setMarcaTemporal(posicionActual.getTime());

    setChanged();
    notifyObservers(getPosicion());
    return true;
}
```

Al recibir como argumento la nueva muestra, actualiza los dos objetos Location posicionAnterior y posicionActual que mantiene en memoria. Comprueba que posicionActual sea como máximo un 50% menos precisa que posicionAnterior. Si no es lo suficientemente precisa se sustituye el proveedor de posicionamiento y se devuelve false. Si es suficientemente precisa, se actualiza la instancia de Posicion con la longitud y la latitud de posicionActual y se notifica a los observadores la nueva posición geográfica devolviendo true al terminar la ejecución del método.

La propiedad accuracy se mide en metros y representa el radio del círculo dónde la probabilidad de que se encuentra la posición real del dispositivo es de 2/3. Cuanto más pequeño es el radio, más pequeño es el círculo y la precisión es mayor. La librería exige para

tomar una nueva muestra que el radio del círculo que denota la probabilidad de 2/3 no sea mayor que 1.5 veces el radio de la muestra anterior. Tómese como ejemplo:



Comparación de precisión entre dos actualizaciones de la posición

Como se aprecia, la propiedad accuracy del objeto Location posicionAnterior es 2 metros. Como accuracy de posicionActual es 4 metros, mayor que 3 (2 * FACTOR_ACCURACY = 1.5) se sustituirá el proveedor de posicionamiento actual para obtener una precisión adecuada. Este podría ser el caso en el que el proveedor fuese GPS_PROVIDER y el usuario entrase en un edificio haciendo que la precisión del GPS disminuya. Consciente de esta pérdida de precisión, la librería sustituiría el proveedor por NETWORK_LOCATION en busca de una precisión mayor.

```
private void sustituirProveedor()
```

Este método supone un conjunto de condiciones anidadas cuya representación en un diagrama de flujo es más intuitiva. Antes de sustituir un proveedor de posicionamiento por otro, comprueba que esté habilitado, ya que el usuario podría haber deshabilitado el proveedor en cuestión después de lanzar la aplicación. Si no están habilitados ni NETWORK_PROVIDER ni GPS_PROVIDER, se elegirá PASSIVE_PROVIDER que es un proveedor de las posiciones geográficas que obtienen otras aplicaciones dentro del sistema operativo.

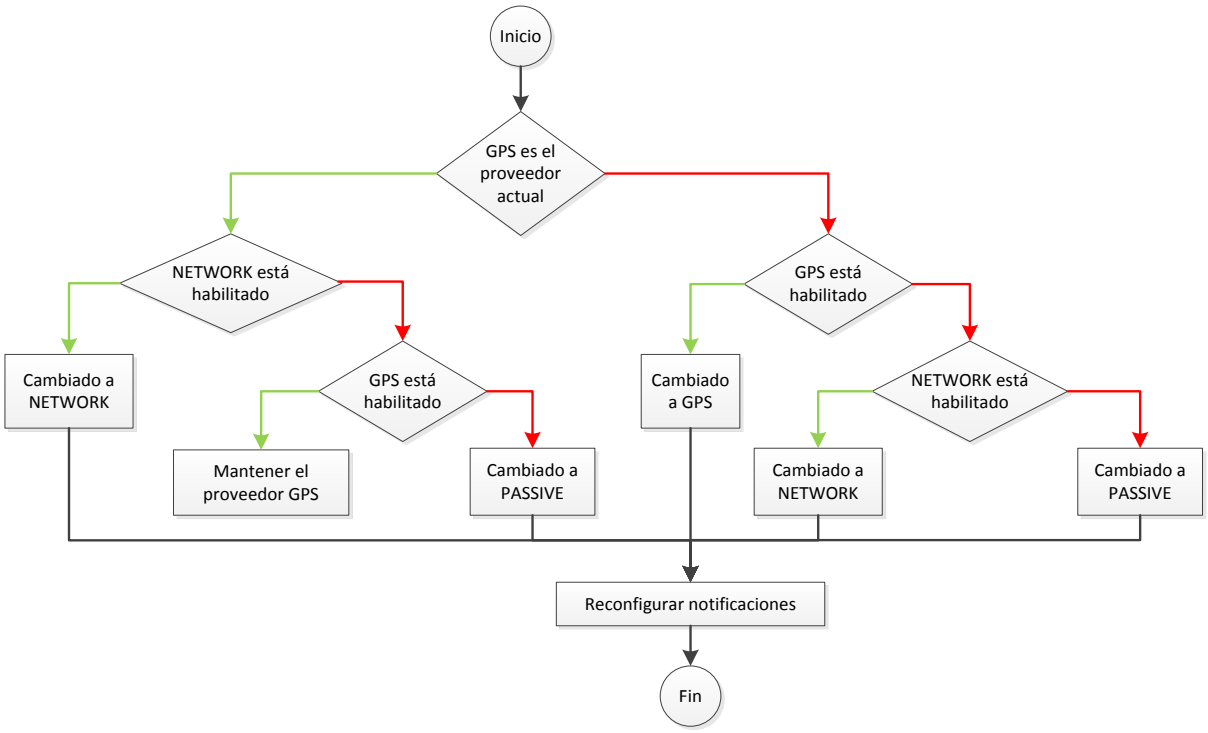


Diagrama de flujo de sustituirProveedor()

`private void reconfigurarNotificaciones()`

Este método simplemente indica al sistema operativo que deje de proporcionar actualizaciones de la posición a través del método detenerGeolocalizacion(). Posteriormente requiere las actualizaciones de la posición de forma asíncrona con los parámetros minTiempo, minDistancia, locationProvider y locationListener que son propiedades de la clase GeolocalizadorAndroid.

`public Posicion requerirPosicion()`

Al igual que en el constructor, se hace uso del método getLastKnownLocation(…) de la clase LocationManager para obtener la última posición que se solicitó desde alguna otra aplicación y que almacena el sistema operativo. La ventaja de este método es que no hay que esperar respuesta asíncrona, si no que se obtiene la instancia Posicion directamente. El inconveniente es que puede que la actualización de la posición esté obsoleta. Es responsabilidad del usuario de la librería comprobar la marca temporal de la instancia Posicion para asegurarse que es reciente.

`public void requerirPosicionAsincrona()`

Hace uso del método requestLocationUpdates de la instancia de LocationManager para obtener actualizaciones de la posición periódicas y de forma asíncrona. Incluye como argumento la implementación de un LocationListener. Este objeto LocationListener

implementa los métodos a los que el sistema operativo pasará mensajes cuando el proveedor de posicionamiento haya obtenido una posición nueva o haya cambiado su estado.

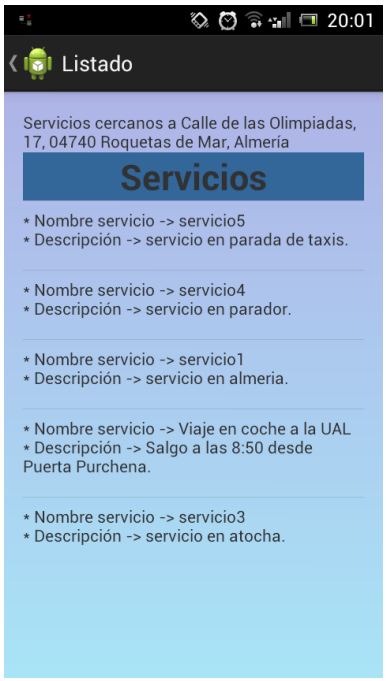
```
public void requerirPosicionAsincrona() {
    locationManager.requestLocationUpdates(locationProvider,
        minTiempo.getFrecuencia(),minDistancia,locationListener);
}

private LocationListener locationListener = new LocationListener() {
    public void onLocationChanged(Location location) {
        gestionarNuevaPosicion(location);
    }

    @Override
    public void onProviderDisabled(String proveedor) {
        sustituirProveedor();
    }

    @Override
    public void onProviderEnabled(String arg0) {}
    @Override
    public void onStatusChanged(String arg0, int arg1, Bundle arg2) {
};
```

En el código anterior se encuentra el método onLocationChanged() que recibe un objeto de tipo Location cuando Android notifica una actualización de la posición. Este método llama a gestionarNuevaPosicion().

Si el proveedor que se está utilizando cambia su estado y se deshabilita, se ejecutará el método onProviderDisabled() desde el que la librería llamará a sustituirProveedor() para escoger otro que pueda proporcionar actualizaciones del a posición.

Se ha implementado la clase GeolocalizacionInversa que extiende AsyncTask para realizar una tarea asíncrona y su propósito es conectarse a la API de Google y obtener la dirección mediante las coordenadas recogidas con GeoposicionamientoAndroid. Esta dirección se muestra en la actividad Listado al listar los servicios filtrados por proximidad para que el usuario pueda comprobar que la posición geográfica obtenida es veraz.

Se realiza una petición GET a https://maps.googleapis.com/maps/api/geocode/json? con el parámetro latlong indicando la posición en formato latitud,longitud y se obtiene un objeto JSON del que se extrae el campo formatted_address. El constructor de la clase GeolocalizacionInversa recibe una instancia TextView para establecer de manera asíncrona la dirección extraída de formatted_address.

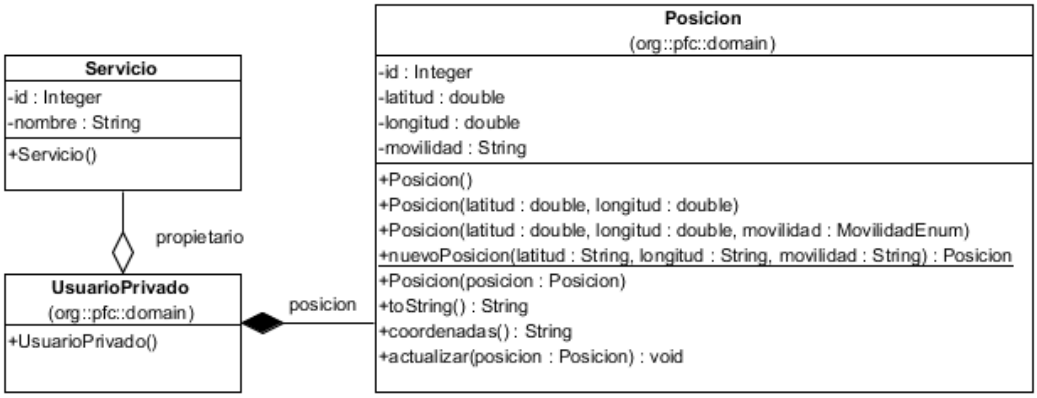


Captura de pantalla de la actividad para listar servicios

3.3.3. Diseño de la librería para la aplicación servidor

Para convertir un desarrollo de software en un servicio basado en geolocalización no sólo es necesario obtener la posición geográfica de cada dispositivo que ejecuta la aplicación, si no también personalizar el servicio en función de la posición geográfica de cada dispositivo.

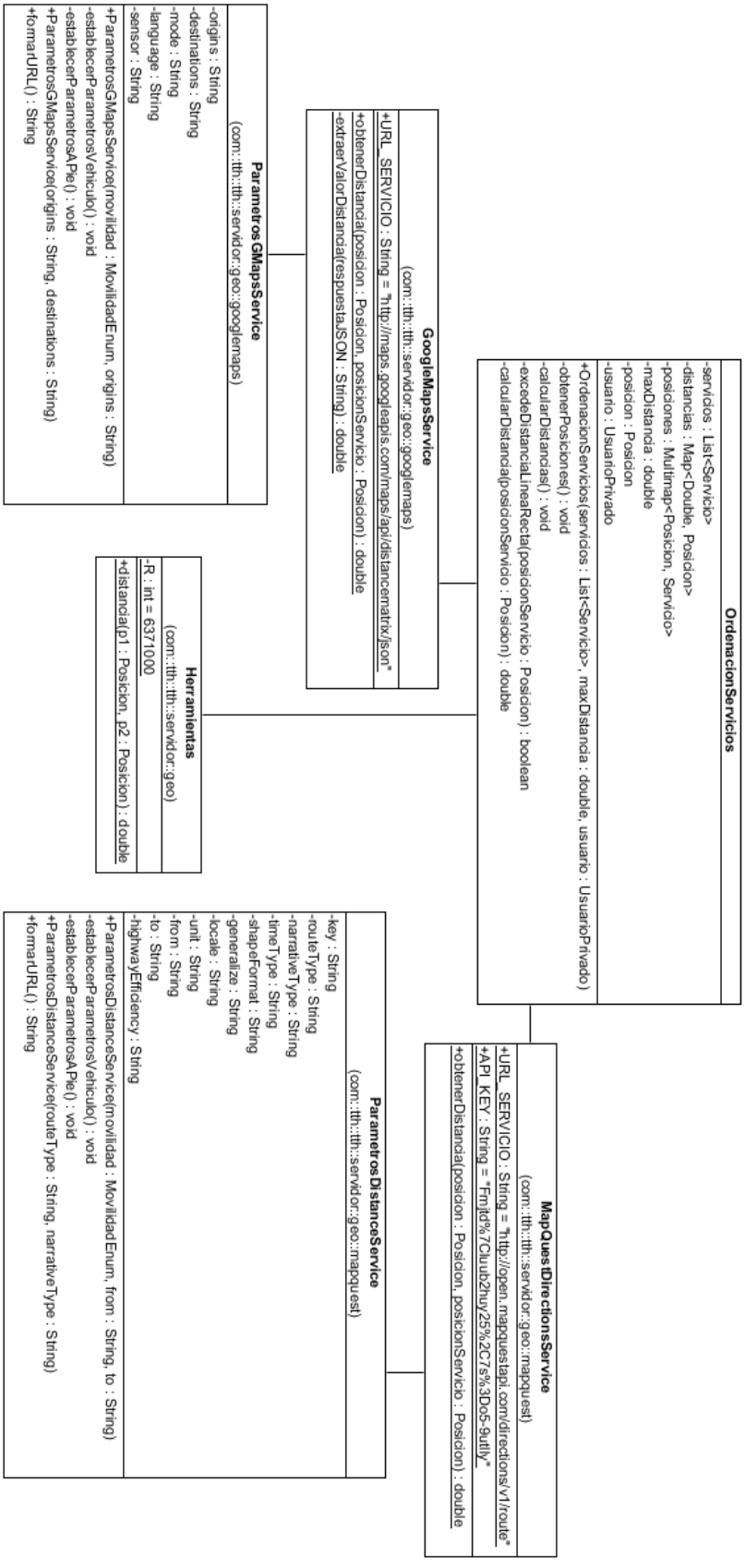
La aplicación servidor se abstrae de qué dispositivo y qué sensor obtiene la posición geográfica. Ya que la posición que se recibe está relacionada con el identificador de usuario, se asigna directamente la posición al usuario. La singularidad del paquete de servicios por proximidad es que se filtran servicios y no usuarios. Por ello, será necesario gestionar distintas estructuras de datos para relacionar posición geográfica con servicio. El modelo UML simplificado entre usuario y servicios es el siguiente:



Relación Servicio - Posicion

La clase central de la librería en la parte del servidor es OrdenacionServicios. El objetivo de esta clase es proporcionar una lista de los servicios disponibles para un usuario filtrando con una distancia máxima entre el usuario y estos servicios. El constructor de la clase recibirá la lista de servicios que filtrará, la distancia máxima entre el usuario y los servicios de dicha lista y una instancia de la clase UsuarioPrivado que proporcionará el identificador del usuario que solicita la lista filtrada y su posición geográfica.

A continuación se presenta el diagrama de clases de la clase OrdenacionServicios y sus relaciones GoogleMapsService y MapQuestDirectionsService.


OrdenacionServicios
-servicios : List<Servicio>
-distancias : Map<Double, Posicion>
-posiciones : Multimap<Posicion, Servicio>
-maxDistancia : double
-posicion : Posicion
-usuario : UsuarioPrivado
+OrdenacionServicios(servicios : List<Servicio>, maxDistancia : double, usuario : UsuarioPrivado)
-obtenerPosiciones() : void
-calcularDistancias() : void
-excedeDistancialLineaRecta(posicionServicio : Posicion) : boolean
-calcularDistancia(posicionServicio : Posicion) : double
GoogleMapsService
(com::tth::tth::servidor::geo::googlemaps)
+URL_SERVICIO : String = "http://maps.googleapis.com/maps/api/distancematrix/json"
+obtenerDistancia(posicion : Posicion, posicionServicio : Posicion) : double
-extraerValorDistancia(respuestaJSON : String) : double
ParametrosGMapsService
(com::tth::tth::servidor::geo::googlemaps)
-origins : String
-destinations : String
-mode : String
-language : String
-sensor : String
+ParametrosGMapsService(movilidad : MovilidadEnum, origins : String)
-establecerParametrosVehiculo() : void
-establecerParametrosAPie() : void
+ParametrosGMapsService(origins : String, destinations : String)
+formarURL() : String
MapQuestDirectionsService
(com::tth::tth::servidor::geo::mapquest)
+URL_SERVICIO : String = "http://open.mapquestapi.com/directions/v1/route"
+API_KEY : String = "Fmjtd%7Cluub2huy25%2C7s%3Do5-9utfy"
+obtenerDistancia(posicion : Posicion, posicionServicio : Posicion) : double
Herramientas
(com::tth::tth::servidor::geo)
-R : int = 6371000
+distancia(p1 : Posicion, p2 : Posicion) : double
ParametrosDistanceService
(com::tth::tth::servidor::geo::mapquest)
-key : String
-routeType : String
-narrativeType : String
-timeType : String
-shapeFormat : String
-generalize : String
-locale : String
-unit : String
-from : String
-to : String
-highwayEfficiency : String
+ParametrosDistanceService(movilidad : MovilidadEnum, from : String, to : String)
-establecerParametrosVehiculo() : void
-establecerParametrosAPie() : void
+ParametrosDistanceService(routeType : String, narrativeType : String)
+formarURL() : String

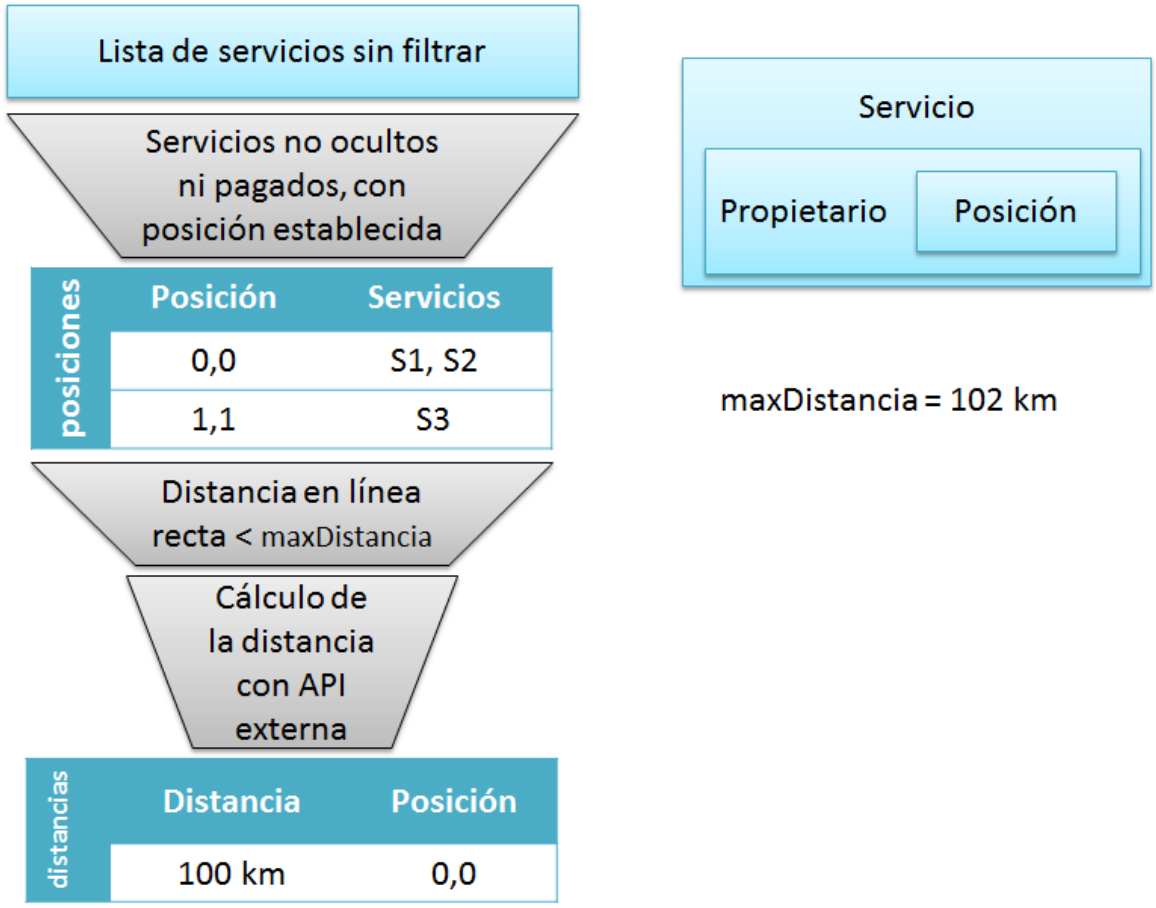
El primer paso para filtrar los servicios y que se realiza desde el mismo constructor de la clase es obtener la posición geográfica de cada servicio. Para ello, la clase OrdenacionServicios incluye una estructura de datos que mapea instancias de tipo Posicion con listas de servicios, ya que un usuario puede ofrecer varios servicios y por tanto estos servicios se encuentran en la misma posición. El procedimiento se basa en recorrer la lista de servicios proporcionada a través del constructor, hacer un primer filtrado de servicios y añadir en el mapa posiciones el servicio según su posición. El primer filtrado excluirá servicios si:

- El servicio está oculto o ha sido pagado.
- El usuario es propietario del servicio.
- La posición del servicio no está establecida o no es válida.

El segundo paso consiste en calcular la distancia entre la posición geográfica del usuario que ha realizado la petición del filtrado de los servicios y cada posición del mapa posiciones. En este caso se va a realizar otro segundo filtrado ya que el cálculo de la distancia se realiza a través de APIs externas al sistema que tardan unos segundos en responder y que tienen límite mensual de conexiones.

El segundo filtrado consiste en calcular la distancia en línea recta entre dos posiciones geográficas. Es sabido que la distancia en línea recta es una cota inferior de la distancia en vehículo o caminando. Así, si la distancia en línea recta es mayor que maxDistancia, los servicios con dicha posición no se incluyen en la lista de servicios filtrados.

Posteriormente, se calcula haciendo uso de las APIs externas la distancia y los servicios que están más cerca que maxDistancia se incluyen en otra estructura de datos que mapea la distancia calculada con la posición geográfica.



Sistema de filtrado de los servicios

En el diagrama se observa cómo la lista de servicios sin filtrar pasa a través de tres filtros (dibujados como trapecios invertidos de color gris) generando dos estructuras de datos referenciadas como posiciones y distancias. La lista de servicios filtrados se obtiene enlazando las posiciones del mapa distancias con el mapa posiciones. A la derecha del diagrama se muestra que la posición de un servicio se obtiene accediendo a la propiedad propietario del servicio.

La fórmula que se aplicará para calcular la distancia en línea recta se conoce como fórmula de Haversine[23]. Para calcularla se utilizará la librería Math de la API de Java a través del método estático distancia() en la clase Herramientas. La fórmula es la siguiente:

$$\Delta lat = lat_2 - lat_1$$

$$\Delta long = long_2 - long_1$$

$$a = \sin^2\frac{\Delta lat}{2} + \cos lat_1 \cdot \cos lat_2 \cdot \sin^2\frac{\Delta long}{2}$$

[23] Cálculo de las distancias sobre la tierra http://personal.franchu.net/2007/11/16/gis-calculo-de-distancias-sobre-la-tierra/

$$c = 2\text{atan}^2\left(\sqrt{a}, \sqrt{1-a}\right)$$

$$distancia = 6{,}371 \cdot 10^6 \cdot c$$

El método para calcular la distancia entre dos puntos geográficos hace uso en primer lugar de la API Matriz Distancia de Google Maps usando el método estático obtenerDistancia() de la clase GoogleMapsService. En caso de que el valor devuelto por la API no sea válido, la librería realiza la petición de la distancia a la API de MapQuest a través del método estático calcularDistancia() en la clase MapQuestDirectionsService.

3.3.4. Implementación de la librería para la aplicación servidor

La estructura de datos posiciones se ha implementado usando un HashMultiMap de la librería Guava. Esta librería ha sido desarrollada por Google para facilitar el uso de la API de Java en sus propios proyectos y extender su funcionalidad en diversos aspectos como estructuras de datos, gestión de E/S y manejo de Strings. La clase HashMultiMap permite que la clave mapee a una colección de objetos. En este caso, una instancia de Posicion puede mapear diferentes servicios. HashMultiMap<Posicion,Servicio> es equivalente a usar HashMap<Posicion,Collection<Servicio>> pero los métodos de HashMultiMap la convierte en una clase realmente útil y fácil de usar.

El mapa distancias se ha implementado como un árbol rojo-negro en el que las claves son instancias de tipo Double que almacenan la distancia en metros y mapea a instancias de tipo Posicion. Al ser la estructura de datos un árbol rojo-negro que obliga que el tipo de la clave implemente la interfaz Comparable, es seguro que al recorrer el mapa los valores estarán ordenados de forma creciente por su clave.

Calcular la distancia con Google Maps API

El método calcularDistancia() de la clase GoogleMapsService devuelve la distancia en metros calculada por el servicio de Google a partir de dos instancias de tipo Posicion. Este método se complementa con la clase ParametrosGMapsService parar elaborar de forma sencilla la cadena de parámetros de la petición GET que envía usando la librería REST Client de Spring.

El servicio de Google Maps responde con una cantidad bastante considerable de información, lo que hace complicado moverse a través de mapas y listas si se convierte el objeto JSON a un mapa. Por ello, para la respuesta de Google Maps se decidió extraer la distancia en metros buscando una expresión regular con las clases Matcher y Pattern del paquete regex de la API de Java.

Calcular la distancia con MapQuest API

Al igual que con la API de Google Maps, la distancia se obtiene enviando una petición HTTP GET al servicio Directions de MapQuest usando la librería Spring REST Client. En este caso también el método calcularDistancia utiliza una clase ParametrosDistanceService para formar la cadena de parámetros. No obstante, la respuesta que devuelve el servicio de MapQuest es reducida y permite que se extraiga la distancia convirtiendo el objeto JSON en un HashMap y navegando a través de sus claves route y posteriormente distance como se ve en el siguiente fragmento de código:

```
ParametrosDistanceService parametros = new ParametrosDistanceService(
    MovilidadEnum.set(posicion.getMovilidad()),
    posicion.coordenadas(),
    posicionServicio.coordenadas());

String url = parametros.formarURL();

HashMap result = new HashMap<String, String>();

try {
    result = restTemplate.getForObject(new URI(url),HashMap.class);
} catch (Exception e) {
    e.printStackTrace();
    return -1;
}

double distancia = ((Double)(((HashMap)result.get("route"))
    .get("distance"))).doubleValue() * 1000;
```

3.3.5. Referencia 1. La API Android Location

El sistema operativo Android proporciona para los desarrolladores sobre su plataforma un conjunto de clases para poder utilizar los sensores de posicionamiento del dispositivo móvil. Esta API está disponible desde el nivel 8 con el que se lanzó la versión 2.2 de Android. La documentación se puede encontrar en http://developer.android.com/google/play-services/location.html. Las clases que comprenden esta API se encuentran organizadas en el paquete android.location

Google, por su parte proporciona en la librería Google Play Services la API Location Services que proporciona clases para el uso de la API Android Location a más alto nivel.

La clase principal de la API Android Location es LocationManager. Para obtener una instancia de esta clase, es necesario llamar al método del contexto getSystemService(Context.LOCATION_SERVICE). Con la instancia de LocationManager, el desarrollador puede:

- Consultar la posición del dispositivo a través de alguno de la lista de proveedores de posicionamiento.
- Solicitar la recepción periódica de actualizaciones de la posición del usuario a través de algún proveedor de posicionamiento que puede ser escogido según criterios del desarrollador.
- Solicitar que el sistema operativo lance un objeto Intent especificado si el dispositivo se encuentra en una determinada posición.

Proveedores de posicionamiento

La interfaz de geolocalización de Android proporciona tres sensores diferentes para obtener la posición del dispositivo respecto a las coordenadas terrestres. El sensor se puede elegir a través de las constantes que implementa el tipo LocationManager. Es importante elegir el proveedor de posicionamiento adecuado en cada momento ya que cada uno tiene características diferentes:

GPS_PROVIDER:

Esta constante solicita al GPS la notificación de la posición del dispositivo. Es la opción que más batería consume y no funciona correctamente en interiores pero proporciona una posición de mayor precisión.

NETWORK_PROVIDER:

Este proveedor obtiene la posición del dispositivo calculándola a través de las antenas de telefonía y obteniendo información a través de la red Wifi a la que está conectada el dispositivo. NETWORK_PROVIDER no consume tanta carga de batería como GPS_PROVIDER y obtiene la posición en menos tiempo además de servir tanto en interior como en exterior. La desventaja de este proveedor de posicionamiento es que la posición que calcula no es tan precisa como la que se obtendría con GPS_PROVIDER.

PASSIVE_PROVIDER:

Técnicamente PASSIVE_PROVIDER no es un sensor pero el uso de esta constante permite a la aplicación que obtenga la posición que solicita otras aplicaciones del dispositivo. Esta opción permite el ahorro de batería pero es posible que no se obtenga las actualizaciones del posicionamiento en el momento deseado.

Para elegir el proveedor de posicionamiento más adecuado también es interesante conocer la velocidad de su movimiento. Si el usuario está caminando, la posición obtenida mediante GPS es más adecuada ya que la precisión es mayor y el intervalo entre actualizaciones de la posición puede ser mayor. En cambio, si el usuario se mueve en un vehículo, es necesario que el intervalo de actualización de la posición sea más reducido por lo que obtener la posición a través del proveedor de red es la opción acertada.

Es necesario especificar en el archivo de configuración AndroidManifest.xml qué permisos se necesitan para usar el GPS o para que Android obtenga la posición del dispositivo a través de antenas de telefonía y la red Wifi.

Entre las etiquetas <manifest> para poder usar GPS_PROVIDER o NETWORK_PROVIDER se indicará con el permiso ACCESS_FINE_LOCATION:

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

En cambio, para usar sólo NETWORK_LOCATION, bastará con indicar el permiso COARSE_LOCATION:

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

Con LocationManager se puede hacer uso del método getBestProvider(Criteria criteria, boolean enabledOnly) que devuelve la constante que identifica al proveedor que mejor se adecúe a los criterios indicados en un objeto Criteria. Los criterios que se pueden especificar para ayudar al sistema operativo a decidir cuál es el mejor sensor para obtener la

posición son: Requerimientos de batería, precisión, orientación, velocidad, altitud y coste monetario.

Obtener la posición del usuario

Los proveedores de posicionamiento cuando entregan la posición del usuario proporcionan tanto una marca de tiempo como un valor que indica la precisión de la muestra. Si en la aplicación se obtienen diferentes muestras a través de diferentes proveedores de posicionamiento, es tarea del desarrollador comparar las muestras obtenidas según la precisión y según la marca de tiempo porque una muestra más reciente puede ser menos precisa y ser desechada contra una muestra anterior.

Para obtener una posición a través de LocationManager es necesario implementar la interfaz LocationListener:

```
LocationListener locationListener = new LocationListener() {
    public void onLocationChanged(Location location) {}
    public void onStatusChanged(String provider, int status, Bundle extras)
{}
    public void onProviderEnabled(String provider) {}
    public void onProviderDisabled(String provider) {}
  };
```

El procedimiento consiste en registrar la implementación de la interfaz en `LocationManager` para que de manera asíncrona, cuando `LocationManager` quiera notificar una nueva posición llamará al método `onLocationChanged()` pasando como argumento el objeto de tipo `Location`. Este enfoque se conoce generalmente como implementación de un `callback` que permite manejar eventos asíncronos.

Tras definir la implementación de `LocationListener`, hay que registrar la solicitud de actualización de la posición del usuario. Esto se hace a través de `LocationManager`. Algunos métodos útiles para obtener la posición son:

```
Location getLastKnownLocation(String provider)
```

Este método obtiene la última posición que se obtuvo con el proveedor indicado como argumento en el mensaje. Este objeto `Location` puede estar almacenado en memoria así que quizás el sistema operativo no ponga en marcha el sensor de posición. Debido a esto, se hace obligado para el usuario de la API comprobar la fecha de la muestra obtenida comprobado que no esté desactualizada.

```
void requestLocationUpdates(String provider, long minTime, float
minDistance, LocationListener listener)
```

Con este método se registra una interfaz `LocationListener` que recibirá las actualizaciones que proporcione el proveedor de posicionamiento. El primer parámetro es la constante que determina qué proveedor tomará la muestra.

El segundo parámetro es el intervalo mínimo de milisegundos en el que se quiere obtener muestras. Si `minTime` es 4000, en condiciones favorables se podría obtener notificaciones de la posición cada 4 segundos. El tercer argumento `minDistance` es la distancia mínima en metros con la que se desea recibir notificaciones de la posición. Por ejemplo, si se indica que `minDistance` es 5, se recibirán actualizaciones de la posición cada 5 metros si el sensor está completamente disponible para la aplicación. El algoritmo de notificación de una nueva muestra de la posición primero comprueba el parámetro `minDistance`, si es 0 lo ignora y pasa a comparar el parámetro `minTime`. Este algoritmo puede ser resumido de la siguiente manera:

```
if (minDistance > 0 && distanciaRecorrida < minDistance) return;
if (tiempoTranscurrido >= minTime) {
     notificarAplicacion();
}
```

Es recomendable establecer el valor 0 al parámetro `minDistance` si no se quiere obtener posiciones al recorrer una distancia mínima y obtener posiciones sólo según el tiempo que haya transcurrido. Hay que tener en cuenta que aunque se establezca el valor `minDistance` a 1000 y no se reciban notificaciones pasado un kilómetro, Android, continuamente comprobará la posición para determinar si se ha modificado la distancia lo suficiente como para notificar a la aplicación con el consecuente consumo de batería.

El último argumento es el objeto `LocationListener` comentada anteriormente. Como mínimo se ha de implementar el método `onLocationChanged` que será al que pase el mensaje el sensor con el objeto de tipo `Location`.

Este método es un punto de entrada para el método más complejo `requestLocationUpdates(long minTime, float minDistance, Criteria criteria, PendingIntent intent)` que es llamado internamente desde el código que implementa `requestLocationUpdates(String provider, long minTime, float minDistance, LocationListener listener)`.

Alternativamente, se puede utilizar el siguiente método sin necesidad de implementar la interfaz `LocationListener`:

```
void addProximityAlert(double latitude, double longitude, float radius,
long expiration, PendingIntent intent)
```

Establece una alerta para una posición determinada y un radio alrededor de la posición. En este caso el sistema operativo se encarga de procesar las notificaciones de los sensores que pueden ser tanto NETWORK_PROVIDER como GPS_PROVIDER. Se lanza el objeto `Intent` pasado como argumento cuando el dispositivo es detectado dentro del área que representa la posición y el radio. El argumento `expiration` marca el tiempo de vida de la alerta. Si se desea que no expire la alerta, se puede pasar -1 como argumento.

Elección de los parámetros de actualización de la posición

Por lo general, los valores a tener en cuenta para requerir al sistema operativo la actualización de la posición son:

- La precisión de la muestra.
- Si el usuario se mueve caminando en vehículo.

Con estos valores se variarán parámetros como son el proveedor de posicionamiento y el intervalo mínimo para recibir una actualización de la posición. De esta manera se pretende optimizar la precisión de la posición obtenida y el consumo de la batería del dispositivo.

La aplicación almacena tanto la última muestra de la posición como la anterior. Al obtener una actualización de la posición, se compara la precisión de ambas posiciones y si resulta que la muestra actual tiene una precisión más baja, se sustituye el proveedor de posicionamiento por otro disponible. Esto puede ocurrir, por ejemplo, cuando el proveedor de posicionamiento es el GPS y se solicita la muestra desde el interior de un edificio donde el proveedor no obtiene una muestra con una precisión adecuada. En este caso, la aplicación sustituye el GPS por el proveedor de red si está disponible, si no, sustituirá por PASSIVE_PROVIDER.

Si el usuario se mueve en vehículo, es necesario que el intervalo entre actualizaciones de la posición sea menor y se obtenga actualizaciones con más frecuencia. Se ha estimado para la aplicación que en vehículo se obtenga la posición del usuario como mínimo cada dos minutos. Sin embargo, si el usuario camina, el intervalo de actualización puede ser mayor lo que repercute positivamente en un menor uso de la batería. En este caso, el parámetro minTime de la llamada al método requestLocationUpdates se modifica para que se obtengan actualizaciones como mínimo cada 10 minutos.

El tipo Location

Es el tipo de objeto que pasa por argumento el sistema operativo al método onLocationChanged(). Contiene como mínimo la latitud y la longitud. En circunstancias especiales, puede obtener otros valores como la orientación, la velocidad y la altitud. De este objeto también se puede obtener la precisión de la muestra y el proveedor que ha obtenido la muestra.

Android proporciona la precisión a través del método getAccuracy(). El valor que devuelve esté método se mide en metros y representa el radio del círculo que desde la posición del usuario supone el 68% de acierto de la posición real. Por ejemplo, si el usuario está en 0,0 y getAccuracy() devuelve 1 m, quiere decir que hay un 68% de probabilidad de que la posición real del usuario se encuentre en el círculo trazado de radio 1 metro desde la posición del usuario.

Por tanto, si se comparan dos muestras, será más precisa la de menor getAccuracy(). Por ejemplo, si de dos muestras, la primera devuelve 1 metro y la segunda devuelve 5 m, se sabe que en un radio de 1 metro hay un 68% de probabilidad, en ese mismo radio para la segunda muestra hay menos probabilidad de que se encuentre la posición real.

Cancelar la actualización de la posición

Para cancelar la actualización de la posición se puede llamar al método removeUpdates(LocationListener listener) que indica al sistema operativo que no se desean recibir más actualizaciones de la posición. Imprescindible para dejar de consumir batería cuando ya no se necesitan más el servicio de geolocalización.

Novedades sobre geoposicionamiento en Android

En las últimas versiones se ha continuado el desarrollo de las funcionalidades de geoposicionamiento de dispositivos Android desde Google Play Services. Esto quiere decir que Google ha actualizado la API Location Services en la última versión del sistema operativo pero no así la API Android Location. Para poder usar esta actualización es necesario que el desarrollador añada al proyecto la librería Google Play Services Client y que el usuario para poder instalar la aplicación tenga Google Play Services instalado en el dispositivo móvil.

En la web de Play Location Services http://developer.android.com/google/play-services/location.html se destacan tres novedades que se han incluido en la última versión de la librería, lanzada con la versión 4.4 (Kit Kat) de Android. Las novedades son:

- **Fusión de proveedores de posicionamiento:** Esta característica permite al desarrollador despreocuparse de qué proveedor de posicionamiento usar ya que es la librería la que gestiona los proveedores de posicionamiento según el nivel de carga del dispositivo móvil o la precisión que el usuario requiera. Además, se destaca la versatilidad de la API ya que se puede configurar para tener en cuenta otros factores tales como el uso que la aplicación hará del posicionamiento obtenido: Algunas aplicaciones pueden hacer uso de la posición del usuario en primer plano, mostrándola por pantalla o insertándola en un mapa. En cambio, otras aplicaciones pueden hacer uso de la posición para enviarla a un servidor remoto, como es el caso de la aplicación de este documento.

- **Áreas limitadas geográficamente:** La librería Google Play Location Services permite al desarrollador indicar áreas geográficas para que sea notificado si el dispositivo entra o sale de dicha o dichas áreas geográficas. Es posible añadir y eliminar una o varias y configurar si se desea ser notificado cuando el dispositivo entra, sale o entra y sale en un área con límites geográficos.

  Esta característica también se ha optimizado para hacer uso mínimo de la carga de la batería del dispositivo. Se tendrá en cuenta la lejanía al área con límites geográficos para reducir el uso del proveedor de posicionamiento.

- **Reconocimiento de la actividad del usuario:** Con el desarrollo de aplicaciones móviles cada vez más conscientes del entorno y de las circunstancias del usuario, la API Location Services pretende facilitar al desarrollador la información necesaria para determinar la actividad que está realizando el usuario. En base a esta información, la aplicación proporcionará al usuario el contenido correcto acorde a su actividad. Ha sido añadida a la API la capacidad de informar a la aplicación cuál es la actividad que el usuario está realizando, sea caminando, montando en bicicleta o moviéndose en un vehículo.

Tanto la característica de fusión de proveedores de posicionamiento como el reconocimiento de la actividad del usuario han sido desarrolladas a medida para la aplicación. Lamentablemente, estas características han aparecido para la API nivel 19 e integrando en el desarrollo la librería Google Play Services. Los desarrollos de la API de Android no tienen carácter retroactivo con lo cual, estas novedades no se pueden usar desde la API 8 que es para la plataforma sobre la que se desarrolla la aplicación.

3.3.6. Referencia 2. La API de Google Maps

El equipo de Google Maps ofrece a los desarrolladores de aplicaciones basadas en datos geográficos de la plataforma Google Maps cinco servicios webs diferentes para

obtener datos geográficos. Estos servicios webs se pueden encontrar en la web https://developers.google.com/maps y son los siguientes:

- API de rutas
- API de elevación
- API de codificación geográfica
- API de Google Places
- API de matriz de distancia

Además de estos servicios web a los que se se pueden hacer uso desde cualquier lenguaje de programación, el equipo de Google Maps pone a disposición de los desarrolladores una potente API para Javascript en su versión 3 y diferentes SDKs para iOS y para Android. Estos SDKs recogen más información y con mejor formato que los servicios webs, pero aunque lamentablemente, no hay ningún SDK disponible para Java, los servicios webs que se detallan a continuación cubren de sobra la necesidad de la aplicación, que es el cálculo de la distancia entre dos puntos.

API de rutas

La API de rutas es un servicio web que calcula una ruta entre dos puntos. Se puede requerir que entre estos dos puntos se pase por una serie de hitos (puntos intermedios) y que se apliquen restricciones como evitar peajes o autopistas. El servicio web de rutas no está diseñado para calcular rutas en tiempo real y ofrecerlas gráficamente al usuario. De hecho, se recomienda al desarrollador que se conozca de antemano el destino de la ruta a calcular y que se mantenga una memoria caché propia para almacenar información de rutas ya calculadas.

La dirección URL del servicio es
https://maps.googleapis.com/maps/api/directions/output?parameters

En la URL anterior, output se sustituye por json o xml, esta es la nomenclatura usual de los servicios RESTful para indicar el formato de la respuesta. En cambio, parameters es una lista de parámetros que incluye alguno de los siguientes:

- **origin y destination:** Pueden indicarse con las coordenadas latitud,longitud o con la dirección que será codificada por el servicio a coordenadas latitud,longitud.

- **sensor:** Parámetro obligatorio para indicar si la solicitud de indicaciones procede de un dispositivo con sensor de ubicación. En la documentación no se especifica para que es conveniente indicar este hecho, pero hacen bastante hincapié en que es beneficioso.

- **mode**: Los posibles valores para este parámetro son "driving", "walking", "bicycling" y "transit". El parámetro "transit" solicita indicaciones a través de rutas de transporte público, opción que realmente distingue a Google Maps de los demás servicios de cálculo de rutas porque manejan una cantidad de información de transporte público muy completa. Se puede sacar el máximo provecho del parámetro mode configurado en "transit" con los parámetros "arrival_time" y "departure_time" que indican en segundos la hora de salida del origen y llegada al destino deseada.

- **waypoints**: Parámetro para indicar puntos intermedios por los que la ruta tiene que pasar. Estos puntos se indican de la misma forma que "origin" y "destination".

- **avoid**: Este parámetro, que se comentó brevemente, permite escribir expresiones para intentar evitar ciertas rutas. Por el momento, se puede configurar con los valores "tolls" y "highways" que indican que se prefieren rutas que no pasen por peajes o por autovías respectivamente.

Aunque no es la API recomendada para calcular distancias, este servicio devuelve en la respuesta la distancia y la duración del trayecto estimada en segundos.

API de elevación

El API de elevación proporciona información sobre la elevación de un punto de la superfície terrestre. También proporciona la profundidad en el océano de ese punto, indicando este dato con valor negativo. En caso de que no se tenga información sobre la elevación de un punto en concreto, se devuelve un cálculo medio de las cuatro ubicaciones más cercanas. En la documentación, sugieren que esta API se puede usar para aplicaciones relacionadas con deportes y actividades como ciclismo y senderismo u otro tipo de aplicaciones como vigilancia.

La URL del servicio de elevación de Google Maps
es: http://maps.googleapis.com/maps/api/elevation/output?parameters

Los parámetros son path en el que se indican una ruta a través de una serie de puntos con sus coordenadas latitud,longitud y samples que se indica cuántas muestras se desean obtener en esa ruta.

Un ejemplo del resultado que se obtiene en formato JSON es el siguiente:


```
{
"results" : [
      {
         "elevation" : 1608.637939453125,
         "location" : {
            "lat" : 39.73915360,
            "lng" : -104.98470340
         },
         "resolution" : 4.771975994110107
      }
   ],
   "status" : "OK"
}
```


Destaca en este ejemplo el valor resolution que indica la distancia máxima (en metros) entre los puntos de datos desde los que se interpoló la elevación. Cuantos más puntos se consultas, la resolución es más alta, si se desea obtener una elevación más precisa, el punto se debería consultar de forma independiente, obteniendo así un valor mínimo de resolución.

API de codificación geográfica

Esta API es la encargada de codificar direcciones textuales como "Doctor López Ibor, 8" en coordenadas latitud,longitud. También se puede solicitar la conversión inversa para obtener la dirección textual a partir de coordenadas latitud,longitud.

https://maps.googleapis.com/maps/api/geocode/output?parameters

Los parámetros básicos son address o latlng. Se incluirá uno u otro dependiendo de si se desea codificación a coordenadas o inversa. Obligatoriamente también hay que incluir el parámetro sensor con el que se indica si la solicitud procede de un dispositivo con un sensor de ubicación.

API de Google Places

Esta API proporciona información detallada sobre sitios. A diferencia de las demás APIs de los servicios webs de Google Maps, ésta si requiere la API KEY del proyecto de aplicación. La URL del servicio web es: https://maps.googleapis.com/maps/api/place/details/output?parameters

Los parámetros son key, reference, sensor y language. El parámetro reference indica el identificador textual exclusivo del sitio que se obtiene al enviar una solucitud de búsqueda

del sitio. Existe otro servicio para responder a solicitudes de búsquedas de sitios que recoge información como la posición y el radio y devuelve la cadena reference necesaria.

La respuesta de la API de Google Places contiene información tan valiosa como el número de teléfono, la web o el horario de apertura del sitio si toda esta información se encuentra disponible.

API de matriz de distancia

El último servicio web que ofrece Google Maps API es el servicio de matriz de ditancia. Este servicio proporciona la distancia y tiempo estimado de viaje entre una matriz de puntos de origen y de destino. La URL para obtener información del servicio es la siguiente: https://maps.googleapis.com/maps/api/distancematrix/output?parameters

Los parámetros son origins, destinations, sensor, mode, language, avoid y units. Tanto origins como destinations admiten un conjunto de puntos latitud,longitud. El parámetro mode a diferencia de la API de rutas sólo admite walking, driving y bicycling no pudiendo calcular la distancia en transporte público.

El servicio de matriz de distancias devuelve como resultado un conjunto de rows que contiene un conjunto elements con campos status, duration y distance. El dato que interesa para la aplicación es el valor del campo value del campo distance.

3.3.7. Referencia 3. La API de MapQuest

MapQuest ofrece un servicio web basado en HTTP para que los usuarios de la API puedan consultar rutas entre dos puntos geolocalizados con latitud y longitud. No sólo ofrece la ruta entre los dos puntos calculada sobre el mapa de OpenStreetMaps si no que ofrece además instrucciones para un posible servicio de GPS y los diferentes tramos de la ruta con su distancia correspondiente.

La URL a la que se realiza la petición GET para utilizar el servicio se muetra a continuación http://open.mapquestapi.com/directions/v1/route?. Los parámetros de la petición son descritos a continuación:

- **key**

  Es la clave de la aplicacion proporcionada por MaqQuest para poder hacer uso del servicio. Todas los servicios web de MapQuest requieren incluir la clave para tener constancia de qué aplicación hace la petición.

- **callback**

  Parámetro para indicar el nombre de una función Javascript en la que se envolverá la respuesta del servicio para poder utilizar las capacidades de JSONP. Este parámetro no es nada útil para la aplicación que se ejecuta desde una máquina virtual Java y no desde un navegador con Javascript.

- **outFormat**

  Parámetro para indicar json o xml como formato de la respuesta. La notación Javascript JSON es más ligera en tanto que para la misma respuesta se consumen menos caracteres y para una aplicación Javascript es trivial utilizar la respuesta como un objeto. Sin embargo, se podría desear recibir la respueta en XML ya que su formato es más entendible para nosotros. La aplicación requiere la respuesta en JSON que además al ser el parámetro por defecto, no es necesario indicar este requerimiento.

- **routeType**

  Condiciona la búsqueda de la ruta. Los valores posibles para el parámetro son: fastest, shortest, pedestrian, multimodal y bycicle. El valor multimodal contempla el uso de bicicleta o de transporte público. Ya que el usuario desde el dispositivo móvil indica si camina o se mueve en vehículo, los valores que se usarán desde la aplicación serán "pedestrian" y "fastest". Este último además es el parámetro por defecto.

- **timeType, dateType, date y localTime**

  Son cuatro parámetros estrechamente relacionados. El parámetro timeType permite indicar None con 0 como valor del parámetro en la URL, Current time = 1 y Custom time = 2. La finalidad del parámetro timeType es obtener la respuesta basada en la hora local del servicio o en la hora local del usuario. Si se quiere la respuesta en la hora local del usuario es necesario escribir el valor 2 que es Custom time y además incluir los parámetros dateType, date y localTime. La fecha deseada se puede indicar con dateType = 0 y el parámetro adicional date con la fecha como por ejemplo 04/14/2011. El parámetro localTime incluye la hora como por ejemplo 12:00.

- **enhancedNarrative y narrativeType**

  Parámetro booleano que si se marca como true, la respuesta incluirá más información como número de intersecciones, información sobre la intersección anterior y siguiente. Si se desea obtener esta información adicional, se requiere que el parámetro

narrativeType sea microformat. Las otras opciones para narrativeType son none, text y html.

- **shapeFormat**

  Este parámetro configura los puntos (latitud,longitud) de la ruta para poder dibujar la figura en un espacio de coordenadas. Los posibles valores son none, raw, cmp y cmp6 ambos establecen la ruta como una cadena de puntos, cmp con puntos de 5 dígitos de precisión y, en cambio, cmp6 con 8 dígitos de precisión. Para calcular la distancia entre dos puntos, esta información no es necesaria así que se pasará por parámetro none desde la aplicación.

- **generalize**

  Cuanto mayor es este parámetro entero positivo, más se reducirá el número de puntos en la figura de la ruta. El valor 0 no simplificará los puntos de la figura calculada.

- **locale**

  Este parámetro indica al servicio el idioma de la narrativa de la ruta. Los valores posibles son los contemplados por la norma ISO 639-1. Por defecto, el idioma es inglés de Estados Unidos. Para el cálculo de distancias no se requiere la narrativa así que este parámetro es indiferente.

- **units**

  Especifica la unidad de medida de la distancia. m indica millas, en cambio, k solicita que la unidad de distancia se mida en kilómetros.

- **from y to**

  Parámetros para el punto de origen y el punto/los puntos destino. Se pueden enviar varios parámetros to. Los puntos se indican de la forma latitud,longitud donde latitud y longitud son números decimales positivos.

- **drivingStyle**

  Este parámetro especifica el estilo de conducción para configurar la velocidad de movimiento en el cálculo de la ruta más rápida. Los posibles valores son 0 o cautious, 2 o normal y 3 o aggressive.

- **highwayEfficiency**

  Este parámetro se usa para indicar el consumo del vehículo en autopista. La unidad de medida es milla/galón.

3.4. Comercio electrónico desde dispositivos móviles

El objetivo de la librería de pagos para la plataforma Android es facilitar integrar diferentes pasarelas de pago en una misma aplicación Android para que comprador y vendedor tengan flexibilidad y capacidad de elección para cobrar o comprar.

Debido a que la heterogenia del software de desarrollo para la integración de servicios de pago en aplicaciones Android es notable, implementar una librería en la que se hayas tenido en cuenta tal heterogenia no es trivial.

Sea el caso de Paypal que entre muchas APIs, desde de 2013 ofrece un SDK para Android que consiste en una librería con sus propias actividades. Por ello, en el momento de pago, elegida Paypal, la librería ejecutará la actividad principal del SDK de Paypal y cederá el flujo de la aplicación a ésta.

En cambio, Coinbase no dispone de SDK para Android y ofrece una interfaz REST sobre HTTP con la que se comunica la librería sin necesidad de actividades que tomen el control de la aplicación.

Así, teniendo en cuenta las diferencias entre mecanismos de pago de los servicios abarcados, la tarea del diseño de una librería de pagos se traslada a un plano superior en el que es preciso definir un mecanismo de pago común para cualquier aplicación Android al que se le pueda adaptar cualquier otro servicio de pago. La librería seguirá el siguiente proceso de pago:

1) Inicio del pago En el que se instanciará el servicio de pago elegido y se le pasarán a este los parámetros necesarios para realizar el pago.
2) Realización del pago: En esta etapa, el comprador interactúa con el servicio de pago que puede ser parte de la librería como Coinbase o comprender una librería externa como el SDK de Paypal. El comprador se autenticará en el servicio y realizará el pago.
3) Envío del recibo al servidor: La librería obtiene el recibo de pago generado por el servicio de pago y lo envía al servidor de la aplicación.
4) Verificación del recibo: En el servidor, el recibo se confirma verificando con el servidor del servicio de pago.
5) Confirmación del pago: Se notifica al usuario el éxito del pago.

3.4.1. Diseño de la librería para la aplicación móvil

Del proceso de pago descrito anteriormente, las tres primeras etapas y la última pertenecen al ámbito de la aplicación móvil. De estas etapas, la segunda es la más

importante. La librería debe permitir la realización del pago a través de diversos servicios de pago que se puedan integrar. Para facilitar la integración se ha diseñado la interfaz IServicioPago que es el contrato que tienen que cumplir los servicios de pago para que puedan ser integrados en la librería. La interfaz es la vía de comunicación entre la librería y el servicio de pago. A través de ella, la librería pasará la información de pago y recogerá el recibo y el resultado del pago.

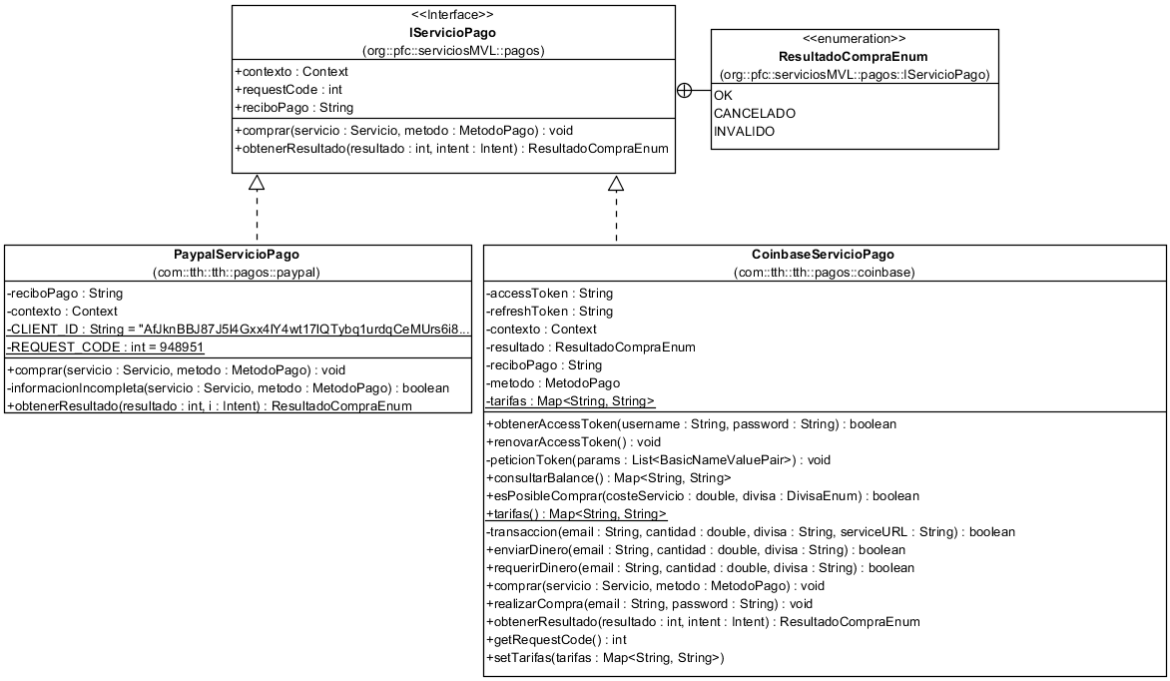


Diagrama de clases de la interfaz IServicioPago

El método setContext() de la interfaz IServicioPago permitirá a la implementación de la interfaz hacer uso de los métodos del contexto de la aplicación Android. El método comprar() recibe como argumentos una instancia de Servicio y una instancia de MetodoCompra. La implementación obtendrá el nombre del servicio de la instancia Servicio y el importe y la divisa de la instancia MetodoPago.

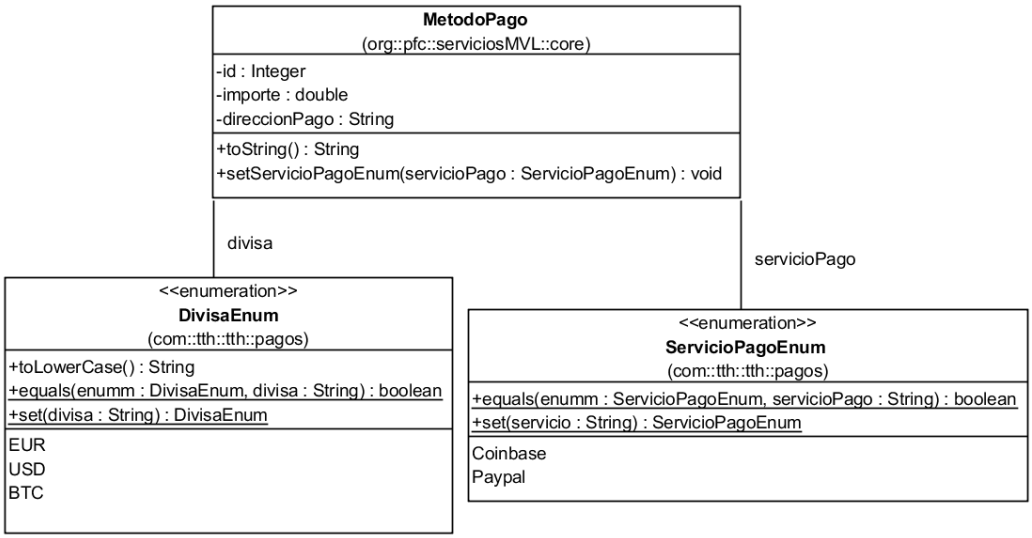


Diagrama de clases de MetodoPago

En este diagrama de clases se incluye la clase MetodoPago. La información que almacena las instancias de esta clase incluye el importe a pagar, la dirección de pago -que suele ser una dirección de correo de un usuario registrado en el servicio de pago-, el enumerado que indica la divisa elegida y el enumerado que indica el servicio de pago escogido. Las posibles divisas son euros, dólares y bitcoins. Los servicios de pago que incluye la librería son Paypal y Coinbase.

Diseño de PaypalServicioPago

En el caso de que el servicio de pago sea Paypal, el método comprar() lanzará la actividad principal del SDK de Paypal para que el usuario se autentique y realice el pago. En el caso de que el servicio de pago sea Coinbase, se lanzará una actividad desarrollada dentro de la librería para que el usuario se autentique en Coinbase y posteriormente se realizará la conexión con la API HTTP REST de Coinbase para realizar el pago.

Sea cual sea el servicio de pago elegido, el método obtenerResultado() proporcionará a la librería el resultado tras la compra. Para esto se define en la interfaz un enumerado ResultadoCompraEnum con los posibles valores OK, CANCELADO o INVALÍDO si el pago ha sido rechazado por alguna razón desde Paypal o Coinbase. Si el pago ha resultado OK a través del método getReciboPago() se podrá obtener un recibo de pago elaborado por el servicio de pago para ser más tarde verificado en el servidor de la aplicación. Si se desea extender la librería con añadiendo otro servicio de pago tan sólo es necesario desarrollar una nueva implementación de la interfaz IServicioPago.

La implementación para la pasarela de pago de Paypal es PaypalServicioPago que hace uso del SDK de Paypal para Android. Por ello, el método comprar() lanza la actividad

PaymentActivity del SDK, no sin antes comprobar que la información necesaria para el proceso de pago con Paypal esté cubierta con el método informacionCompleta() que valida la instancia Servicio y la instancia MetodoPago.

Cuando el pago se completa, el SDK de Paypal termina su ejecución y envía el resultado a la actividad que lo lanzó. Esta actividad recoge tanto el valor entero que indica el éxito de la ejecución como información extra almacenada en una instancia Intent. Para notificar a la implementación PaypalServicioPago de esta información hace uso del método obtenerResultado(). Posteriormente, se recoge el recibo de pago para enviarlo al servidor y que compruebe su validez con el método getReciboPago()

El uso de la implementación PaypalServicioPago y de las implementaciones de IServicioPago quedará más claro tras observar el diagrama de secuencia de cómo realizar un pago a través de Paypal con la librería:

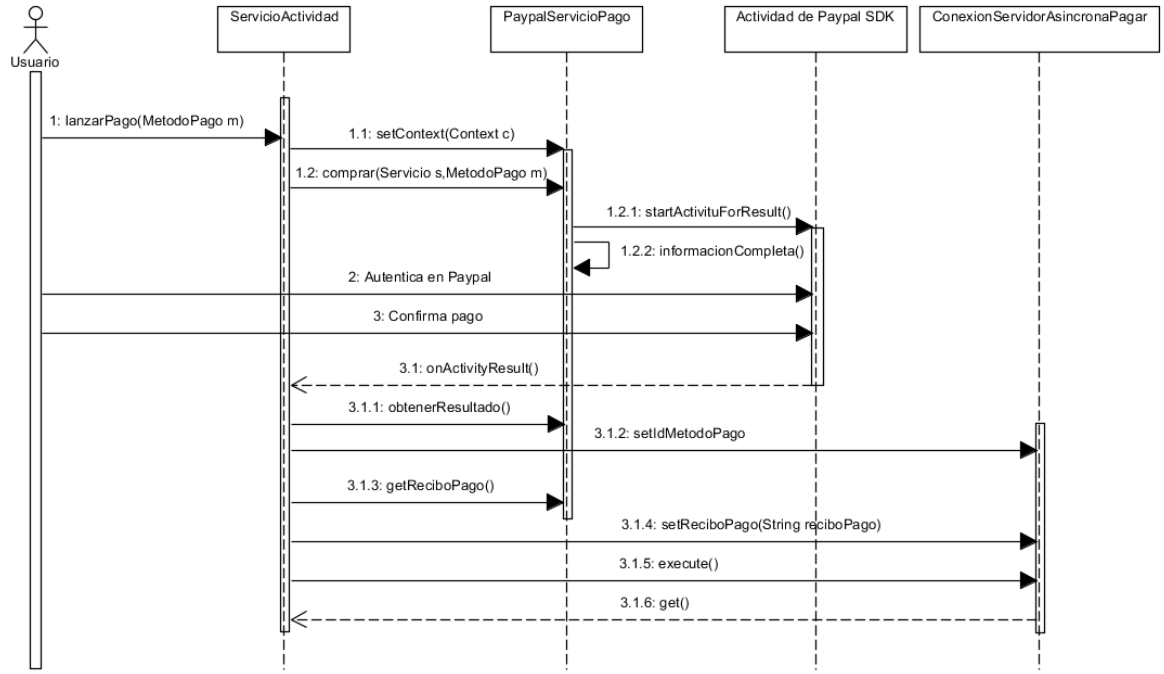


Diagrama de secuencia de un pago con IServicioPago

Como se aprecia en el diagrama de secuencia anterior, tras solicitar a la interfaz IServicioPago el recibo de pago, se hace uso de la clase ConexionServidorAsincronaPagar para enviar el recibo junto con el identificador del método de pago y que desde el servidor se verifique el pago y se obtenga una respuesta a través del método get() de esta clase. Posteriormente, se notificará al usuario el éxito o fracaso de la compra.

Diseño de CoinbaseServicioPago

La responsabilidad de la clase CoinbaseServicioPago consiste en generar el mensaje que recibirá la API HTTTP REST de Coinbase para procesar el pago con bitcoins. Además, gestiona la comunicación con esta API que utiliza el protocolo de autenticación OAuth 2 para que aplicaciones autorizadas por el usuario del servicio puedan hacer uso de información de este servicio sin comprometer la seguridad. Los métodos y propiedades de la clase CoinbaseServicioPago son los siguientes:

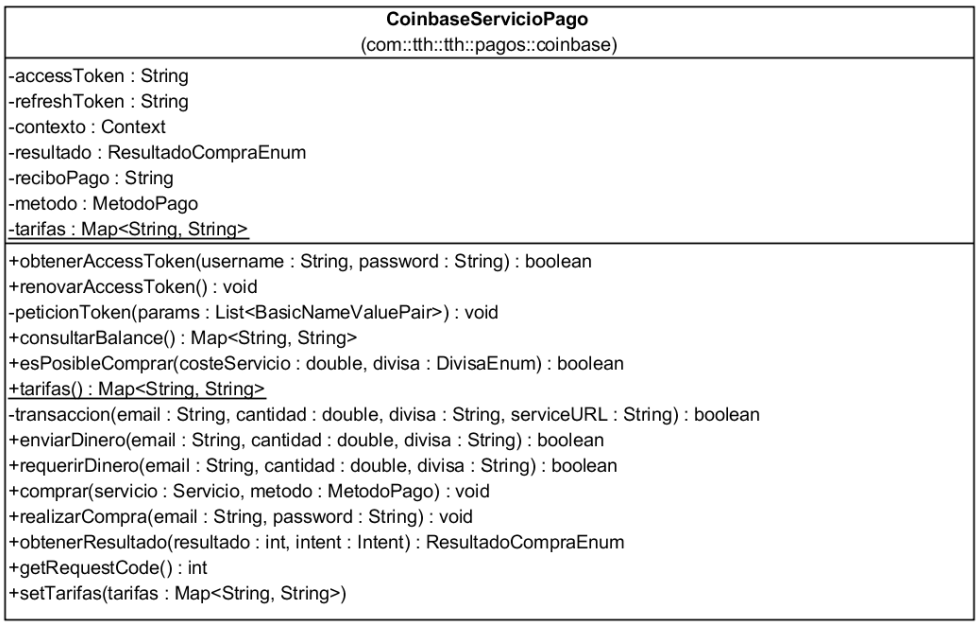


Diagrama de clases de CoinbaseServicioPago

Antes de poder enviar o recibir información de la API es necesario obtener un Access token. Con este fin se han diseñado los métodos obtenerAccessToken() y renovarAccessToken(). El primer método usa el nombre de usuario y la contraseña del usuario para obtener el Access token que guarda en el String accessToken. Al recibir el Access token también se recibe un refresh token que se alamacena en la propiedad refreshToken para utilizarlo pasado un tiempo con el método renovarAccessToken().

Una vez esté el usuario autenticado en el servicio Coinbase, desde la librería se podrá consultar el balance de bitcoins con el método consultarBalance(), consultar la tasa de intercambio entre Bitcoin y otras divisas con tarifas(), enviar bitcoins a través de enviarDinero() o solicitar el envío de bitcoins a un usuario con el método requerirDinero().

Al igual que con la implementación de la pasarela de Paypal, la clase ServicioActividad instanciará CoinbaseServicioPago y ejecutará comprar(), método que lanzará la actividad para que el usuario del dispositivo móvil se pueda autenticar en Coinbase introduciendo

nombre de usuario y contraseña. Desde esta actividad se llamará al método realizarCompra() que contiene la principal lógica de la clase que se modela en el siguiente diagrama de flujo:

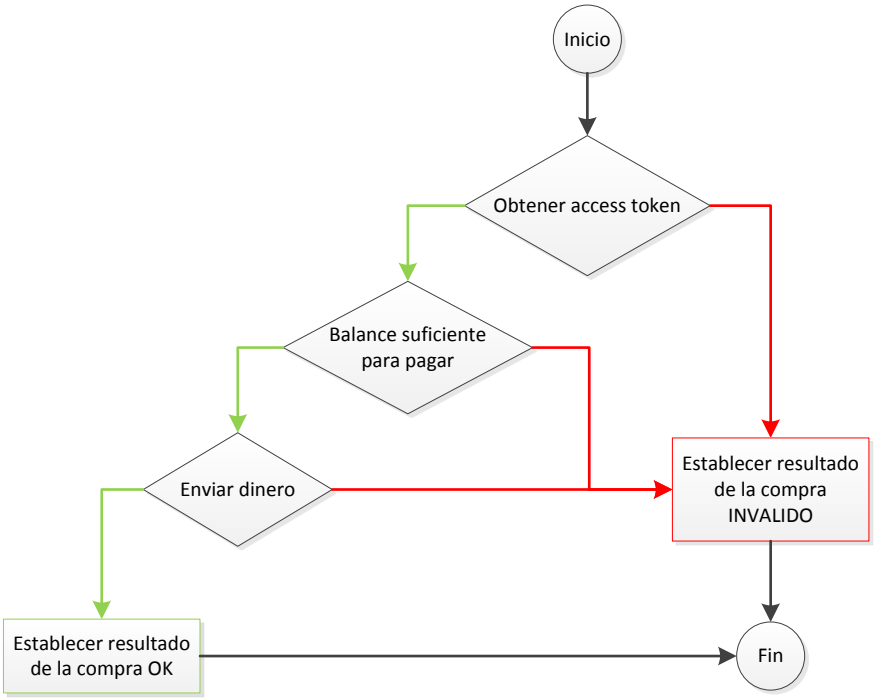


Ilustración 1Diagrama de flujo de realizarCompra()

La conexión con la API REST de Coinbase se realiza desde la clase CoinbaseServicePeticion que hereda de AsyncTask para realizar la conexión en un hilo diferente del que dibuja las actividades en pantalla.

3.4.2. Implementación de la librería para la aplicación móvil

Implementación de PaypalServicioPago

La implementación de la clase PaypalServicioPago es sencilla. El primer paso es incluir la librería Paypal Android SDK en el proyecto Android. A continuación, es necesario declarar en el archivo de configuración AndroidManifest.xml el servicio y las actividades de Paypal:

```
<service android:name="com.paypal.android.sdk.payments.PayPalService"
    android:exported="false" />

<activity android:name="com.paypal.android.sdk.payments.PaymentActivity" />
<activity android:name="com.paypal.android.sdk.payments.LoginActivity" />
<activity android:name="com.paypal.android.sdk.payments.PaymentMethodActivity" />
<activity android:name="com.paypal.android.sdk.payments.PaymentConfirmActivity" />
```

Se hace uso de este SDK en el método comprar() de PaypalServicioPago. En este método primero se comprueba que se tengan todos los datos necesarios para la compra y posteriormente se lanza la actividad habiéndole pasado varios parámetros de configuración como es el parámetro EXTRA_PAYPAL_ENVIRONMENT que permite configurar el SDK para que las transacciones se realicen con dinero virtual en un entorno de pruebas conocido como Paypal Sandbox. Esta opción es la que se utiliza en la librería y se muestra en el código que sigue. Una vez que la aplicación que incluya la librería entre en estado de producción se podrá cambiar este parámetro por ENVIRONMENT_PRODUCTION.

El parámetro CLIENT_ID identifica la aplicación que utiliza el SDK de Paypal y se obtiene en el sitio web de Paypal para desarrolladores. El parámetro EXTRA_PAYER_ID identifica el comprador internamente en la aplicación. No es un parámetro necesario para el proceso de pago con Paypal pero Paypal lo incluirá en el recibo de pago, así la aplicación que incluya la librería podrá tener otro dato que ayude a verificar el comprador y evitar el fraude.

Por último, referente al uso de Paypal SDK en el método comprar(), es remarcable el parámetro EXTRA_RECEIVER_PAYMENT que se configura con la cuenta de Paypal a la que se pagará el importe. Las cuentas de Paypal se identifican con la dirección de correo con la que se registró el usuario. Este parámetro permite establecer como cobrador a otro que no sea el propietario de la aplicación. A diferencia de otros muchos servicios de pago, este parámetro permite que Paypal sea una pasarela de pagos lo suficientemente flexible para permitir el pago entre usuarios de Paypal dentro de una aplicación. Por lo general, la mayoría de servicios de pago se enfocan en el modelo de tienda en el que el vendedor es sólo uno, el propietario de la aplicación.

```
public void comprar(Servicio servicio,MetodoPago metodo) {
    if (informacionIncompleta(servicio,metodo)) return;

    PayPalPayment payment = new PayPalPayment(new BigDecimal(metodo.getImporte()),
        metodo.getDivisa(),servicio.getNombreServicio());

    Intent intent = new Intent(contexto,PaymentActivity.class);
    intent.putExtra(PaymentActivity.EXTRA_PAYPAL_ENVIRONMENT,PaymentActivity.ENVIRONMENT_SANDBOX);
    intent.putExtra(PaymentActivity.EXTRA_CLIENT_ID,CLIENT_ID);
    intent.putExtra(PaymentActivity.EXTRA_PAYER_ID,MenuPrincipal.usuario.getEmail());
    intent.putExtra(PaymentActivity.EXTRA_RECEIVER_EMAIL,metodo.getDireccionPago());
    intent.putExtra(PaymentActivity.EXTRA_PAYMENT,payment);
    ((Activity)contexto).startActivityForResult(intent,REQUEST_CODE);
}
```

Cuando termina la ejecución del SDK de Paypal, envía el resultado de esta a la actividad que lanzó la actividad de Paypal. Entonces se llama al método obtenerResultado() de PaypalServicioPago. La implementación de este método es como sigue:

```
public ResultadoCompraEnum obtenerResultado(int resultado,Intent i) {
    if (resultado == Activity.RESULT_OK) {
        PaymentConfirmation confirm = i.getParcelableExtra(PaymentActivity.EXTRA_RESULT_CONFIRMATION);
        if (confirm != null) {
            try {
                reciboPago = confirm.toJSONObject().toString();
            } catch(Exception e) {
                Log.e(this.getClass().getName(),e.getLocalizedMessage(),e);
            }
        }
        return ResultadoCompraEnum.OK;
    }
    else if (resultado == Activity.RESULT_CANCELED)
        return ResultadoCompraEnum.CANCELADO;
    else return ResultadoCompraEnum.INVALIDO;
}
```

Paypal aprovecha las constantes de Android para indicar el resultado del pago a través del argumento, de tipo entero, resultado. La librería encapsula dichas constantes en el enumerado ResultadoCompraEnum. Si el pago se ha realizado correctamente, recoge el recibo de pago desde la estructura de datos que almacena la actividad en el argumento i de tipo Intent.

Implementación de CoinbaseServicioPago

La implementación del proceso de pago con el servicio Coinbase es algo más compleja. El punto de entrada es el método comprar() que lanza la actividad ComprarConCoinbaseActividad para que el usuario pueda introducir su correo electrónico y la contraseña de su cuenta en el servicio Coinbase. Una vez el usuario haya introducido dichos datos, la actividad terminará llamando al método realizarCompra() y pasando como argumentos el email y la contraseña obtenidos.

```
public void realizarCompra(String email,String password) {
    if (!obtenerAccessToken(email, password)) {
        resultado = ResultadoCompraEnum.INVALIDO;
        return;
    }

    if (esPosibleComprar(metodo.getImporte(),DivisaEnum.valueOf(metodo.getDivisa()))) {
        if (enviarDinero(metodo.getDireccionPago(),
            metodo.getImporte(),metodo.getDivisa())) {
            Log.i(this.getClass().getName(),"Éxito enviando dinero.");
            resultado = ResultadoCompraEnum.OK;
        } else {
            Log.i(getClass().getName(),"No se pudo enviar el dinero.");
            resultado = ResultadoCompraEnum.INVALIDO;
        }
    } else {
        Log.i(getClass().getName(),"No puede comprar el servicio.");
        resultado = ResultadoCompraEnum.INVALIDO;
    }
}
```

El primer paso es obtener el Access token para lo que se envía una petición HTTP GET con el nombre de usuario y contraseña a la API oauth/token. Los parámetros del mensaje

son CLIENT_ID y CLIENT_SECRET, dos claves proporcionadas al registrar la aplicación en el servicio de Coinbase. En la nomenclatura del protocolo OAuth, estas claves también se suelen denominar CONSUMER_ID y CONSUMER_SECRET respectivamente. La respuesta del servidor de Coinbase permite establecer las propiedades accessToken y refreshToken.

Posteriormente se hace uso del método consultarBalance() a través del método esPosibleComprar() que verificará que haya saldo suficiente para realizar el pago. El método consultarBalance() realiza una petición GET a la API api/v1/account/balance. El resultado, que es un objeto JSON, se convierte en un mapa y se obtiene de éste el campo amount. De la misma manera se obtiene un mapa de las tarifas de intercambio entre Bitcoin y otras divisas haciendo uso del método tarifas() que de forma asíncrona crea un mapa con la respuesta de la API api/v1/currencies/rate_exchanges. La propiedad tarifas se establece de forma asíncrona tras ejecutar tarifas() a través de setTarifas() desde la clase CoinbaseServicePeticion. Este mapa es necesario ya que el valor amount del balance es devuelto en bitcoins, no obstante la divisa del servicio puede ser EUR o USD. Desde la clase de apoyo App también se puede obtener el mapa de tarifas a través del método getTarifas() en dicha clase.

Tras satisfacer la comprobación de que hay saldo suficiente, se llama al método enviarDinero() con los argumentos direccionPago, importe y divisa. La dirección de pago es un correo electrónico con el que el vendedor se registró en Coinbase. El método enviarDinero() internamente ejecuta el método transacción() pasándole un argumento más que es el recurso HTTP POST de la API para enviar dinero: api/v1/transactions/send_money.

El método transaccion() es el encargado de enviar un objeto JSON tanto para enviar como para solicitar dinero al servidor de Coinbase y comprobar el éxito de la transacción en la respuesta del servidor. Crea una petición HTTP POST en la que añade como cabeceras los parámetros Accept y Content-Type que son ambos application/json. Incluye el parámetro Authorization con el valor Bearer concatenando el Access token según el protocolo OAuth 2. De la respuesta de la transacción, que es otro objeto JSON, extrae los campos id, hsh, status, errors y success. Como cabe esperar, el campo success a true denota que la transacción ha tenido éxito. Sin embargo, el campo status en un primer momento indicará que la transacción está pendiente a completarse (PENDING) y en un corto espacio de tiempo pasará a COMPLETE.

Si hubo éxito en el envío de dinero, el método enviarDinero() devolverá true y además transacción() establecerá la propiedad reciboPago. En este punto, el método realizarCompra() establecerá la propiedad resultado con el enumerado OK de tipo ResultadoCompraEnum.

El método realizarCompra() por sí sólo es capaz de determinar que la compra se ha realizado con éxito o que la compra ha sido rechazada por el servicio de pago. La tercera posibilidad es que el usuario haya cancelado la compra desde la actividad ComprarConCoinbase. Esta posibilidad se evalúa en la implementación del método obtenerResultado() que comprueba si al terminar la actividad se finalizó con la constante RESULT_CANCELED:

```
public ResultadoCompraEnum obtenerResultado(int resultado, Intent intent) {
    if (resultado == Activity.RESULT_CANCELED) return ResultadoCompraEnum.CANCELADO;
    return this.resultado;
}
```

Uso de IServicioPago desde ServicioActividad

La actividad ServicioActividad se lanza tras pulsar sobre un servicio del que se desea ver sus detalles. Si al servicio le han sido añadidos métodos de pago para comprarlo, aparecerá un botón con cada método de pago. Al pulsar sobre algún botón “Comprar con…” se ejecuta el método lanzarPago() pasándole como argumento el método de pago asociado al botón. Desde dicho método, se instancia la implementación de IServicioPago elegida y se le envía a dicha instancia el contexto de la aplicación Android.

Al terminar la ejecución del SDK de Paypal o el pago a través de CoinbaseServicioPago, se ejecuta onActivityResult(). El código del método es el siguiente:

```
protected void onActivityResult(int requestCode, int resultCode, Intent i) {
    ResultadoCompraEnum resultado = servicioPago.obtenerResultado(resultCode, i)

    String servicioEscogido = metodoPago.getServicioPago();

    switch (resultado) {
    case INVALIDO:
        if (ServicioPagoEnum.equals(ServicioPagoEnum.Coinbase,servicioEscogido))
            lanzarResultadoCompra(false);
        }
    case CANCELADO:
        ocultarServicio(false);
        return;
    }

    if (servicioPago.getReciboPago() == null) {
        lanzarResultadoCompra(false);
        return;
    }

    ConexionServidorAsincronaPagar
    c = new ConexionServidorAsincronaPagar(this);
    c.setIdMetodoPago(metodoPago.getId().intValue());
    c.setReciboPago(servicioPago.getReciboPago());
    c.execute();

    boolean exito = false;
    try {
        exito = c.get() && c.getServicio() != null;
    } catch (Exception e) {
        Log.e(getClass().getName(),getString(R.string.servicio_no_comprado),e);
    }

    lanzarResultadoCompra(exito);
}
```

En primer lugar, el método recoge el resultado del proceso de pago de la instancia de IServicioPago y nombre del servicio de pago que ha escogido el usuario. Dependiendo de este resultado y del servicio de pago elegido realizará una o varias tareas:

<table>
<tr><th colspan="2" rowspan="2"></th><th colspan="3">Resultado del proceso de pago</th></tr>
<tr><th>OK</th><th>INVALIDO</th><th>CANCELADO</th></tr>
<tr><td rowspan="2">Servicio de pago</td><td>Paypal</td><td rowspan="2">Solicitar verificación
Informar al usuario</td><td>Hacer visible el servicio</td><td rowspan="2">Hacer visible el servicio</td></tr>
<tr><td>Coinbase</td><td>Hacer visible el servicio
Informar al usuario</td></tr>
</table>

Acciones a realizar según el resultado y el servicio de pago

Para solicitar la verificación del pago, el método utiliza la clase ConexionServidorAsincronaPagar que envía una petición HTTP POST al servidor de la aplicación con el recibo de pago y el identificador del servicio que se quiere verificar.

Nótese que es necesario informar al usuario si el servicio de pago es Coinbase y este servicio ha rechazado la transferencia. Con la pasarela de pago Paypal no se requiere porque el SDK informa al usuario a través de su propia interfaz. El método lanzarResultadoCompra() es el encargado de lanzar la actividad informando al usuario si la compra se ha realizado con éxito y ha podido ser verificada o si, por el contrario ha fracasado o no se ha podido verificar.

El método ocultarServicio() puede ocultar el servicio o hacerlo visible dependiendo del argumento booleano ocultar que se le pase. Si el parámetro es true, la aplicación solicitará al servidor que oculte el servicio porque el usuario entrará en el proceso de compra. En cambio, si el parámetro es false tal como en el código del método onActivityResult(), la aplicación solicitará al servidor que vuelva a hacer visible el servicio ya que éste finalmente no ha sido comprado.

3.4.3. Diseño de la librería de pagos en el servidor

La librería de pagos en la aplicación Java en el servidor sólo tiene un cometido, que es el de verificar el recibo de pago que ha generado el servicio de pagos cuando un usuario ha comprado un servicio.

La aplicación móvil del usuario envía el recibo de pago junto con el identificador del servicio que se ha pagado y es tarea de la librería en el servidor, comprobar que el recibo es válido y que el pago se ha efectuado.

Realizar la verificación del pago es una recomendación de buenas prácticas por parte del equipo de Paypal. De esta manera se evita que un usuario malintencionado finja la compra de un servicio sin haber realizado la transferencia económica.

No obstante, la librería también comprende la verificación de pagos con otros servicios con los que se quieran extender. Por el momento, hay dos implementaciones incluidas en la librería: Una para verificar pagos que se hayan realizado con Paypal y otra para verificar pagos que se hayan realizado con Coinbase.

La interfaz que es necesario implementar para verificar el pago de un servicio de pago incluido en el paquete de gestión de servicios es la interfaz IVerificacionPago. Esta interfaz sólo compromete a implementar un método, que es verificarPago() que recibe el recibo de pago como String. Este método devuelve un valor booleano, que será true si el

pago se ha verificado correctamente o false si el pago no se ha podido verificar o la verificación ha determinado que el recibo de pago no es válido.

Por lo general, el proceso de verificación del recibo de un pago comprenderá los siguientes pasos:

1. Extracción de la información del recibo
2. Envío de la información del recibo al servicio de pago
3. Comparación de la respuesta del servicio de pago con el recibo

Las dos implementaciones de la interfaz IVerificacionPago incluidas en la librería son PaypalVerificacionPago y CoinbaseVerificacionPago. El diagrama de clases facilitará la comprensión de la estructura:

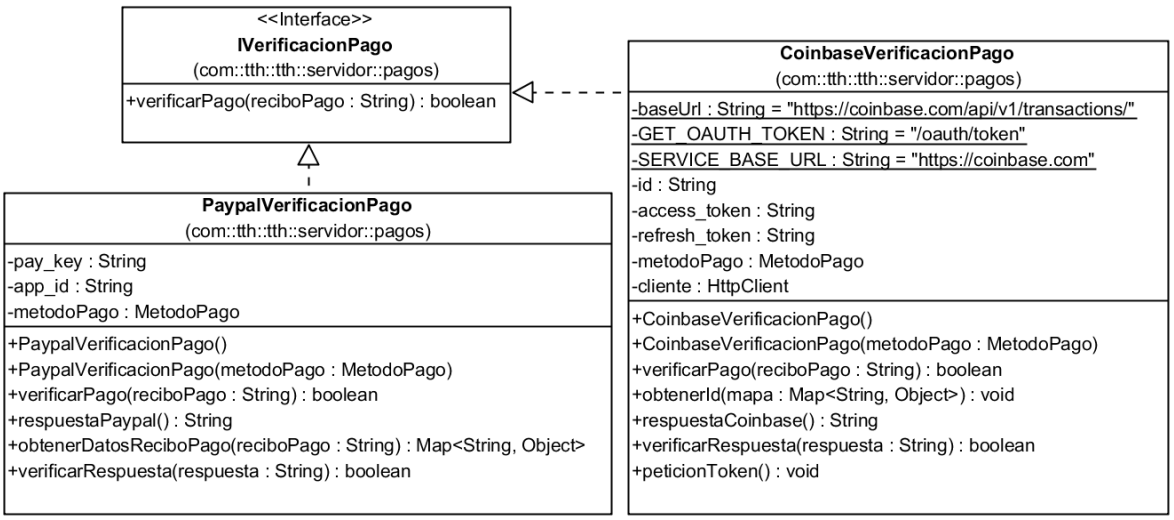


Diagrama de clases de la interfaz IVerificacionPago

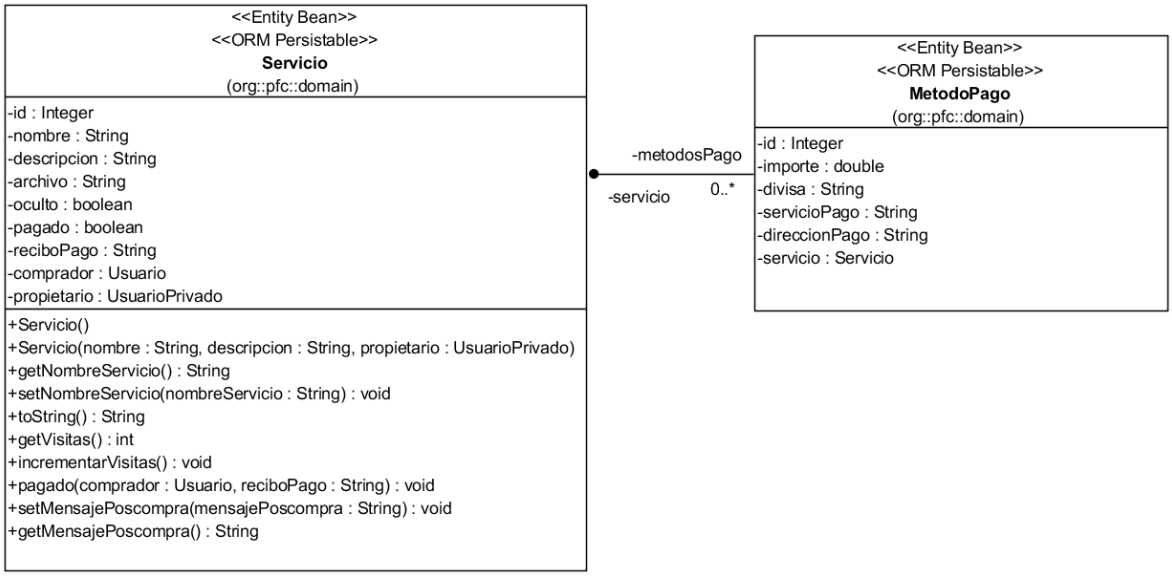
Ambas implementaciones tienen bastantes similitudes. Por ejemplo, ambas tienen un constructor sin argumentos y otro al que se le pasa una instancia MetodoPago. El segundo constructor permite verificar además la divisa, el receptor y el importe. Por ello es recomendable usar este segundo constructor para instanciar la clase de verificación.

En cuanto a la clase PaypalVerificacionPago, primero ejecuta obtenerDatosReciboPago(), método que toma el recibo de pago como String y extrae de este las propiedades pay_key y app_id. Posteriormente envía estos datos al servidor de Paypal con el método respuestaPaypal() y comprueba en esta respuesta que el estado de la transacción sea completado, el importe, la divisa y el receptor. Si algún dato no se puede validar o difiere el método verificarRespuesta() devuelve false y por tanto, el método verificarPago() también devuelve false.

La clase CoinbaseVerificacionPago, por su parte, extrae el identificador de la transacción del recibo de pago que es un String que contiene un objeto JSON con el método obtenerId(). Con este identificador se solicita al servicio Coinbase el recibo de pago con el método respuestaCoinbase(). De este recibo se comprueba que el identificador de la transacción, el importe, la divisa y el receptor coincidan y además que el estado de la transacción sea PENDING o COMPLETE.

Por otra parte, para que el paquete de servicios soporte la compra de servicios a través de servicios de pago, es necesario ampliar tanto los elementos de dominio, la persistencia de estos y la interfaz de conexión con el servidor.

Los servicios pueden ser pagados con diferentes métodos de pago y cuando un servicio es comprado se persiste el recibo del pago del servicio. Los métodos de pago, al igual que en la aplicación móvil, almacenan el importe, la divisa, el servicio de pago y la dirección del vendedor.



Relación Servicio - MetodoPago

La interfaz de conexión con el servidor cuenta con un punto de conexión adicional servicios/pay desde el que se recoge el recibo de pago junto con el identificador del servicio pagado y el identificador del usuario que ha comprado el servicio.

3.4.4. Implementación de la librería de pagos en el servidor

Implementación de PaypalVerificacionPago

En la clase PapypalVerificacionPago, el método verificarPago() coordina los demás métodos de la clase. La implementación es:

```
public boolean verificarPago(String reciboPago) {
    obtenerDatosReciboPago(reciboPago);
    return verificarRespuesta(respuestaPaypal());
}
```

El recibo de pago de Paypal que recibe el servidor es un objeto JSON. Es fácil convertirlo en un mapa de mapas y listas y navegar dentro de él hasta recoger los datos deseados. En este caso pay_key que es el identificador del pago y app_id que es el identificador de la aplicación que utiliza la pasarela de pagos.

```
public Map<String, Object> obtenerDatosReciboPago(String reciboPago) {
    if (reciboPago == null) return null;

    JsonFactory factory = new JsonFactory();
    ObjectMapper mapper = new ObjectMapper(factory);
    TypeReference<HashMap<String, Object>> typeRef =
        new TypeReference<HashMap<String, Object>>() {};

    Map<String, Object> mapa = null;
    try {
        mapa = mapper.readValue(
            new ByteArrayInputStream(reciboPago.getBytes("UTF-8")), typeRef);
    } catch (Exception e) {
        e.printStackTrace();
    }

    Map proof = (Map)mapa.get("proof_of_payment");
    Map adaptive = (Map)proof.get("adaptive_payment");
    pay_key = (String)adaptive.get("pay_key");
    app_id = (String)adaptive.get("app_id");

    return mapa;
}
```

Para obtener el recibo de pago desde el servidor de Paypal es necesario realizar una petición GET con ciertas cabeceras como son el X-PAYPAL-SECURITY-USERID, X-PAYPAL-SECURITY-PASSWORD Y X-PAYPAL-SECURITY-SIGNATURE: nombre de usuario Paypal y la contraseña de la aplicación y el valor SIGNATURE que es único para la aplicación que utiliza la pasarela de pago Paypal.

Si se está utilizando Paypal a través del entorno de pruebas, el nombre de usuario Paypal y contraseña que se pueden incluir en las cabeceras puede ser el de algún usuario creado en la sandbox de Paypal. El código para solicitar el recibo de pago desde Paypal es el siguiente:

```
public String respuestaPaypal() {
    if (pay_key == null || app_id == null) return null;

    StringBuilder url = new StringBuilder();
    url.append(baseUrl);
    url.append("?payKey=");
    url.append(pay_key);
    url.append("&requestEnvelope.errorLanguage=es_ES");

    HttpClient cliente = new DefaultHttpClient();
    HttpGet get = new HttpGet(url.toString());
    get.setHeader("X-PAYPAL-SECURITY-USERID", user_id);
    get.setHeader("X-PAYPAL-SECURITY-PASSWORD", password);
    get.setHeader("X-PAYPAL-SECURITY-SIGNATURE", signature);
    get.setHeader("X-PAYPAL-REQUEST-DATA-FORMAT", "NV");
    get.setHeader("X-PAYPAL-RESPONSE-DATA-FORMAT", "NV");
    get.setHeader("X-PAYPAL-APPLICATION-ID", app_id);

    String respuesta = null;
    try {
        respuesta = EntityUtils.toString(cliente.execute(get).getEntity());
    } catch (Exception e) {
        e.printStackTrace();
    }
    return respuesta;
}
```

Por último, se verifica la respuesta obtenida desde el servidor de Paypal. Lamentablemente, esta respuesta no es un objeto JSON, si no que se encuentra codificada como los parámetros de un recurso HTTP. Esta codificación está regulada por la norma ISO-8859-1. La forma de proceder es utilizar la potencia de la librería de expresiones regulares de Java a través del método contains de la clase String. También se hace uso en este método de la clase URLEncoder para codificar la dirección de correo y buscarla dentro del recibo de pago.

Primero se comprueba que el estado y el estado de la transacción del emisor sean COMPLETED. Entonces se comprueba la divisa, el importe y por último el receptor. En cuanto alguna validación falla, el método termina devolviendo false.

```
public boolean verificarRespuesta(String respuesta) {
    if (respuesta == null) return false;

    if (!respuesta.contains("&status=COMPLETED")) return false;
    if (!respuesta.contains(".senderTransactionStatus=COMPLETED")) return false;
    if (metodoPago == null) return true;

    if (!respuesta.contains("&currencyCode=" + metodoPago.getDivisa())) return false;

    DecimalFormat df = new DecimalFormat("#.##");
    if (!respuesta.contains(".receiver.amount=" + df.format(metodoPago.getImporte()))) return false;

    String email = null;
    try {
        email = URLEncoder.encode(metodoPago.getDireccionPago(), "ISO-8859-1");
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
        logger.error("No se pudo codificar la dir. de pago " + metodoPago.getDireccionPago(),e);
    }
    if (email == null) return true;

    if (!respuesta.contains(".receiver.email=" + email)) return false;
    return true;
}
```

Implementación de CoinbaseVerificacionPago

El método verificarPago() de CoinbaseVerificacionPago es muy similar al de PaypalVerificacionPago:

```
public boolean verificarPago(String reciboPago) {
    obtenerId(JsonMap.toMap(reciboPago));
    return verificarRespuesta(respuestaCoinbase());
}
```

Como es de esperar, el recibo de pago que es un objeto JSON es convertido a un mapa a través del método estático toMap de la clase JsonMap. Este mapa se pasa al método obtener Id que sencillamente tien que extraer el valor del campo id del objeto transaction del mapa.

El método respuestaCoinbase() envía una petición HTTP GET al recurso de la API https://coinbase.com/api/v1/transactions/. El único dato que se proporciona a esta interfaz es el identificador de la transacción obtenido en el método anterior. No obstante, debido a que es una API protegida por el protocolo OAuth 2, es necesario obtener prevaimente un token de acceso e incluirlo en la cabecera Authorization de la petición GET. El código de respuestaCoinbase() es como sigue:

```
public String respuestaCoinbase() {
    if (id == null) return null;

    StringBuilder url = new StringBuilder();
    url.append(baseUrl);
    url.append(id);

    HttpGet get = new HttpGet(url.toString());

    peticionToken();
    get.addHeader("Authorization",String.format("Bearer %s", access_token));

    HttpResponse respuestaGet = null;
    String respuesta = null;
    try {
        respuestaGet = cliente.execute(get);
        respuesta = EntityUtils.toString(respuestaGet.getEntity());
    } catch (Exception e) {
        e.printStackTrace();
        logger.error("No se obtuvo respuesta correcta de Coinbase",e);
        return null;
    }
    return respuesta;
}
```

Al igual que la clase PaypalVerificacionPago, el método verificarRespuesta() analizará la respuesta del servidor de Coinbase para validar la transacción y el recibo de pago recibido en el servidor. En esta ocasión, la respuesta de Coinbase es un objeto JSON y obtener los valores es tan fácil como consultarlos en mapas.

Primero se comprueba el identificador de la transacción, posteriormente se comprueba que el estado de la transacción sea pending o complete. En este punto, si no se ha aportado una instancia del método de pago, es decir, se ha instanciado la clase con el constructor por defecto, la verificación terminará validando el recibo. En cambio, si se ha aportado una instancia del tipo MetodoPago la verificación continuará validando el importe, la divisa y la dirección de correo del receptor. El método se ha implementado así:

```
public boolean verificarRespuesta(String respuesta) {
    if (respuesta == null) return false;

    Map<String,Object> mapa = JsonMap.toMap(respuesta);
    if (mapa == null) return false;

    Map transaction = (Map)mapa.get("transaction");
    if (transaction == null) return false;

    String id = (String)transaction.get("id");
    if (id == null) return false;
    if (!id.equals(this.id)) return false;

    String status = (String)transaction.get("status");
    if (!status.equals("complete") || !status.equals("pending")) return false;

    if (metodoPago == null) return true;

    Map amount = (Map)transaction.get("amount");
    double importe = Double.parseDouble((String)amount.get("amount"));
    if (importe != metodoPago.getImporte()) return false;

    String divisa = (String)amount.get("currency");
    if (!metodoPago.getDivisa().equals(divisa)) return false;

    String email = (String)((Map)transaction.get("recipient")).get("email");
    if (!metodoPago.getDireccionPago().equals(email)) return false;
    return true;
}
```

3.4.5. Referencia 4. Las APIs de Paypal

Paypal es una empresa con una amplia andadura en el sector del comercio electrónica y proporciona una gran variedad de APIs para utilizar la pasarela de pagos desde diferentes medios y con diferentes casos de uso.

Actualmente, la empresa engloba sus interfaces de coenxión en cuatro tipos: Express Checkout API, APIs clásicas, API REST y SDKs móviles. Las dos últimas se pusieron en marcha en 2013 y es posible que aún no estén disponibles en algunos países. Se pueden descargar librerías y ejemplos para utilizar las APIs desde el perfil de Paypal en Github[24].

Paypal Express Checkout API

Es el producto y API más simple que cubre el caso de uso básico. Esta API se utiliza insertando un botón **Pague con Paypal** en la aplicación móvil o aplicación web. Este botón tiene la informacion necesaria para que un comprador, al pulsarlo, pueda iniciar un proceso de pago de un producto o servicio concreto del vendedor.

La API responde a la aplicación del vendedor con una dirección URL a la que el vendedor tiene que redirigir al usuario en un navegador web. Dicha dirección URL pertenece al dominio de Paypal y es donde el comprador introducirá sus datos de Paypal o sus datos financieros para realizar el pago. Nótese que no es necesario que el comprador disponga de cuenta de Paypal.

Esta aplicación web, propiedad de Paypal, en la que se realiza el proceso de pago está diseñada para adaptarse a cualquier tipo de de dispositivo, sea un teléfono móvil, una table o un ordenador de sobre mesa. Así, el vendedor no tiene que preocuparse sobre si la pasarela de pagos está adaptada al dispositivo de cualquier posible comprador.

Teniendo en mente la conversión de un visitante esporádico de la aplicación a potencial comprador, Paypal permite realizar la compra del producto sin necesidad de que el comprador se registre en la web del vendedor. Posteriormente a la compra, los datos de envío del comprador que han sido introducidos en la cuenta de Paypal del comprador son transferidos al vendedor para que pueda enviar el producto.

APIs clásicas

Las APIs clásicas engloban todos los casos de uso que un desarrollador puede habilitar integrando Paypal en la aplicación. Estos casos de uso son:

[24] https://github.com/paypal

- Permitir pagos de productos o servicios no sólo al dueño de la aplicación si no a cualquier usuario de Paypal.
- Permitir que el comprador acepte un contrato entre comerciante y Paypal para poder comprar en la web haciendo sólo 1 click sin necesidad de realizar el proceso de compra con cada compra que se realice en la aplicación.
- Programar cobros recurrentes como la suscripción de un usuario a una revista cuyo pago es mensual.
- Establecer pagos para varios destinatarios a través de una sóla transferencia. Por ejemplo que se venda un pack de varios productos siendo los productos de diferentes proveedores.
- Devolver pagos si el servicio no ha podido ser proporcionado o el producto no cumple con las características esperadas.

Estas APIs clásicas cuentan con bastantes librerías para diferentes lenguajes de programación como Java, .NET o PHP. No obstante, estas librerías simplemente facilitaban el uso de las APIs ya que las APIs clásicas comprenden dos interfaces de conexión: Una interfaz que implementa el protocolo SOAP y otra que funciona a través de peticines HTTP recibiendo y respondiendo mensajes NVP (de pares Nombre-Valor) como *user=email@email.com*.

El uso de las APIs clásicas consiste en integrar la librería en el proyecto que interacutará con la API y, al igual que Express Checkout API, al realizar un pago devolverá una URL para redirigir al usuario y que confirme dicho pago. Como esta parte del pago se realiza en el sitio web de Paypal, la aplicación no tiene información de que hace el usuario. Para que la aplicación reciba información adicional, los servidores de Paypal podrían enviar mensajes a un servidor IPN (Instant Payment Notification).

Aunque lo de un servidor IPN pueda parecer complejo, no es más que un servidor web local con un recurso URL público que se indica a Paypal para que reciba ciertos datos a través de HTTP GET o POST. Esto podría ser un script PHP que procesase las varaibles POST definidas por Paypal.

API REST

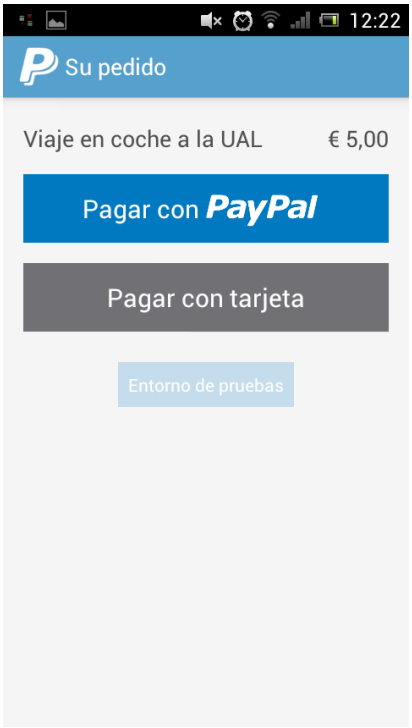
La API REST es una nueva interfaz de Paypal para ofrecer los mismos casos de uso que ofrecen las APIs clásicas a través de peticiones HTTP y encapsulando la información en objetos JSON. Las APIs REST son una interfaz muy liviana e intuitiva. Aprovechan la estructura del protocolo HTTP con sus verbos GET, POST, PUT y DELETE para realizar las operaciones típicas de intercambio de información entre aplicación y servicio.

SDKs móviles

Paypal ha tomado muy en consideración las plataformas móviles teniendo en cuenta que son muchas las aplicaciones para estas plataformas y que los dispositivos móviles acompañan continuamente al usuario. Por ello, en un afán de facilitar al máximo la integración de su pasarela de pago en aplicaciones de tereros, han desarrollado un SDK para iOS y otro SDK para Android.

En el caso del SDK para Android, que es el que se ha utilizado para la librería de pagos, este SDK no es más que una librería con un conjunto de clases Java para la conexión con la interfaz REST de Paypal y las actividades para la interfaz del usuario. El uso es muy sencillo: Consiste en integrar el SDK en el proyecto Android y en el punto en el que sea necesario, lanzar la actividad principal del SDK con la información de pago. A partir de ese punto, la actividad de Paypal solicita al usuario sus credenciales o si va a pagar con tarjeta la información bancaria y una vez autenticado en Paypal, permtie al usuario que confirme el pago y vuelva a la actividad de la aplicación.

Nótese que este SDK por el momento ofrece poco más que confirmar un pago. Si se desea funcionalidad más avanzada como los pagos recurrentes habría que integrar en la aplicación la API clásica.



Captura de pantalla del SDK de Paypal en Android

3.4.6. Referencia 5. La API de Coinbase

Coinbase es el servicio de pago elegido para integrar en la librería de pagos al menos un servicio de pagos que permitiese la transferencia de bitcoins. En internet se pueden encontrar bastantes servicios de pago que aceptan Bitcoin pero tras analizar los más conocidos, Coinbase destacó por ser el servicio con una API más simple, flexible y variada en funcionalidad.

Usando la interfaz de conexión de Coinbase, el desarrollador puede integrar los siguientes casos de uso en su aplicación:

- Comprar y vender bitcoins.
- Enviar y solicitar bitcoins por correo electrónico o a una dirección Bitcoin.
- Siendo un comerciante, aceptar bitcoins por productos o servicios.
- Almacenar bitcoins de forma segura
- Crear nuevas direcciones Bitcoin.
- Recoger información de la red Bitcoin.
- Manejar micro-transacciones y pagos por suscripción.

La API[25] que ofrece Coinbase es una interfaz de comunicación HTTP REST. Hoy en día, es difícil encontrar un lenguaje de programación que no disponga de una librería amigable para realizar peticiones HTTP, por lo tanto, desde cualquier lenguaje de programación, la comunicación con Coinbase es sencilla.

La API recibe y devuelve la información en objetos JSON y por cuestiones de seguridad, la conexión tiene que establecerse HTTPS y es necesario autenticar la aplicación obteniendo un token de acceso a través del protocolo OAuth 2.

A continuación, se detallan los principales puntos de conexión de la API que han sido utilizados para el desarrollo de la librería de pagos. Cada punto de conexión se detalla en una tabla. La cabecera de la tabla contiene el recurso URI y el verbo HTTP usado. Posteriormente se describe la funcionalidad del punto de conexión y se muestra un ejemplo del resultado y si procede un ejemplo del objeto JSON de la petición.

[25] https://coinbase.com/api/doc

GET https://coinbase.com/api/v1/account/balance

Devuelve el balance del usuario

<table>
<tr>
<td>

```
{
 "amount": "36.62800000",
 "currency": "BTC"
}
```

</td>
<td>Se puede utilizar el punto de conexión /api/v1/currencies/exchange_rates para calcular el balance en otra divisa</td>
</tr>
</table>

GET https://coinbase.com/api/v1/currencies/exchange_rates

Devuelve el valor de intercambio entre Bitcoin y otras divisas.

<table>
<tr>
<td>

```
{"btc_to_usd":"819.6788599999999",…}
```

</td>
<td>Devuelve todas las claves posibles btc_to_xxx y xxx_to_btc donde xxx es la codificación ISO de la divisa en minúsculas. En el ejemplo, 1 BTC equivale a algo más de $ 819,678859</td>
</tr>
</table>

GET https://coinbase.com/api/v1/transactions/:id

Devuelve los detalles de una transacción referenciada con :id.

<table>
<tr>
<td>

```
{
 "transaction": {
  "id": "5018f833f8182b129c00002f",
  "created_at": "2012-08-01T02:34:43-07:00",
  "amount": {
   "amount": "-1.10000000",
   "currency": "BTC"
  },
  "request": true,
  "status": "pending",
  "sender": {
   "id": "5011f33df8182b142400000e",
   "name": "User Two",
   "email": "user2@example.com"
  },
  "recipient": {
   "id": "5011f33df8182b142400000a",
   "name": "User One",
   "email": "user1@example.com"
  }
 }
}
```

</td>
<td>Se puede utilizar este punto de conexión enviando el identificado de la transacción como parámetro GET para verificar un pago a través del servicio Coinbase.

En este ejemplo, el recurso al que se accede es https://coinbase.com/api/v1/transactions/5018f833f8182b129c00002f.

En esta transacción, el estado es pending, esto es debido a que es una petición de dinero, por ello, el campo request es true. En un envío de dinero, tras la firma de varios bloques de transacciones, la transacción pasará al estado complete.

La sección de la documentación /api/v1/transactions contiene más información sobre el objeto JSON de la respuesta</td>
</tr>
</table>

POST https://coinbase.com/api/v1/transactions/send_money

Permite enviar dinero a una dirección de correo o dirección Bitcoin

<table>
<tr>
<td>

```
{
 "transaction": {
  "to": "user1@example.com",
  "amount": "1.234",
  "notes": "Sample transaction for you"
 }
}
```

</td>
<td>Objeto JSON que se envía en el cuerpo de la petición POST. En el ejemplo se envían 1,234 BTC.

Si se desea enviar en otra divisa, se incluye la divisa en un campo amount_currency_iso y se sustituye amount por amount_string</td>
</tr>
</table>

```
{
 "success": true,
 "transaction": {
  "id": "501a1791f8182b2071000087",
  "created_at": "2012-08-01T23:00:49-07:00",
  "hsh": "9d6a7d1112c3db9de5315b421a5153d71413f5f752aff75bf504b77df4e646a3",
  "notes": "Sample transaction for you!",
  "amount": {
   "amount": "-1.23400000",
   "currency": "BTC"
  },
  "request": false,
  "status": "pending",
  "sender": {
   "id": "5011f33df8182b142400000e",
   "name": "User Two",
   "email": "user2@example.com"
  },
  "recipient": {
   "id": "5011f33df8182b142400000a",
   "name": "User One",
   "email": "user1@example.com"
  },
  "recipient_address": "37muSN5ZrukVTvyVh3mT5Zc5ew9L9CBare"
 }
}
```

La respuesta de este punto de conexión adjunta un campo **success** indicando el éxito de la operación junto con información detallada de la transacción.

El campo **request** con el valor **false** indica que es una transacción de bitcoins y no sólo una petición realizada a una dirección Bitcoin

POST https://coinbase.com/api/v1/transactions/request_money	
Permite que un usuario solicite dinero de una dirección Bitcoin	

<pre>{
"transaction": {
"from": "user1@example.com",
"amount": "1.234",
"notes": "Sample transaction for you"
}
}</pre>
	Se aprecia que la única diferencia en el objeto JSON que se envía en el cuerpo de la petición con el del punto de conexión api/v1/transactions/send_money es el cambio del campo **to** por **from**

<pre>{
"success": true,
"transaction": {
"id": "501a3554f8182b2754000003",
"created_at": "2012-08-02T01:07:48-07:00",
"hsh": null,
"notes": "Sample request for you!",
"amount": {
"amount": "1.23400000",
"currency": "BTC"
},
"request": true,
"status": "pending",
"sender": {
"id": "5011f33df8182b142400000a",
"name": "User One",
"email": "user1@example.com"
},
"recipient": {
"id": "5011f33df8182b142400000e",
"name": "User Two",
"email": "user2@example.com"
}
}
}</pre>
	Como se comentó anteriormente, la respuesta de la petición de dinero establece el campo **request** a **true** y el campo **status** a **pending**

3.4.7. Uso de la sandbox de Paypal

Para un desarrollador, una de las características que más valora en un servicio de pago es la capacidad de probar el servicio en un entorno virtual. Sobre todo en el campo financiero. Poder poner a prueba la aplicación con los diferentes casos de uso de éste y simular transferencias permite depurar la aplicación y fortalecerla para evitar en la puesta en producción errores que conlleven usuarios insatisfechos y pérdidas económicas no sólo para los responsables de la aplicación si no también para usuarios.

Hay pocos servicios de pago que ofrezcan un entorno virtual donde realizar transferencias ficticias como la sandbox de Paypal y esta característica puede desequilibrar la balanza hacia el servicio de pago que la posea. Lamentablemente, Coinbase es uno de esos servicios que no ofrecen entorno de pruebas y por ello, en este proyecto no ha sido posible utilizar ciertos casos de uso de Coinbase desde la librería desarrollada.

Para usar el entorno de pruebas, es necesario en primer lugar especificar al usar la API que se desea utilizar dicho entorno. En el SDK de Android consiste en añadir a la instancia de tipo **Intent** que lanzará la actividad de Paypal la constante **PAYPAL_SANDBOX** en el extra **EXTRA_PAYPAL_ENVIRONMENT**.

El siguiente paso es crear cuentas virtuales para utilizarlas dentro del entorno de pruebas. Estas cuentas de Paypal virtuales se crean desde el sitio de desarrolladores de Paypal[26], en la pestaña **Applications** y la sección **Sandbox Accounts**.

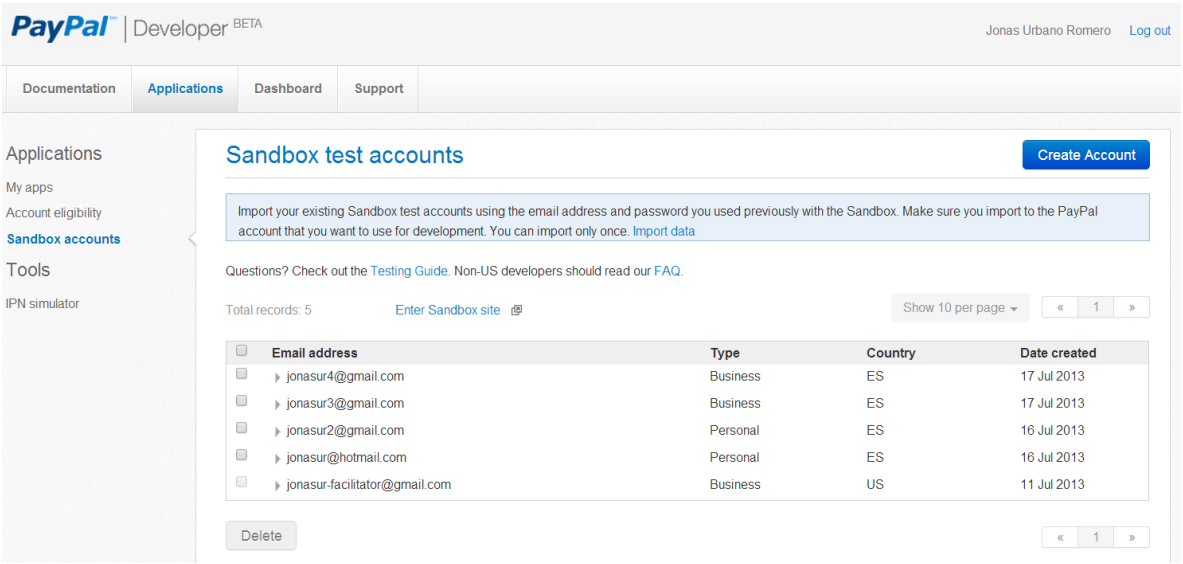


Ilustración 2Apartado Sandbox Accounts del entorno de pruebas de Paypal

[26] https://developer.paypal.com/

Al crear un usuario, se podrá configurar:

- El país donde el usuario usará la cuenta
- El balance de su cuenta
- El tipo de cuenta: Si es personal o comerciante
- Si el banco enlazado a la tarjeta ha sido verificado
- El tipo de tarjeta de crédito (Visa, Mastercard…)

Paypal ha desarrollado una interfaz web para el entorno de pruebas[27] desde la que se pueden crear las cuentas de Paypal virtuales. Una vez creadas, para usar el entorno de pruebas tan sólo es necesario autenticarse en la pasarela de pagos a través de la aplicación con los credenciales de estas cuentas virtuales. Cualquier transferencia que se haga con estas cuentas será ficticia. Por seguridad, Paypal no permite utilizar cuentas reales en entorno de pruebas ni cuentas virtuales con la pasarela en producción. Si se intenta la autenticación con una cuenta real de Paypal en el entorno de pruebas, la respuesta será que el nombre de usuario o contraseña son inválidos.

Lo más interesante del entorno de pruebas es comprobar las transacciones realizadas a través de las notificaciones que se realizan dentro de éste. En la sección **Sandbox notifications** de la pestaña **Dashboard** se pueden comprobar las notificaciones que recibiría un usuario a través de email o SMS si estuviera fuera del entorno de pruebas. Desde esta sección se puede verificar que las transacciones que se han realizado de prueba desde la aplicación desarrollada se han completado correctamente y con éxito.

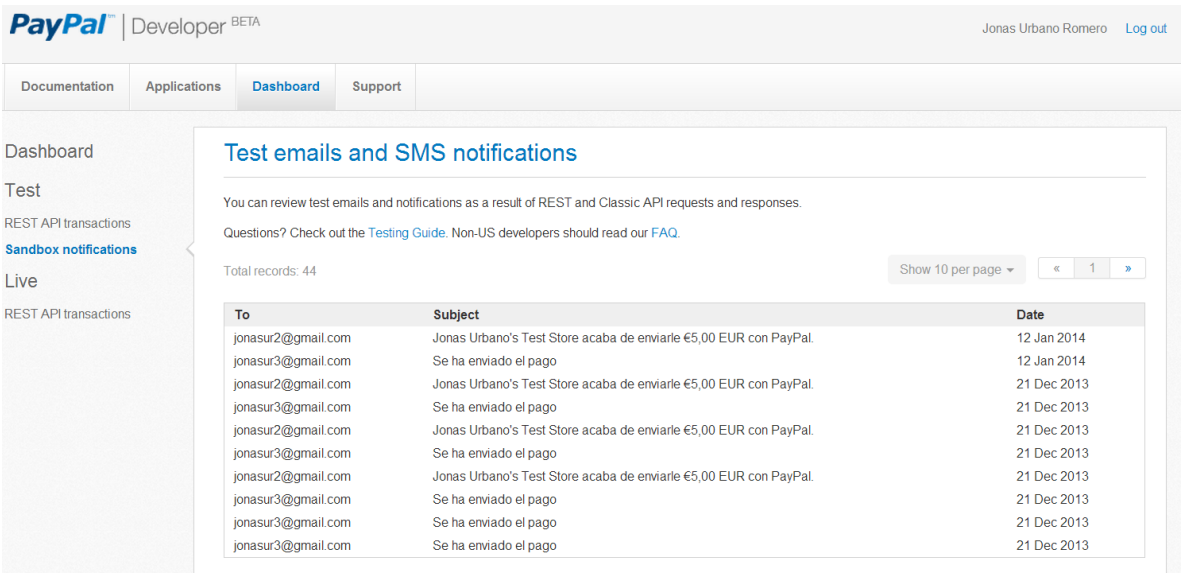


Ilustración 3Apartado Sandbox notifications del entorno de pruebas de Paypal

[27] http://sandbox.paypal.es

A la vez que se comprueban las notificaciones, desde la sección Sandbox Accounts comentada anteriormente se puede comprobar el balance de la cuenta ficticia y verificar si ha aumentado o disminuido este balance dependiendo de la transacción realizada. Nótese que no se refleja la comisión cobrada por Paypal a la persona que recibe el pago en el balance de la cuenta ni en las notificaciones.

Adicionalmente, si se dispone de un servidor que recibe notificaciones IPN de Paypal también puede ser probado a través de la sandbox de Paypal para comprobar que recibe y procesa correctamente notificaciones de Paypal.

3.5. Herramientas para el desarrollo

A continuación se describen las herramientas que he utilizado en el desarrollo de las librerías, explicando brevemente en qué consisten y cuáles son las características que más valoro y más me han ayudado.

3.5.1. Android Development Tools

Android Development Tools [28]es el plugin para el desarrollo de aplicaciones Android en Eclipse. No sólo tiene las características más comunes de un IDE potente como es el mismo Eclipse, si no que además, el editor gráfico de las vistas es muy útil pudiendo configurarlo con la resolución de muchos de dispositivos móviles en el mercado.

Sin embargo, una de las mejores características con las que cuenta el plugin ADT es la depuración de la aplicación en un dispositivo móvil. El entorno permite lanzar la aplicación el en el dispositivo móvil, leer los logs, analizar la secuencia de ejecución y parar la ejecución de la aplicación en puntos de ruptura para analizar el estado de las variables.

3.5.2. Spring Tool Suite

Spring Tool Suite[29] es la distribución de Eclipse para el desarrollo con la librería Spring. Entre otras muchas funciones permite el autocompletado en archivos de configuración XML y anotaciones. También posee vistas especializadas para analizar los beans que gestiona el contenedor de Spring y otros plugins de Eclipse instalados por defecto como m2eclipse para integrar el gestor de dependencias en el IDE y eGit para realizar control de versiones de los proyectos en un repositorio Git.

3.5.3. Bitbucket

Bitbutcket[30] es un servicio web de Atlassian para gestionar repositorios de código con Git. A diferencia de Github, permite crear repositorios privados de forma gratuita. El proyecto de este documento se ha desarrollado gestionando dos repositorios, uno para la librería del dispositivo móvil y otro para la librería de la aplicación del servidor.

[28] http://developer.android.com/sdk/index.html
[29] https://spring.io/tools/sts/all
[30] https://bitbucket.org/

3.5.4. Github

Github[31] es el servicio de gestión de control de versiones por excelencia. Sus capacidades sociales permiten descubrir proyectos muy interesantes. El código de muchas de las librerías usadas en este proyecto está alojado en Github y ya que los usuarios pueden informar sobre incidencias y hacer peticiones de integración, Github se ha convertido en la principal vía para estar informado sobre fallos y actualizaciones.

Revisar código de proyectos, ver cómo evoluciona el software y entender las correcciones sobre ese código es una buena práctica para mejorar como desarrollador.

3.5.5. SourceTree

SourceTree[32] es la herramienta para escritorio de Atlassian que permite gestionar repositorios de código Git y Mercurial. Cuenta con integración para repositorios alojados en Github y en Bitbucket. El control de versiones con esta herramienta es muy intuitivo. Esta imagen es una captura de pantalla del control de versiones de la aplicación Github para Android. El grafo facilita la comprensión de la evolución del proyecto y las aportaciones de los desarrolladores en Github.

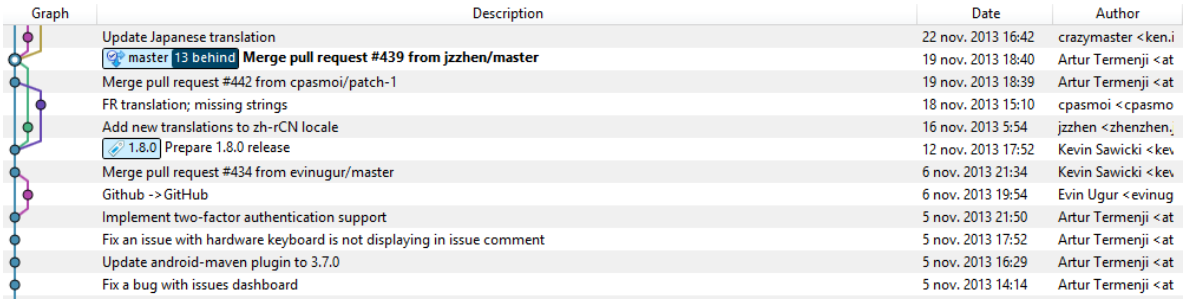


Ilustración 4Grafo del repositorio de la aplicación de Github para Android

Un ejemplo de cómo Github y SourceTree mejoran la productividad del desarrollador lo encontré al usar la librería Adaptive Payments de las APIs clásicas de Paypal. La versión de la librería contenía un error que lanzaba una excepción NoSuchAlgorithmException. Encontré que un usuario de Github había informado de la incidencia. Posteriormente se realizó un commit solucionando el error y al revisar el grafo en SourceTree comprobé que se liberó el tag 1.5.0 con el que este error desapareció.

[31] https://github.com/
[32] http://www.sourcetreeapp.com/

3.5.6. ScrumDo

ScrumDo[33] es un servicio web para gestionar proyectos ágiles Scrum. La aplicación es gratuita aunque cómo máximo se puede administrar un proyecto con tres usuarios. Desde ScrumDo se pueden crear iteraciones, añadir historias de usuario, marcar el estado de las historias de usuario conforme el proyecto evoluciona y revisar las gráficas de la evolución de las iteraciones que se conocen como BurnUp y BurnDown.

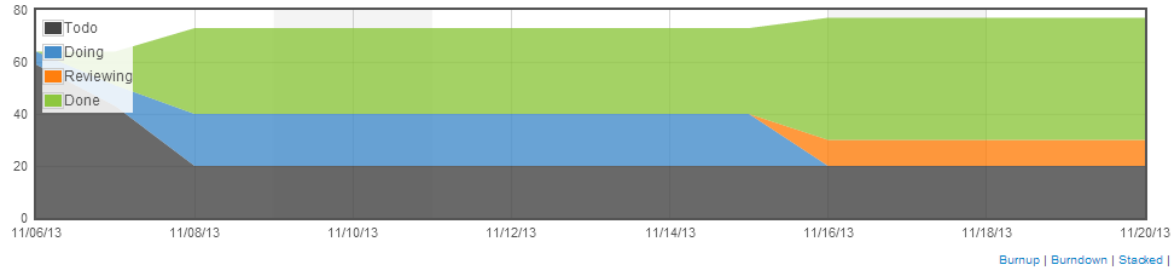


Gráfico de evolución del sprint en ScrumDo

Este es el gráfico de evolución de los estados de las tareas de usuario en la iteración del proyecto desde el 6 hasta el 20 de noviembre de 2013. Se aprecia cómo las tareas pasaron del estado To do –en negro- a Done –en verde- pasando algunas de ellas por revisión –en naranja-.

Una característica muy interesante de ScrumDo es que dispone de API para integrar la herramienta en desarrollos o incluso para integrarla en la metodología de desarrollo de software personal implantándola en un IDE.

3.5.7. Evernote

Evernote[34] es una aplicación para almacenar notas. Tiene un potente buscador para buscar texto en todas las notas e incluso en imágenes contenidas en esas notas. Las notas se crean dentro de libretas y estas libretas a la vez se pueden agrupar en pilas de libretas. De forma transversal, se pueden catalogar las notas con diversas etiquetas para relacionar notas por temas entre diferentes libretas.

Es muy fácil compartir notas a través de correo electrónico o facilitando el enlace de la nota en la aplicación web de Evernote. Internamente, se pueden crear enlaces o atrás notas como si de hipervínculos se tratase, tejiendo así una red de información entre diversas notas.

[33] http://www.scrumdo.com/
[34] https://evernote.com/intl/es/

Para gestionar el proyecto que abarca este documento he utilizado Evernote creando una pila de libretas llamada PFC que contiene 5 libretas con la siguiente finalidad:

- **Diario**: Libreta que contiene una nota por cada día que he trabajado en el proyecto en la que describo qué voy a hacer, que problemas encuentro, que soluciones tomo y el tiempo que dedico.
- **Gestión**: Contiene notas con información variada como las credenciales de las cuentas de Paypal, las claves de desarrollador para las APIs o listas de tareas.
- **Documentación Geolocalización**: Contiene notas subrayadas de artículos sobre geolocalización leídos en la web.
- **Documentación Pagos**: Contiene notas subrayadas de artículos sobre servicios de pago leídos en la web.
- **Documento**: Contiene notas con apuntes y borradores para los apartados de este texto.

El equipo de Evernote ha desarrollado dos plugins para navegadores imprescindibles: **Evernote Web Clipper** y **Clearly**. El primero permite exportar artículos directamente desde el navegador a una libreta de Evernote. El segundo va más allá y aísla el artículo en un fondo claro haciendo desparecer el resto de secciones de la web para aumentar la concentración y proporcionar una lectura más cómoda. A esto se le une la posibilidad de subrayar el texto que se está leyendo y, automáticamente, el artículo se guarda en una libreta de Evernote con el texto subrayado. Todos los artículos guardados en las libretas Documentación Geolocalización y Documentación Pagos han sido capturados de esta forma.

3.6. Metodología de desarrollo

Durante el desarrollo del proyecto he seguido las recomendaciones de la metodología de desarrollo de software ágil Scrum. He creado historias de usuario con las tareas a realizar y las he incluido en iteraciones ordenadas en orden decreciente a su importancia.

He procurado que la duración de las iteraciones oscilase entre dos semanas y un mes, esforzándome por tener un producto potencialmente entregable. Cabe destacar que para un desarrollador habituado a ciertas historias de usuario y a cierta tecnología, cumplir la recomendación de software potencialmente entregable no es difícil. En cambio, cuando las historias de usuario suponen enfrentarse a una nueva tecnología o un nuevo marco de trabajo que no es el habitual como ha sido la integración de las APIs de Paypal, cumplir con los plazos es bastante más complicado.

Al marcar una historia de usuario como Doing, realizaba un pequeño análisis y diseño de la solución esbozando los diagramas de clases y de secuencia que se plasman en este documento. Tras la implementación, el proceso de prueba era variado: Para algunas historias de usuario había desarrollado previamente pruebas unitarias mientras que para otras las pruebas consistían en probar el funcionamiento de la aplicación desde el dispositivo móvil.

El control de versiones ha sido gestionado con SourceTree en un repositorio en Bitbucket. He procurado actualizar el repositorio al acabar cada historia de usuario manteniendo pequeños commits que sean fácilmente entendibles e individuales para favorecer el momento de la integración con otras ramas. Cuando una historia de usuario ha sido especialmente complicada o larga he preferido crear una rama para mantener el desarrollo principal estable.

4. Conclusiones y trabajos futuros

A lo largo de este proyecto se han cumplido de manera exitosa los objetivos propuestos en la planificación del trabajo. Además se ha garantizado el aprendizaje y la adquisición de conocimientos en las áreas tratadas en este documento. Los objetivos cumplidos son los siguientes:

- **He aprendido a desarrollar aplicaciones móviles sobre la plataforma Android** que conlleva sobre problemas de amplio calado en aplicaciones móviles como son la gestión de los recursos del dispositivo y los problemas continuos de conectividad que experimenta una aplicación Android con backend. He hecho uso de una información tan valiosa y delicada como es la posición geográfica de un usuario y me he documentado sobre cómo aplicaciones referentes en este campo la explotan enriqueciendo su contenido.
- **He desarrollado dos librerías** para facilitar la obtención y gestión de la posición geográfica de un dispositivo móvil e integrar de manera sencilla diferentes servicios de pago en una aplicación Android. Estas dos librerías se han implantado sobre un paquete de gestión de servicios para enriquecer la experiencia de usuario teniendo en cuenta la proximidad de éstos y permitiendo que se puedan comprar en la plataforma.
- **He realizado un estudio sobre los diferentes servicios de geolocalización y servicios de pago** que a día de hoy se encuentran disponible entendiendo la problemática de gestionar bienes monetarios mientras se busca ofrecer la mejor experiencia de usuario al comprar.

Si una aplicación Android cuyo servicio esté basado en posicionamiento tiene éxito, es fácil que experimente un crecimiento exponencial del número de usuarios hasta llegar a un punto en el que el rendimiento del servicio se vea afectado debido a la enorme cantidad de cálculos y procesamiento que tanta información geográfica puede suponer. En esta tesitura los **trabajos futuros** se centrarían en el **estudio de sistemas distribuidos de procesamiento de información geográfica donde se aprovechase al máximo el conocimiento de este tipo de información** para balancear la carga y cumplir con un rendimiento adecuado.

En lo referente a la librería de pagos para dispositivos móviles con Android, no se puede mirar a otro lado con la aparición de nuevos dispositivos como son los televisores con conexión a internet denominados Smart TVs. En estos dispositivos hay un nuevo mercado que es el alquiler y compra de contenido digital como el alquiler de películas y la compra de juegos. Otros dispositivos muy interesantes son las Google Glasses que podrían cambiar definitivamente la forma de compra habitual. Sería un reto **adaptar la librería a dichos dispositivos para crear una experiencia de compra excelente** en aplicaciones Android sin olvidar proporcionar nuevas implementaciones con diferentes servicios de pago.

5. Referencias y bibliografía

M-Commerce
Bravo Torres, Jack (2011)
Extracto de conferencia

Estudio del comercio electrónico 2012
Online Business School (2013)
Informe técnico

Responsive Web Design
Marcotte, Ethan (2010)
Libro

Estudio sobre comercio electrónico B2C 2012. Edición 2013
Observatorio nacional de las telecomunicaciones y de la SI. (2013)
Informe técnico

Diffusion of Mobile Commerce Application in the Market
Huang H. et al. (2007)
Extracto de conferencia

Sistemas de pagos electrónicos
Altamirano Di Luca, Marlon (2012)
Artículo web

Location Based Services in Android
Ch. Radhika Rani, A. Praveen Kumar, D. Adarsh, K. Krishna Mohan, K.V.Kiran (2011)
Artículo técnico

Bitcoin: A Peer-to-Peer Electronic Cash System
Satoshi Nakamoto (2008)
Artículo técnico

Bitcoin: La moneda del futuro
Bitcoin en español (2012)
Libro

GeoFire: Location Queries for fun and profit
Kavya Joshi (2013)
Artículo web

A deep dive into location
Reto Meier (2013)
Artículo web

¿Es seguro Bitcoin? Su tecnología al desnudo
Guillermo Julián Moreno (2013)
Artículo web

Apple, iBeacon, y la internet de las cosas
Javier Pastor (2013)
Artículo web

PayPal Has A New Retail Trick Up Its Sleeve
Dylan Love (2013)
Artículo web

La revolución con el pago móvil está sucediendo… Al cobrar
Jaime Novoa (2013)
Artículo web

Can PayPal Crack the Design Challenge of Digital Wallets?
Kyle Vanhemert (2013)
Artículo web

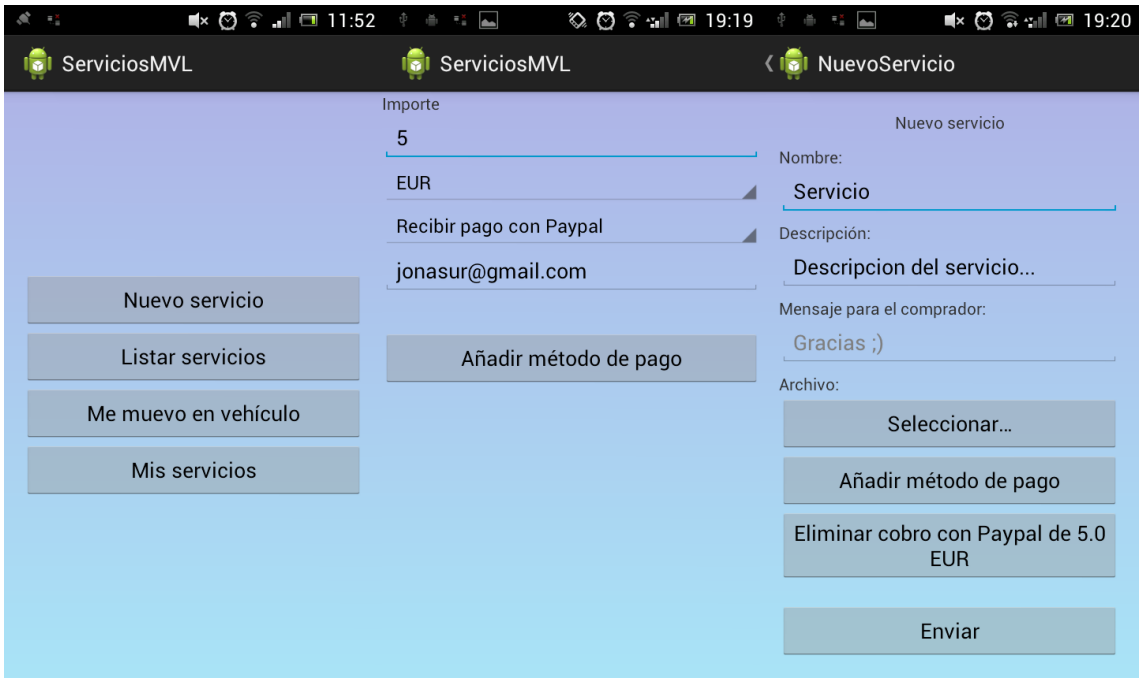
El gran libro de Android
Girones, Jesús Tomás (2011)
Libro

Android 4: Desarrollo de aplicaciones
Wei Meng Lee (2012)
Libro

Descubriendo Bitcoin: un viaje real al interior de la moneda virtual
Ordovás, Javier (2013)
Artículo web

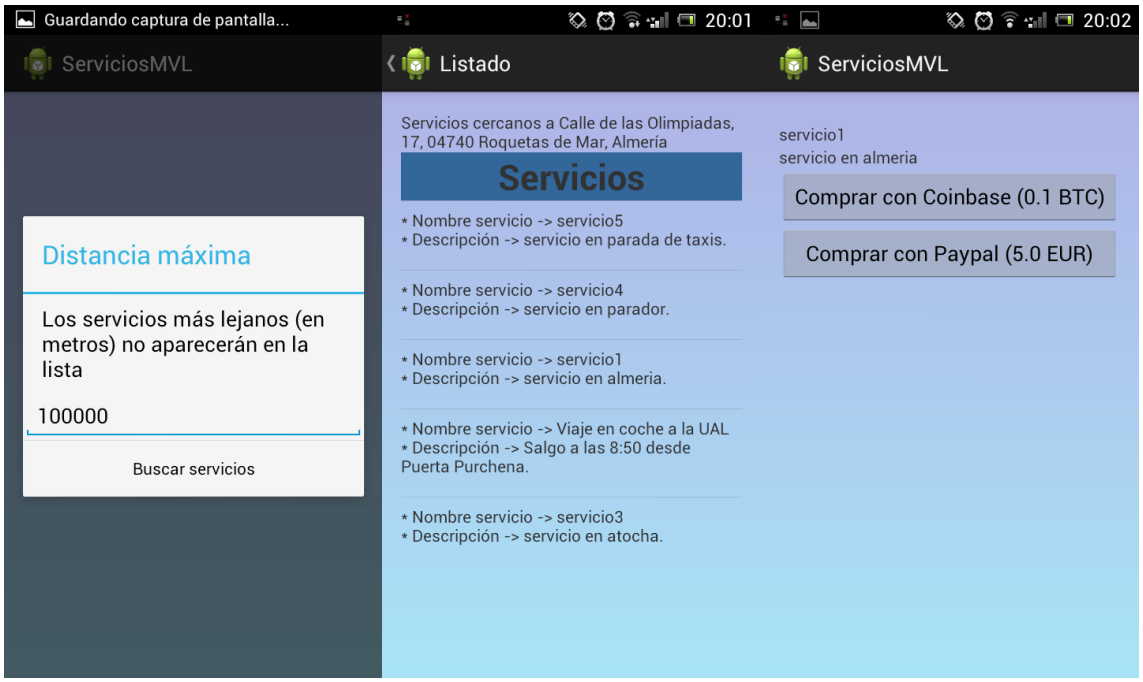
6. Anexos

6.1. Capturas de las actividades de la aplicación móvil



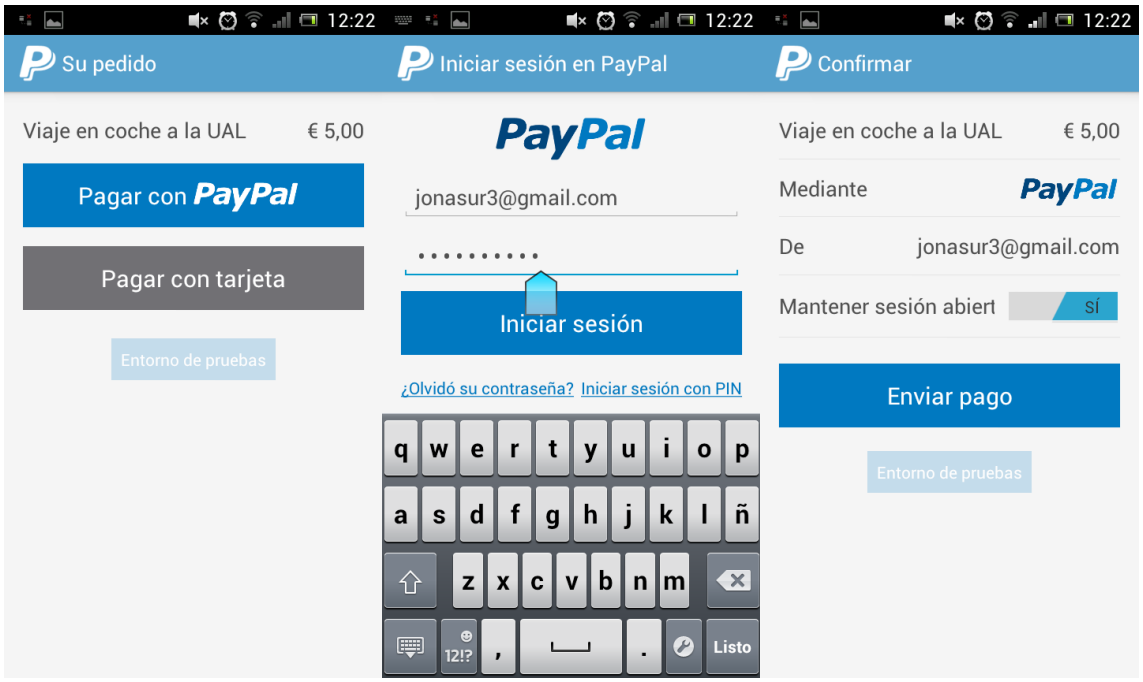
La actividad principal que alberga el menú de la aplicación cuenta con un botón para que el usuario especifique su movimiento, es decir si anda o si se mueve en vehículo. En la primer fotografía, el botón tiene el texto “Me muevo en vehículo”, si el usuario lo pulsa, la librería se configurará según el movimiento del usuario.

La segunda captura muestra la actividad para añadir un método de pago. Se especifica el importe, la cantidad, el servicio de pago y la dirección del vendedor. En la tercera imagen, se muestra el proceso de creación de un servicio con el botón “Eliminar cobro con Paypal de 5,0 EUR” que indica que ya se ha añadido un método de pago al servicio.

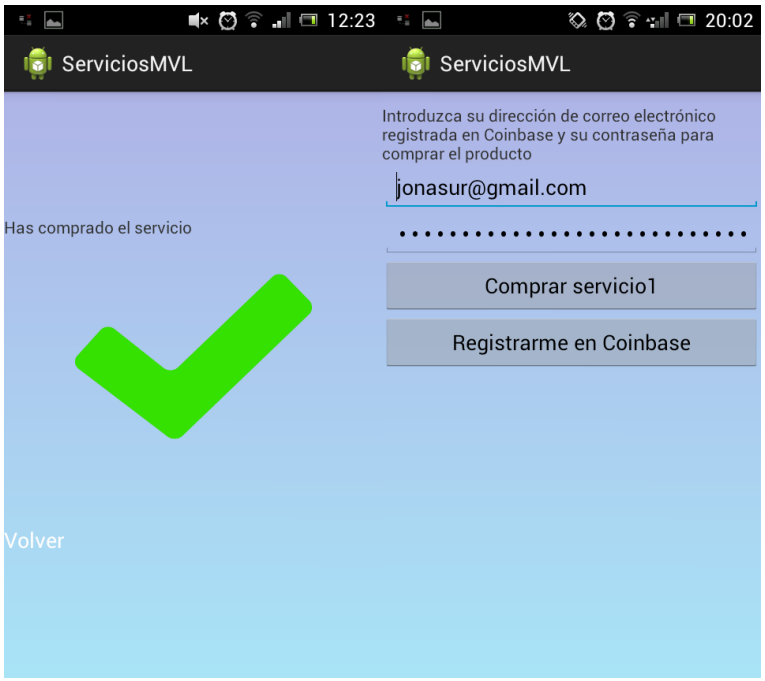


En esta captura aparece un cuadro de diálogo para indicar la distancia en metros a la que se quiere buscar servicios. La siguiente captura muestra un listado con los servicios cercanos a la posición del usuario que es la que se describe sobre la cabecera.

La tercera captura muestra el detalle de un servicio y el botón para comprar el servicio a través de la pasarela de pagos de Paypal.



Tras elegir realizar el pago con Paypal, en la primera imagen se elige pagar con la cuenta de Paypal. En la segunda se introduce el nombre de usuario y contraseña y en la tercera se confirma el pago.



La primera captura muestra el resultado exitoso tras pagar con la pasarela de pagos Paypal. La segunda captura es la actividad donde se introduce el nombre de usuario y contraseña en el servicio de pago Coinbase para pagar un servicio.