CAMPUS
DE EXCELENCIA
INTERNACIONAL

Graduado en Ingeniería Informática

Universidad Politécnica de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

TRABAJO FIN DE GRADO

Desarrollo de una solución nativa y multiplataforma para la interacción desde una plataforma móvil con una aplicación bancaria

Autor: Carlos Sánchez Marín

Director: Fernando Pérez Costoya

MADRID, ENERO DE 2019

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE EJEMPLOS DE CÓDIGO

Resumen

Aquí el texto del abstract.
Palabras clave: palabra 1, palabra 2, palabra 3...

Abstract

Here goes the abstract text.
Keywords: keyword 1, keyword 2, keyword 3...

1. INTRODUCCIÓN

En los últimos años el sector bancario ha visto como los pagos con tarjetas ha crecido exponencialmente. Según los datos ofrecidos por el Banco de España, desde 2002 hasta 2017 el número de operaciones creció un 245 % y el importe de las operaciones un 188 %, llegando a un gasto total de 135.246,47 millones de € en el 2017[1].

Este rápido incremento de las operaciones con tarjeta también se ha trasladado al entorno corporativo, generando nuevas necesidades para el control por parte de las empresas de las tarjetas que dan a sus empleados. Con el objetivo de cubrir esta necesidad, tanto Mastercard [1] como Visa [2] han implementado nuevas funcionalidades de control de tarjetas que permiten el control directo por parte de las empresas de sus tarjetas, de manera que se puedan recibir alertas de gasto o bloquear determinados tipos de pago.

Este servicio es ofrecido a través de diversas API REST y en Vipera hemos implementado una aplicación que abstrae a los clientes finales (bancos y empresas) de las API proporcionadas por Visa y Mastercard, ofreciendo una API REST centralizada que simplifica las operaciones. Esta API interactúa con un servicio propietario de Vipera que integra toda la funcionalidad requerida por los distintos clientes finales.

Los SDK desarrollados para la integración de nuestro servicio en las aplicaciones móviles de los clientes pretenden abstraer totalmente de las llamadas al servicio de manera que los desarrolladores tan solo vean funciones totalmente descriptivas sin tener que preocuparse de la configuración necesaria para la interacción. Así mismo, ya que la demo técnica realizada por Vipera está implementada con Ionic[3], se ha implementado un plugin de Cordova[4] que habilita la llamada a los SDK de iOS y Android.

2. ESTADO DEL ARTE

2.1. Tecnologías

En el mercado móvil actual iOS y Android lideran con mano de hierro dejando una cuota residual a otros entornos como Windows Phone y otros sistemas menores. En conjunto, iOS y Android acapararon el 99,7 % de las ventas en el primer trimestre de 2018 en España, dejando a Windows fuera de las estadísticas al no lograr el 0,1 % de las ventas totales[5]. Teniendo presentes estos datos, resulta obvio que cualquier tipo de desarrollo fuera de Android e iOS tendría un impacto muy reducido y difícilmente sería rentable para cualquier aplicación que no tenga como objetivo a todos los usuarios del mercado móvil.

[1] `https://developer.mastercard.com/product/spend-controls`
[2] `https://developer.visa.com/capabilities/vctc`
[3] `https://ionicframework.com/`
[4] `https://cordova.apache.org/`
[5] `https://bit.ly/2Cu8NVy`

2.1.1. iOS

El sistema operativo iOS es un sistema propietario desarrollado por Apple[6] para sus dispositivos móviles iPhone, iPad y iPod Touch. Fue presentado junto con el primer iPhone en junio de 2007. En un principio Apple no iba a proporcionar SDK para desarrollo de aplicaciones nativas por parte de terceros, pero ante la reacción negativa de todos los desarrolladores, Apple reconsideró su decisión y terminó lanzando la primera versión del SDK en Marzo de 2008 junto a la segunda versión de iOS.

El SDK originalmente proporcionaba un acceso muy limitado a las funciones del sistema operativo, además de tan solo proporcionar soporte para el lenguaje Objective-C[7]. Año tras año Apple ha actualizado su sistema operativo móvil junto con el SDK aprovechando el lanzamiento de los nuevos iPhone.

En 2014, Apple lanzó Swift[8], un nuevo lenguaje más moderno que sustituyó a Objective-C como lenguaje de referencia para el desarrollo de aplicaciones iOS. No obstante, ambos lenguajes son actualmente soportados por el SDK de Apple, aunque la documentación tan solo es actualizada para Swift.

2.1.2. Android

Android[9] fue creado en 2003 como un fork del kernel de Linux para dispositivos móviles. Inicialmente fue concebido como un sistema operativo móvil inteligente que estuviese pendiente de la localización del usuario y sus preferencias. En 2005 la empresa propietaria del sistema operativo fue comprada por Google, que se hizo cargo del desarrollo. Dos años más tarde, Google hizo público el primer SDK en fase beta para Android, que fue distribuido a desarrolladores y fabricantes de dispositivos móviles. No fue hasta septiembre de 2008 que se lanzó al mercado el primer dispositivo con Android.

Las primeras versiones del SDK proporcionado por Google contaban tan solo con soporte para el lenguaje Java[10]. Al contrario de su principal competidor, el código fuente de Android siempre ha sido open source[11] lo que ha atraído a una gran cantidad de desarrolladores y fabricantes. De todos modos, la práctica totalidad de los fabricantes integran en sus terminales la versión de Android propietaria de Google, que incluye integración con las apps de Google bajo la capa Google Play Services[12].

---

[6] `https://apple.com/es/`

[7] `https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html`

[8] `https://swift.org/`

[9] `https://www.android.com`

[10] `https://www.java.com`

[11] `https://source.android.com/`

[12] `https://developers.google.com/android/guides/overview`

En el evento Google I/O de 2017, Google anunció de manera oficial que se añadía un nuevo lenguaje soportado al SDK de Android [13], Kotlin[14]. Este lenguaje desarrollado por la empresa Jetbrains[15] funciona sobre la Java Virtual Machine, por lo que es totalmente compatible con el código Java y puede coexistir con este en una misma aplicación.

2.1.3. Cordova

En el año 2011, Adobe Systems[16] compró la empresa Nitobi y renombró su producto como Phonegap[17] . Así mismo, lanzó una versión open source del producto nombrada Cordova que es mantenida por la Apache Software Foundation[18]. Este framework permite el desarrollo de aplicaciones híbridas basadas en la web, embebiendo una aplicación web dentro de una aplicación nativa para el sistema operativo deseado. Esta solución se acerca a la idea que tenía Apple de aplicaciones móviles en un principio, pero va más allá otorgando un sistema de plugins nativos que permiten la interacción con el sistema operativo.

Los plugins proporcionan una implementación en código nativo (Objective-C para iOS y Java para Android) que será invocada por un wrapper escrito en Javascript[19]. En tiempo de ejecución Cordova es capaz de saber sobre qué plataforma está ejecutándose e invocar al código nativo de esta, con lo que abstrae a los desarrolladores de aplicaciones de esta gestión y facilita la escritura de código.

2.2. MOTIF

El despliegue del lado del servidor de la aplicación bancaria con la que se ha operado para la realización de este trabajo está realizado sobre una plataforma propietaria propiedad de Vipera[20] cuyo nombre comercial es MOTIF[21].

MOTIF es un servicio desarrollado en Java haciendo uso del framework OSGi[22], creado en 1999 como una primera aproximación al mundo de los microservicios en Java. En concreto, el desarrollo de MOTIF ha sido llevado a cabo empleando Apache Felix[23] como implementación del framework OSGi. El uso de este framework facilita la integración con los clientes finales, por lo general banco y otros servicios financieros, ya que estos suelen emplear Java en sus propios servicios y la arquitectura OSGi permite una integración sencilla de diferentes módulos a los que se llama *plugins* con el resto del sistema. En la figura 1 se representan los servicios básicos implementados en el core de MOTIF.

[13] https://blog.jetbrains.com/kotlin/2017/05/kotlin-on-android-now-official/
[14] https://kotlinlang.org/
[15] https://www.jetbrains.com/
[16] https://www.adobe.com/
[17] https://phonegap.com/
[18] https://www.apache.org/
[19] https://developer.mozilla.org/bm/docs/Web/JavaScript
[20] http://www.vipera.com/
[21] http://www.vipera.com/motif
[22] https://www.osgi.org/
[23] https://felix.apache.org/

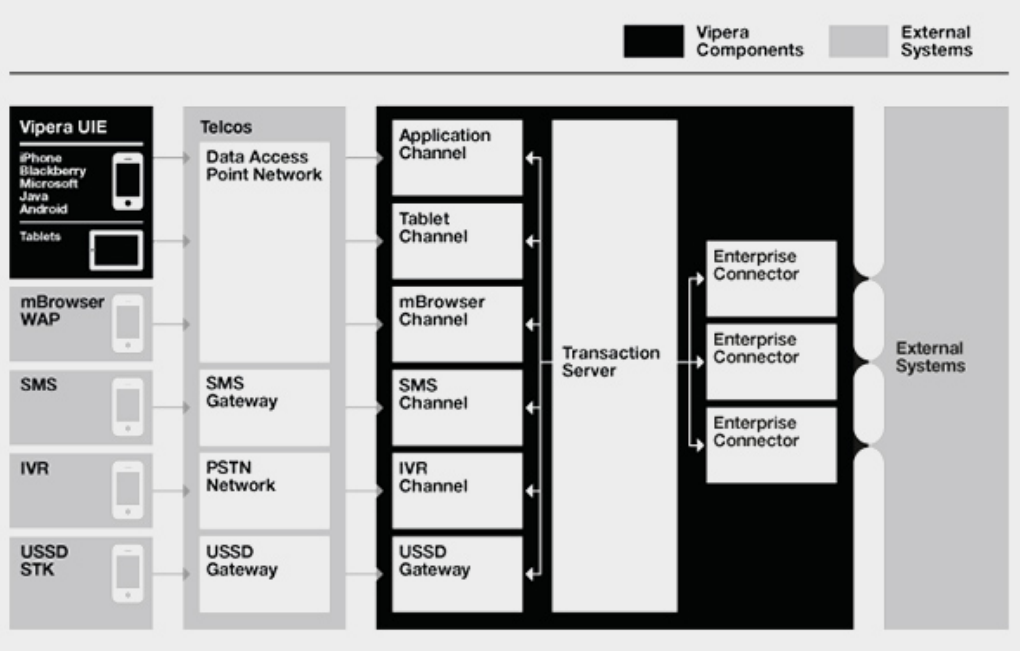

Figura 1: Diagrama de módulos predefinidos de MOTIF

En el caso concreto de la aplicación para la que se ha implentado la solución desarrollada como trabajo fin de grado, se ha hecho uso de los plugins que conforman el core de MOTIF, además de distintos plugins que permiten la interacción con los servicios de Mastercard y VISA, así como plugins de persistencia de datos empleando la librería Hibernate[24] y otros que implementan toda la lógica de negocio, que será explicada más adelante.

Para realizar la conexión con MOTIF, por defecto existen 2 métodos distintos:

- **API REST**: Está pensada para dar servicio a implementaciones de administración del sistema como podría ser un Help Desk.
- **JSON Protocol**: Este método de conexión es el que se emplea en las soluciones móviles. Se basa en peticiones POST a un endpoint estático, el cual procesa la información enviada en la petición y determina el comportamiento que debe seguir el servidor.

2.3. Modos de conexión con MOTIF desde una aplicación móvil

2.3.1. Peticiones POST

La forma más básica de hacer llamadas al endpoint JSON de MOTIF es la construcción de peticiones POST haciendo uso de aplicaciones como Postman[25] o empleando

[24] `https://hibernate.org/`
[25] `https://www.getpostman.com/`

directamente las librerías HTTP de los distintos lenguajes. El siguiente JSON es un ejemplo del cuerpo de una llamada a MOTIF.

```
{
  "req" : {
        "dom" : "{{domain}}",
        "app" : "{{application}}",
        "srv" : "{{service}}",
        "op"  : "{{operation}}",
        "header" :
        {
            "user":"user1@company1.com",
            "otp":"123456",
            "private:pwd":"1111",
            "private:vpwd":"1111",
            "email": "user1@company1.com"
        }
    }
}
```

Fragmento de código 1: Cuerpo de una petición POST a MOTIF

Como se puede apreciar, todos los campos de la llamada están encapsulados dentro del parámetro *req*, el cual es interceptado por el endpoint JSON de MOTIF. Una vez se ha interceptado la llamada, MOTIF comprueba los campos *dom* (dominio al que se está llamando), *app* (aplicación del dominio), *srv* (servicio de la aplicación) y *op* (operación del servicio).

Por último, en el campo *header* se encontrarán contenidos los parametros requeridos por la operación. En este caso se trata de una llamada de ejemplo para la activación de usuarios, por lo que se pasan como parámetros el usuario, OTP (One Time Password), contraseña y su verificación y el email del usuario. Cabe destacar que los campos ligados a la contraseña tienen el prefijo *private*, el cual al ser detectado por MOTIF le indica que no debe quedar constancia en los logs de esos campos.

2.3.2. Motif Connector

Uno de los equipos de desarrollo de Italia implementó una librería para abstraer de ciertos detalles a los desarrolladores de las aplicaciones móviles que emplean como backend MOTIF. Esta librería fue implementada tanto en Objective-C para iOS como en Java para Android. También se implementó un plugin de Cordova para facilitar su uso en aplicaciones híbridas.

En el siguiente código podemos observar como se compone la misma llamada a

MOTIF expuesta en el apartado anterior empleando el plugin para Cordova. Podemos observar el campo *header* sigue existiendo y siendo definido como un JSON, pero el usuario queda abstraido del formato del JSON que configura el dominio, aplicación, servicio y operación.

Por último, se puede observar en la composición de la llamada al servidor que hay un campo booleano marcado como *false*. Este campo determina si la llamada es segura, lo que implicaría la necesidad de que en la llamada incluya un ID de sesión para el usuario que la hace. El plugin implementa una función que guarda en la instancia este ID una vez el usuario se autentica con el servido. En este caso la llamada es de activación de un usuario, por lo que no se puede tener un ID de sesión.

```
let myHeader = {
        "user":"user1@company1.com",
        "otp":"123456",
        "private:pwd":"1111",
        "private:vpwd":"1111",
        "email": "user1@company1.com"
      };

      let request: MotifRequest =
       ↪  this.motifClient.buildServerRequest("register",
       ↪  "activateUser", myHeader, false);

      this.motifClient.sendRequest(request).then((res:
       ↪  MotifResponse) => {
        /*Codigo a ejecutar en caso de exito*/
      },(err) => {
        /*Codigo a ejecutar en caso de error*/
      });
```

Fragmento de código 2: Ejemplo de una llamada empleando el plugin Motif Connector

Esta libreria es la que se emplea como base para todas las soluciones personalizadas ofrecidas por Vipera, incluyendo la solución expuesta en este documento.

2.4. Card Control

En verano de 2017 se implementó un prototipo para Deutsche Bank en colaboración con Mastercard, que implementaba una solución parecida a la desarrollada para SME Pay, pero sin ofrecer la opción del control de tarjetas empresariales. Esta solución fue desplegada con éxito y ha sido usada como base de la lógica de negocio del proyecto SME Pay.

3. SME PAY

La solución implementada forma parte del proyecto SME Pay[26], un proyecto iniciado en colaboración con Mastercard para el control de gastos efectuado con tarjetas de empresa. En la actualidad el producto también ofrece la misma funcionalidad para tarjetas de VISA, siendo la gestión totalmente indiferente de cara al usuario, ya que las diferencias entre los distintos entornos son manejadas internamente por MOTIF.

3.1. Estructura de la aplicación

En la actualidad SME Pay ofrece 2 areas diferenciadas claramente. Por un lado se proporciona la posibilidad de controlar tarjetas personales, que podrán ser añadidas y eliminadas de manera arbitraria por cada usuario y a las que ningún otro usuario que no sea el propietario podrá tener acceso. Por otro lado, los gestores de las empresas que adquieran el producto tendrán acceso desde su aplicación a un apartado diferenciado que permitirá controlar las tarjetas de la empresa. Para efectuar este control se han implementado distintos servicios que interactuan entre ellos tal y como se puede observar en la figura 2.

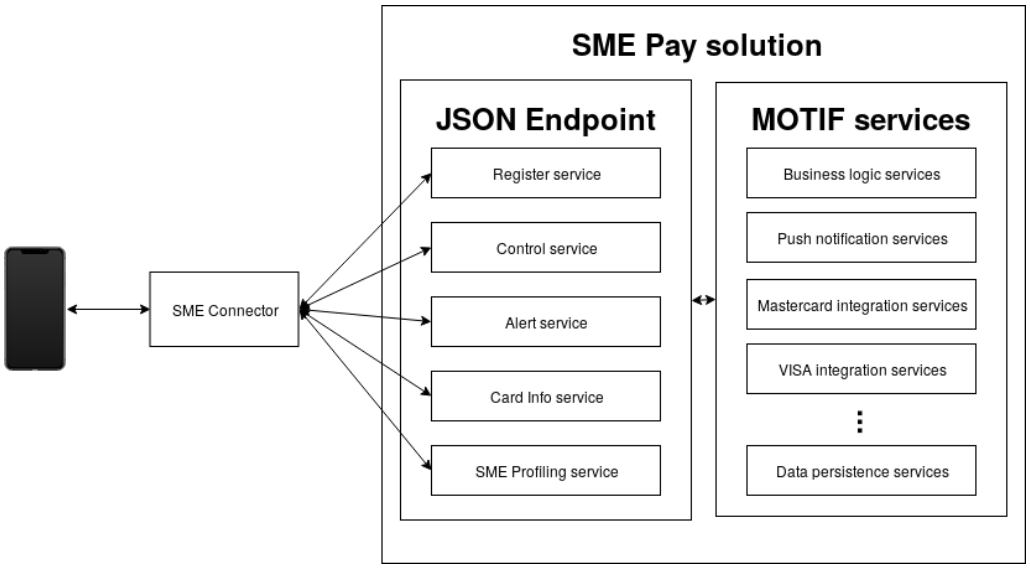

Figura 2: Servicios proporcionados por el JSON Endpoint de SME Pay

3.1.1. Servicio *register*

El servicio register ofrece funcionalidades de activación, login y logout. Para efectuar un login se puede hacer tanto empleando el PIN elegido por el usuario como su huella dactilar en caso de que el dispositivo móvil permita, así como el uso de la tecnología de reconocimiento facial en los modelos de Apple que así lo permitan.

[26] `http://www.vipera.com/sme-pay-2/`

3.1.2. Servicio *alert*

El servicio alert gestiona la configuración de notificaciones de las tarjetas personales de cada usuario. Tal y como se puede observar en la figura 3 en este punto debemos diferenciar entre notificaciones de los servicios ofrecidos por Mastercard y VISA y las notificaciones push que manda SME Pay a los terminales móviles de los usuarios.

Por un lado SME Pay configura todas las tarjetas registradas con Mastercard y VISA para recibir notificaciones de cualquier operación realizada. Sin embargo, los usuarios tienen la capacidad de configurar a través de este servicio que notificaciones push quieren recibir (retirada de dinero de un cajero, pagos online, operaciones por encima de un límite marcado...). De este modo el usuario tiene la libertad de gestionar las notificaciones que recibe en su dispositivo y al mismo tiempo el banco o empresa que contrata el servicio SME Pay tiene un registro accesible de todas las tarjetas registradas.

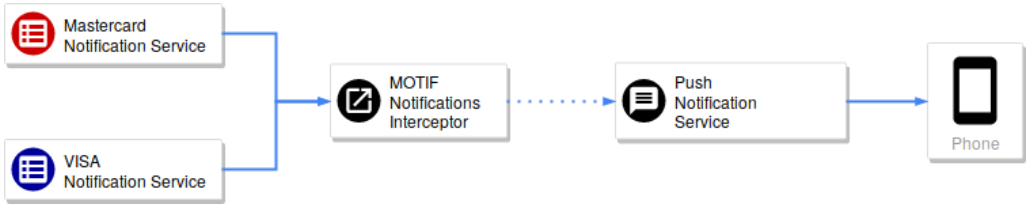

Figura 3: Diagrama de funcionamiento de notificaciones en SME Pay

3.1.3. Servicio *control*

El servicio control gestiona los bloqueos que se pueden establecer sobre las tarjetas personales de cada usuario.

3.1.4. Servicio *cardinfo*

El servicio cardinfo ofrece funcionalidades de registro, activacion, desactivación y eliminación de tarjetas, así como también facilita las tarjetas registradas por el usuario y sus transacciones asociadas almacenadas en los servidores de SME Pay.

3.1.5. Servicio *smeprofiling*

El servicio smeprofiling es el principal del producto SME Pay. Este proporciona las mismas funcionalidades que los servicios *alert*,/control/ y *cardinfo* añadiendo toda la lógica de negocio necesaria para permitir la gestión de las tarjetas de empresa tanto por los usuarios que las tienen asignadas, como por los gestores. En SME Pay se diferencia entre 3 tipos de usuarios: Business owner, Group Admin y User.

3.1.5.1. Business Owner El business owner es el usuario que tiene todos los permisos posibles dentro de la escala de privilegios establecida en SME Pay. Puede configurar las notificaciones push para las tarjetas de compañía de forma que le lleguen notificaciones

de todas las tarjetas o notificaciones de tarjetas de grupos concretos.

Respecto a los distintos bloqueos que pueden realizarse sobre las tarjetas, también puede establecer un bloqueo sobre todas las tarjetas de la compañía o sobre las tarjetas de un grupo concreto. Cuando establece estos bloqueos, todas las tarjetas afectadas deshabilitan la gestión de dicho bloqueo al resto de usuarios.

3.1.5.2. Group Admin El business owner puede establecer un group admin para cada uno de los grupos que decida crear. Estos group admin podrán configurar las notificaciones push para las tarjetas de su propio grupo. Por otro lado, también tendrán la capacidad de establacer bloqueos en las tarjetas del grupo siempre y cuando el business owner no lo haya hecho antes.

3.1.5.3. User Todos los usuarios de SME Pay tiene por defecto privilegios de user. Pueden configurar notificaciones para la tarjeta de compañía que les ha sido asignada y bloqueos, siempre y cuando no hayan sido establecidos por el business owner o el group admin.

3.2. Interacción con SME Pay

3.2.1. Aplicación de Vipera

Para la presentación del producto SME Pay en el Finovate London 2018[27] se desarrolló un prototipo de aplicación móvil en Ionic que conectaba con el producto mediante llamadas HTTP realizadas directamente por el módulo http del core de Angular.

```
let headers = new HttpHeaders();
      headers = headers.set('Content-Type',
       ↪  'application/json');

        this.http.post(ENV.serverUrl, {
          "req": {
            "dom": ENV.domain,
            "app": ENV.application,
            "srv": "register",
            "op": "activateUser",
            "header": {
              "user": this.regForm.value.user,
              "email": this.regForm.value.email,
              "otp": this.regForm.value.otp,
              "private:pwd": this.regForm.value.password,
              "private:vpwd": this.regForm.value.verify,
            }
```

[27] https://finovate.com/videos/finovateeurope-2018-vipera-mastercard/

```
17              }
18            },
19            {headers: headers})
20            .subscribe(
21              res => {
22                if(res["res"]["header"]["err"] != undefined) {
23                  /*Codigo a ejecutar en caso de error*/
24                } else {
25                  /*Codigo a ejecutar en caso de exito*/
26                }
27              });
```

Fragmento de código 3: Llamada a la operación activation del servicio register empleando el cliente HTTP

Este método de conexión a pesar de ser completamente funcional, requería muchas lineas de código, lo que generaba un código muy denso y costoso de mantener. Por este motivo, tras el interes de diversos potenciales clientes se decidió integrar la librería MOTIF Connector, con lo que él código se reduce notablemente (ver Fragmento de código 2).

3.2.2. Integración en aplicaciones de terceros

Dado que los potenciales clientes objetivo de SME Pay son bancos y PYMES, se decidió desarrollar unas librerías nativas para Android e iOS y adicionalmente un plugin de Cordova, con lo que se facilita a los clientes la integración del producto en sus aplicaciones móviles ya funcionales, con una abstracción total de la interacción con MOTIF.

4. DESARROLLO DE LA SOLUCIÓN

4.1. Desarrollo de la librería para Android

4.1.1. Estructura del proyecto

El proyecto cuenta en su raiz con un fichero de configuracion para Gradle[28], herramienta utilizada para construir la librería.
El código se encuentra distribuido en 2 partes. Por un lado se implementan las clases que establecen la conexión con el servidor y tratan las respuestas. En el caso de Java, ya que este no contempla el uso de funciones de callback en la versión 1.7, se ha desarrollado para cada clase una interfaz asociada que será la que implemente cada función para emplearla de callback.

[28] https://gradle.org/

```java
public interface RegisterServiceCallback<T> {
    /**
     * method invoked when an {@link IRegisterService} api
 ↪  is successfully done
     * @param result api result
     */
    void onSuccess(T result);

    /**
     * method invoked when an {@link IRegisterService} api
 ↪  is done with an error
     * @param error the error
     */
    void onError(IDEError error);
}
```

Fragmento de código 4: Implementación de una interfaz de callback para el servicio register

Por otro lado, se encuentra el código encargado de serializar y deserializar las llamadas y respuestas del servidor.

4.2. Desarrollo de la libreria para iOS

4.2.1. Estructura del proyecto

El proyecto está estructurado en 2 partes. Por un lado se encuentra todo el código referente a los servicios de SME Pay, donde se establece la conexión con el servidor y se tratan las respuestas.
Por otro lado, se encuentra el código encargado de serializar y deserializar las llamadas y respuestas del servidor.
Por último, en la raiz del proyecto se encuentra el fichero `DECardControl.h`, que expone todas las cabeceras públicas de la librería para posibilitar su uso por parte de los desarrolladores.

4.3. Desarrollo del plugin para Cordova

4.3.1. Estructura del proyecto

Un plugin para Cordova tiene una estructura definida que puede ser encontrada en su documentación online[2]. Tal y como se indica, en la raiz del proyecto existe el fichero plugin.xml, el cual proporciona información acerca del plugin (nombre, versionado,

licencia...) y su composición interna. Este archivo es crucial ya que es el que lee Cordova al realizar la instalación de un plugin, con lo que un error puede provocar un mal funcionamiento del plugin.

```
<plugin id="de-smepay-plugin" version="0.0.1"
  ↪  xmlns="http://apache.org/cordova/ns/plugins/1.0"
  ↪  xmlns:android="http://schemas.android.com/apk/res/android">
  <name>SMEPayConnector</name>

  <platform name="android">

    ...

    <source-file src="src/android/CardControlPlugin.java"
     ↪  target-dir="src/com/vipera/cardcontrolplugin" />
    <source-file src="src/android/AlertServicePlugin.java"
     ↪  target-dir="src/com/vipera/cardcontrolplugin" />
    <source-file
     ↪  src="src/android/CardInfoServicePlugin.java"
     ↪  target-dir="src/com/vipera/cardcontrolplugin" />
    <source-file
     ↪  src="src/android/ControlServicePlugin.java"
     ↪  target-dir="src/com/vipera/cardcontrolplugin" />
    <source-file src="src/android/LoginServicePlugin.java"
     ↪  target-dir="src/com/vipera/cardcontrolplugin" />
    <source-file
     ↪  src="src/android/VirtualCardServicePlugin.java"
     ↪  target-dir="src/com/vipera/cardcontrolplugin" />
    <source-file
     ↪  src="src/android/CardControlServicesProvider.java"
     ↪  target-dir="src/com/vipera/cardcontrolplugin" />
    <source-file
     ↪  src="src/android/SMEProfilingServicePlugin.java"
     ↪  target-dir="src/com/vipera/cardcontrolplugin" />
    <source-file src="src/android/Encode.java"
     ↪  target-dir="src/com/vipera/cardcontrolplugin" />
    <framework custom="true"
     ↪  src="src/android/SMEPay.gradle"
     ↪  type="gradleReference" />

  </platform>


  <platform name="ios">

```

```
24      ...
25
26      <framework src="src/ios/SMEPay.framework" custom="true"
        ↪  embed="true"/>
27
28      <header-file src="src/ios/AlertServicePlugin.h" />
29      <source-file src="src/ios/AlertServicePlugin.m" />
30      <header-file src="src/ios/CardControlPlugin.h" />
31      <source-file src="src/ios/CardControlPlugin.m" />
32      <header-file src="src/ios/CardInfoServicePlugin.h" />
33      <source-file src="src/ios/CardInfoServicePlugin.m" />
34      <header-file src="src/ios/ControlServicePlugin.h" />
35      <source-file src="src/ios/ControlServicePlugin.m" />
36      <header-file src="src/ios/LoginServicePlugin.h" />
37      <source-file src="src/ios/LoginServicePlugin.m" />
38      <header-file src="src/ios/SmeProfilingServicePlugin.h"
        ↪  />
39      <source-file src="src/ios/SmeProfilingServicePlugin.m"
        ↪  />
40      <header-file src="src/ios/CardControlConfig.h" />
41      <source-file src="src/ios/CardControlConfig.m" />
42      <header-file
        ↪  src="src/ios/MotifConnectorConfiguration.h" />
43      <source-file
        ↪  src="src/ios/MotifConnectorConfiguration.m" />
44
45    </platform>
46
47  </plugin>
```

Fragmento de código 5: Código parcial del fichero plugin.xml del plugin

Como se puede observar en el código, cada plataforma para la que se ofrece soporte se configura de manera explicita, indicando los ficheros y su ubicación. Estos ficheros realizan llamadas a las librerías nativas desarrolladas, y son invocados por Cordova cuando la aplicación hace llamadas a la interfaz Javascript que proporciona el plugin.

5. RESULTADOS Y CONCLUSIONES

5.1. Pruebas

5.2. Conclusiones

6. REFERENCIAS

[1] B. de España. (2018). Estadísticas Tarjetas para la Web 2T-2018, dirección: `https://www.bde.es/f/webbde/SPA/sispago/ficheros/es/estadisticas.pdf` (visitado 05-11-2018).

[2] A. Cordova. (2018). Plugin Development Guide, dirección: `https://cordova.apache.org/docs/en/8.x/guide/hybrid/plugins/index.html` (visitado 27-12-2018).