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 EJEMPLOS DE CÓDIGO

Resumen

Con la extensión de los pagos con tarjetas bancarias en sustitución del efectivo, se ha vuelto necesario el control exhaustivo de estas para evitar robos, fraudes y descubiertos inesperados entre otras posibles situaciones. En el ambito empresarial también se ha notado esta situación, obligando a las empresas a tener un control más exhaustivo sobre los gastos que se hacen con sus tarjetas.

Este trabajo fin de grado plantea la implementación de una solución móvil nativa e híbrida para facilitar a los bancos y empresas el control de sus tarjetas, implementando una lógica de privilegios que permita controlar que usuarios tienen permisos para distintas acciones. Esta solución se integra en el proyecto SME PAY desarrollado por Vipera.

Palabras clave: MOTIF, SME Pay, iOS, Android, Cordova

Abstract

With the expansion of the payments using bank cards instead of cash, it has become mandatory to set an exhaustive control on these cards to avoid robberies, fraud and unexpected overdrafts among other possible situations. Also, in the bussiness world this situation has been noticed, forcing the companies to have a more comprenhensive control over the expenses made by their employees with the company cards.

This final degree project proposes the implementation of a native and hybrid mobile solution to make it easier for the banks and companies to control their bank cards, deploying a privileges logic that allows to control which users have the privileges to perform different actions. This solution is integrated into the SME PAY project developed by Vipera.

Keywords: MOTIF, SME Pay, iOS, Android, Cordova...

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 estas necesidades, tanto Mastercard[1] como Visa[2] han implementado distintas funcionalidades para el control de sus tarjetas, las cuales muchos bancos facilitan en la actualidad a clientes particulares. En Vipera[3] se desarrolló un prototipo para un gran banco europeo que integraba la solución de Mastercard y debido al éxito del proyecto se decidió crear un nuevo proyecto partiendo de la base de esta solución que añadiese la lógica de negocio necesaria para la integración de estos controles en un entorno empresarial.

Dado que el proyecto ha sido de gran envergadura, ha habido un equipo de 8 personas trabajando desde Madrid tanto en el desarrollo del backend como de la aplicación móvil, así como gestores coordinando el proyecto desde Londres y Milán. En mi caso, entré en el equipo cuando este se formó en Enero de 2018, y he desarrollado parte de la lógica del servidor, así como gran parte de la aplicación móvil. A principios de Junio de 2018 uno de los clientes de Vipera reclamó un SDK para conectar sus aplicaciones nativas a la solución, por lo que los desarrolladores de la aplicación móvil viajamos a Milán para formarnos y poder realizar el trabajo solicitado. A raíz de esta formación surgió la idea de proponer el desarrollo de estos SDK y su integración en un plugin de Cordova[4] como trabajo fin de grado.

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.

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

[1] `https://developer.mastercard.com/product/spend-controls`
[2] `https://developer.visa.com/capabilities/vctc`
[3] `http://www.vipera.com/`
[4] `https://cordova.apache.org/`

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.

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 so-

[5] https://bit.ly/2Cu8NVy
[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

porte para el lenguaje Java[10]. Al contrario que 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].

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 cuyo nombre comercial es MOTIF[20].

MOTIF es un servicio desarrollado en Java haciendo uso del framework OSGi[21], crea-

[10] https://www.java.com
[11] https://source.android.com/
[12] https://developers.google.com/android/guides/overview
[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/motif
[21] https://www.osgi.org/

do 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[22] como implementación del framework OSGi. El uso de este framework facilita la integración con los clientes finales, por lo general bancos 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.

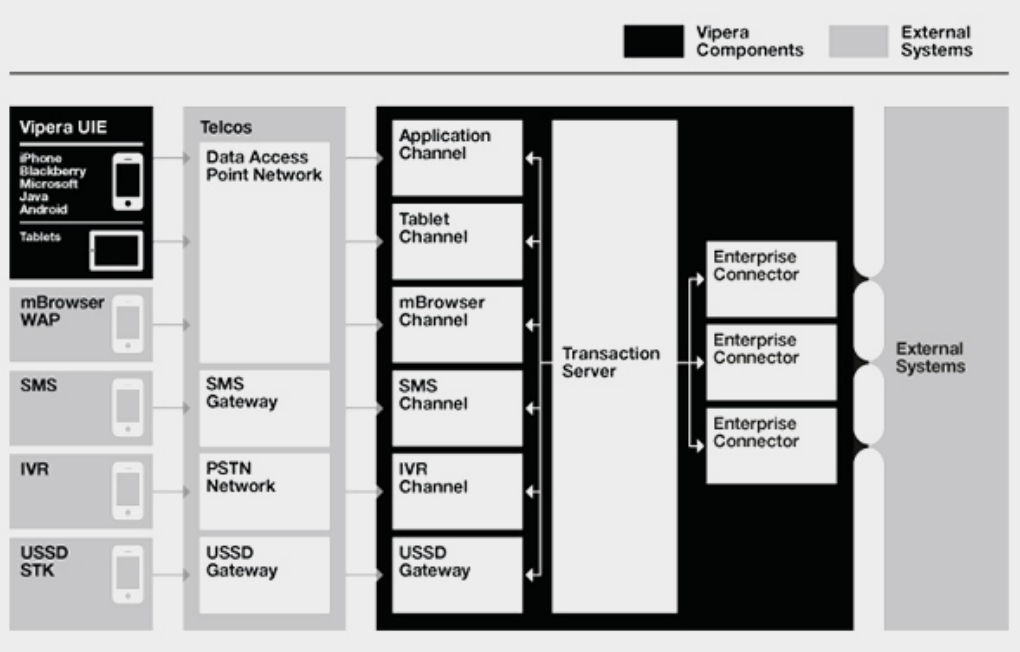

Figura 1: Diagrama de módulos predefinidos de MOTIF

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[23] 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

[22] `https://felix.apache.org/`
[23] `https://hibernate.org/`

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[24] o empleando 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.

[24] https://www.getpostman.com/

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 que 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 se 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 servidor. En este caso la llamada es de activación de un usuario, por lo que no se puede tener un ID de sesión.

```
1  let myHeader = {
2          "user":"user1@company1.com",
3          "otp":"123456",
4          "private:pwd":"1111",
5          "private:vpwd":"1111",
6          "email": "user1@company1.com"
7        };
8
9        let request: MotifRequest =
          ↪  this.motifClient.buildServerRequest("register",
          ↪  "activateUser", myHeader, false);
10
11       this.motifClient.sendRequest(request).then((res:
          ↪  MotifResponse) => {
12         /*Codigo a ejecutar en caso de exito*/
13       },(err) => {
14         /*Codigo a ejecutar en caso de error*/
15       });
```

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[25], 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

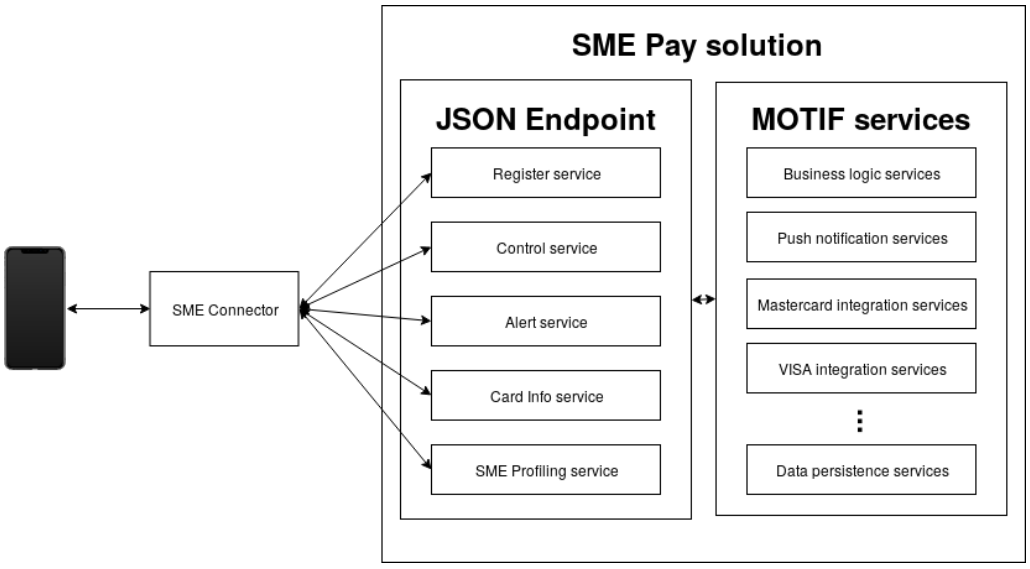

Figura 2: Servicios proporcionados por el JSON Endpoint de SME Pay

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

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

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.

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 lo permita, así como el uso de la tecnología de reconocimiento facial en los modelos de Apple que así lo permitan.

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.

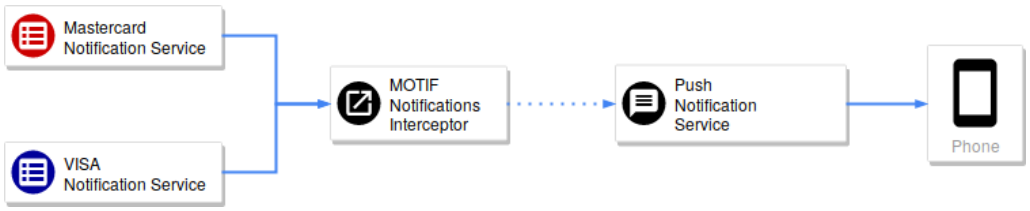

Figura 3: Diagrama de funcionamiento de notificaciones en SME Pay

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[26] se desarrolló un prototipo de aplicación móvil en Ionic[27] que conectaba con el producto mediante llamadas HTTP realizadas directamente por el módulo http del core de Angular.

```
1 let headers = new HttpHeaders();
2       headers = headers.set('Content-Type',
        ↪  'application/json');
3
4         this.http.post(ENV.serverUrl, {
5           "req": {
```

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

```
            "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,
            }
          }
        },
        {headers: headers})
        .subscribe(
          res => {
            if(res["res"]["header"]["err"] != undefined) {
              /*Codigo a ejecutar en caso de error*/
            } else {
              /*Codigo a ejecutar en caso de exito*/
            }
          });
```

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.

```
1  public interface RegisterServiceCallback<T> {
2      /**
3       * method invoked when an {@link IRegisterService} api
    ↪  is successfully done
4       * @param result api result
5       */
6      void onSuccess(T result);
7
8      /**
9       * method invoked when an {@link IRegisterService} api
    ↪  is done with an error
10      * @param error the error
11      */
12     void onError(IDEError error);
13 }
```

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.

4.1.2. Serialización y deserialización

Al realizarse un desarrollo para Android, se emplea la librería incluida en su SDK, GSON[29]. Esta librería desarrollada por Google implementa toda la funcionalidad necesaria para realizar las operaciones de serialización y deserialización.

[28] https://gradle.org/
[29] https://github.com/google/gson

Java puede crear una gran cantidad de "basura" y esta lastra el funcionamiento general de un programa al entrar el garbage collector[30] en funcionamiento. Para intentar evitar esta situación una de las prácticas que recomendiendan desde el equipo que se dedica al desarrollo móvil en las oficinas centrales de Vipera en Milan es el uso del patrón singleton siempre que se vayan a emplear objetos reutilizables, ya que este garantiza que un objeto solo tendrá una instancia que será accedida a través de un acceso global[2] (en el caso de Java un método público estático).

```
1  package com.vipera.de.cardcontrol.serializer;
2
3  import com.google.gson.Gson;
4  import com.google.gson.GsonBuilder;
5  import com.google.gson.reflect.TypeToken;
6
7  import org.json.JSONArray;
8  import org.json.JSONException;
9  import org.json.JSONObject;
10
11 import java.util.List;
12
13 /**
14  * Created by SME Project Team on 26/09/2018.
15  * Copyright © 2018 Vipera. All rights reserved.
16  */
17
18 public class JSONSerializer {
19
20     private static JSONSerializer instance;
21     private Gson gson;
22
23     private JSONSerializer(){
24         this.gson = createGSONInstance();
25     }
26
27     private Gson createGSONInstance() {
28         return new GsonBuilder()
29                 ...
30                 .create();
31     }
32
33     public static synchronized JSONSerializer getInstance()
       ↪ {
34         if(instance == null){
```

[30] https://bit.ly/2CKqpfT

```
35              instance = new JSONSerializer();
36          }
37          return instance;
38      }
39
40      public <T> T fromJson(JSONObject jsonObject, Class<T>
        ↪  classOfT) throws DEJSONParseException {
41          try {
42              return
                ↪  gson.fromJson(jsonObject.toString(),classOfT);
43          }catch (Exception ex){
44              throw new DEJSONParseException(ex);
45          }
46      }
47
48      public <T> List<T> fromJsonArray(JSONArray jsonArray)
        ↪  throws DEJSONParseException {
49          try {
50              return gson.fromJson(jsonArray.toString(), new
                ↪  TypeToken<List<T>>(){}.getType());
51          }catch (Exception ex){
52              throw new DEJSONParseException(ex);
53          }
54      }
55
56      public JSONObject toJSON(Object target) throws
        ↪  JSONException {
57          return new JSONObject(gson.toJson(target));
58      }
59
60      public JSONArray toJSONArray(Object target) throws
        ↪  JSONException {
61          return new JSONArray(gson.toJson(target));
62      }
63
64
65  }
```

Fragmento de código 5: Implementación parcial de un singleton en Java para la (de)serialización de JSON

Por otro lado, también hay que definir los objetos que van a contener los datos que pueden ser deserializados desde un JSON o serializados a este. Para ello, se define una clase con un conjunto de atributos que serán los que contengan la información. Estos atributos tienen que ser vinculados empleando la etiqueta `@SerializedName` para que

GSON pueda realizar la serialización o deserialización.

El acceso y modificación de los atributos del objeto se realiza a través de métodos públicos get y set, habiendo uno de cada para cada atributo. Dado que Java es un lenguaje muy verboso y este tipo de implementaciones ocupan mucho espacio, se adjunta el código relacionado con el JSON de la llamada a la operación *activateUser* en el Anexo A.

4.1.3. Desarrollo de un servicio

Cada servicio de SME Pay cuenta con una interfaz del servicio, una interfaz del callback y una implementación del servicio en la librería desarrollada para Android.

4.1.3.1. Interfaz del servicio Este fichero define las cabeceras de todas la operaciones del servicio, que en Java serán funciones. También define el nombre de todas las operaciones del servicio de manera que puedan ser usados por el fichero que contiene la implementación. Cabe destacar que cada función tiene siempre como parámetro el callback, que será implementado independientemente para cada una de las operaciones. Así mismo, en caso de que existan parámetros de llamada en la operación, estos también serán parametros en la función asociada.

A continuación se expone la parte de la interfaz que define la operación *activateUser*, pudiéndose encontrar todo el código en el Anexo A.

```
1  package com.vipera.de.cardcontrol.services.login;
2
3  /**
4   * Created by SME Project Team on 28/09/18.
5   * Copyright © 2018 Vipera. All rights reserved.
6   */
7
8  public interface IDELoginService {
9
10     String LOGIN_SERVICE_OP_ACTIVATE_USER = "activateUser";
11
12     /**
13      * Check the current session status.
14      * @param callback the checkSession callback. No result
    ↪  is provided in {@code onSuccess} method
15      */
16     void activateUser(String user, String email, String
       ↪  otp, String private_pwd, String private_vpwd,
       ↪  DELoginServiceCallback<Void> callback);
17 }
```

Fragmento de código 6: Implementación parcial de la interfaz del servicio register para la librería de Android

4.1.3.2. Implementación del servicio El fichero que realiza la implementación del servicio implementará todas las funciones definidas en la interfaz, y también definirá los nombres de los campos de los JSON que empleará. Por otro lado, como se ha mencionado anteriormente también tendrá que implementar el comportamiento del callback en cada una de las funciones.

En el siguiente código se puede observar la implementación de la función que llama a la operación *activateUser* del servicio register. Todo el código relacionado con la implementación de los servicios se entrega compilado en el fichero JAR que contiene la librería, de manera que su funcionamiento interno queda oculto a los usuarios mientras no empleen decompiladores.

```
1  package com.vipera.de.cardcontrol.services.login;
2
3  import android.util.Log;
4
5  import
    ↪  com.vipera.de.cardcontrol.data.login.DEActivateUserRequest;
6  import com.vipera.de.cardcontrol.serializer.JSONSerializer;
7  import com.vipera.de.cardcontrol.data.error.DEError;
8  import com.vipera.de.cardcontrol.data.error.IDEError;
9  import
    ↪  com.vipera.de.cardcontrol.data.error.exceptions.DEJSONParseExceptio
10 import
    ↪  com.vipera.de.cardcontrol.services.network.DEMotifRequest;
11 import
    ↪  com.vipera.de.cardcontrol.services.network.DEMotifRequestCallback;
12 import
    ↪  com.vipera.de.cardcontrol.services.network.DEMotifResponse;
13
14 import org.json.JSONArray;
15 import org.json.JSONException;
16 import org.json.JSONObject;
17
18 /**
19  * Created by SME Project Team on 28/09/2018.
20  * Copyright © 2018 Vipera. All rights reserved.
21  */
22
23 public class DELoginService extends DEBaseService
    ↪  implements IDELoginService {
```

```
24      String LOGIN_USER_KEY = "user";
25      String LOGIN_EMAIL_KEY = "email";
26      String LOGIN_OTP_KEY = "otp";
27      String LOGIN_PRIVATE_PWD_KEY = "private:pwd";
28      String LOGIN_PRIVATE_VPWD_KEY = "private:vpwd";
29
30      @Override
31      public void activateUser(String user, String email,
        ↪  String otp, String private_pwd, String
        ↪  private_vpwd, final DELoginServiceCallback<Void>
        ↪  callback) {
32          DEActivateUserRequest activateUserRequest = new
            ↪  DEActivateUserRequest(user, email, otp,
            ↪  private_pwd, private_vpwd);
33          JSONObject reqHeader;
34          try {
35              reqHeader =
                ↪  JSONSerializer.getInstance().toJSON(
                ↪  activateUserRequest);
36          } catch (JSONException e) {
37              callback.onError(new
                ↪  DEError(IDEError.DEErrorCode.RequestParseError));
38              return;
39          }
40          DEMotifRequest request =
            ↪  this.serverManager.buildRequestForService(
            ↪  IDELoginService.LOGIN_SERVICE_NAME,
            ↪  IDELoginService.LOGIN_SERVICE_OP_ACTIVATE_USER,
            ↪  false);
41          request.setHeader(reqHeader);
42          sendActivateUser(callback, request);
43      }
44  }
```

Fragmento de código 7: Código parcial de la implementación del servicio register para la librería de Android

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.

4.2.2. Serialización y deserialización

Debido a que Objective-C no ofrece soporte nativo para JSON, se necesita implementar clases que se encarguen de serializar y deserializar. Esto se consigue empleando la estructura de datos `NSDictionay` que como su nombre indica es un diccionario.

La estructura de estas clases es muy sencilla, tan solo tienen un constructor al que se le pasan todos los parametros a serializar y dos métodos, para serializar y deserializar. También se tiene que definir estáticamente el nombre de los campos del JSON, de forma que estos se usen como las *key* del diccionario.

```
1  //
2  //  DEActivateUserRequest.m
3  //  DECardControl
4  //
5  //  Created by SME Project Team on 03/10/18.
6  //  Copyright © 2018 Vipera. All rights reserved.
7  //
8
9  #import "DEActivateUserRequest.h"
10
11 #define DE_ACTIVATE_REQUEST_KEY_USER @"user"
12 #define DE_ACTIVATE_REQUEST_KEY_EMAIL @"email"
13 #define DE_ACTIVATE_REQUEST_KEY_OTP @"otp"
14 #define DE_ACTIVATE_REQUEST_KEY_PASSWORD @"private:pwd"
15 #define DE_ACTIVATE_REQUEST_KEY_VPASSWORD @"private:vpwd"
16
17 @interface DEActivateUserRequest ()
18
19 @property (nonatomic, strong, readwrite) NSString *userID;
20 @property (nonatomic, strong, readwrite) NSString *email;
21 @property (nonatomic, strong, readwrite) NSString *otp;
22 @property (nonatomic, strong, readwrite) NSString
    ↪  *password;
23 @property (nonatomic, strong, readwrite) NSString
    ↪  *vPassword;
24
25 @end
26
```

```
27 @implementation DEActivateUserRequest
28
29 + (instancetype)initWithDictionary:(NSDictionary
    ↪  *)dictionary error:(NSError *__autoreleasing *)error
30 {
31     NSString *userID =
        ↪  dictionary[DE_ACTIVATE_REQUEST_KEY_USER];
32     NSString *email =
        ↪  dictionary[DE_ACTIVATE_REQUEST_KEY_EMAIL];
33     NSString *otp =
        ↪  dictionary[DE_ACTIVATE_REQUEST_KEY_OTP];
34     NSString *password =
        ↪  dictionary[DE_ACTIVATE_REQUEST_KEY_PASSWORD];
35     NSString *vPassword =
        ↪  dictionary[DE_ACTIVATE_REQUEST_KEY_VPASSWORD];
36
37     return [[DEActivateUserRequest alloc]
        ↪  initWithUserID:userID email:email otp:otp
        ↪  password:password vPassword:vPassword];
38 }
39
40 - (instancetype) initWithUserID:(NSString *) userID
41                            email:(NSString *) email
42                              otp:(NSString *) otp
43                         password:(NSString *) password
44                        vPassword:(NSString *) vPassword
45 {
46     self = [super init];
47
48     if (self)
49     {
50         self.userID = userID;
51         self.email = email;
52         self.otp = otp;
53         self.password = password;
54         self.vPassword = vPassword;
55     }
56
57     return self;
58 }
59
60 - (NSDictionary *)toDictionary:(NSError *__autoreleasing
    ↪  *)error
61 {
```

```
62      NSMutableDictionary *resultDictionary =
        ↪  [NSMutableDictionary dictionary];
63
64      resultDictionary[DE_ACTIVATE_REQUEST_KEY_USER] =
        ↪  self.userID;
65      resultDictionary[DE_ACTIVATE_REQUEST_KEY_EMAIL] =
        ↪  self.email;
66      resultDictionary[DE_ACTIVATE_REQUEST_KEY_OTP] =
        ↪  self.otp;
67      resultDictionary[DE_ACTIVATE_REQUEST_KEY_PASSWORD] =
        ↪  self.password;
68      resultDictionary[DE_ACTIVATE_REQUEST_KEY_VPASSWORD] =
        ↪  self.vPassword;
69
70      return resultDictionary;
71  }
72
73  @end
```

Fragmento de código 8: Implementación de una clase de serialización/deserialización en Objective-C

4.2.3. Desarrollo de un servicio

Para proceder al desarrollo de una librería coherente con la estructura de SME Pay, se decidió diferenciar cada servicio de la misma forma. Así mismo, cada servicio contará con 3 ficheros.

4.2.3.1. Cabecera del servicio Este fichero actuará como interfaz pública del servicio, proporcionando todos los métodos que incluye el servicio, definiendo los tipos que empleará para el tratamiento de las respuestas y especificando que clases empleará para la serialización y deserialización de las llamadas y respuestas.

```
1   //
2   //  IDELoginService.h
3   //  DECardControl
4   //
5   //  Created by SME Project Team on 02/10/2018.
6   //  Copyright © 2018 Vipera. All rights reserved.
7   //
8
9   #import <Foundation/Foundation.h>
10  #import "IDEError.h"
11
```

```
12 @class DELoginRequest, DELoginResponse,
    ↪ DEActivateUserRequest, DEChangePwdRequest,
    ↪ DEEnableFingerprintRequest,
    ↪ DEEnableFingerprintResponse, DELoginFingerprintRequest;
13
14 /**
15  * Block invoked when a IDELoginService method has been
    ↪ failed
16  **/
17 typedef void(^DELoginServiceFailure)(id<IDEError> error);
18
19 ...
20
21 /**
22  * Block invoked when the activateUser (@see
    ↪ -activateUserWithSuccess:successBlock :failureBlock)
    ↪ method was successful
23  **/
24 typedef void (^DELoginServiceActivateUserSuccess)();
25
26 ...
27
28 /**
29  *
30  *  @param successBlock The block invoked when the user
    ↪ activation has been successfully
31  *  @param failureBlock The block invoked when the user
    ↪ activation has been failed
32  */
33 - (void) activateUserRequest:(DEActivateUserRequest*)
    ↪ activateRequest
    ↪ successBlock:(DELoginServiceActivateUserSuccess)
    ↪ successBlock failure:(DELoginServiceFailure)
    ↪ failureBlock;
34
35 ...
```

Fragmento de código 9: Implementación parcial de la interfaz del servicio register para la librería de iOS

Dado que la implementación completa es muy extensa, ha sido añadida en el Anexo A, dejando en esta parte del documento tan solo la implementación relativa a la llamada *activateUser*.

4.2.3.2. Implementación del servicio Dado que Objective-C es un lenguaje basado en C, emplea la misma estructura para definir librerías. Por un lado se encuentra la cabecera que mantiene la extensión `<file>.h` y por otro la implementación, que en este caso si cambia su extensión a `<file>.m`.

```
1  //
2  //  DELoginService.m
3  //  DECardControl
4  //
5  //  Created by SME Project Team on 02/10/2018.
6  //  Copyright © 2018 Vipera. All rights reserved.
7  //
8
9  ...
10
11 #import "DEActivateUserRequest.h"
12
13 ...
14
15 #import "DEMotifRequest.h"
16 #import "DEMotifResponse.h"
17 #import "DEError.h"
18
19 ...
20
21 #define LOGIN_SERVICE_OP_ACTIVATE_USER @"activateUser"
22
23 ...
24
25 @interface DELoginService ()
26
27 @end
28
29 @implementation DELoginService
30
31 ...
32
33 - (void) activateUserRequest:(DEActivateUserRequest
    ↪  *)activateRequest
    ↪  successBlock:(DELoginServiceActivateUserSuccess)successBlock
    ↪  failure:(DELoginServiceFailure)failureBlock
34 {
35     NSError *error = nil;
36     NSDictionary *activationDictionary = [activateRequest
        ↪  toDictionary:&error];
```

```
37
38      if (error)
39      {
40          failureBlock([DEError buildErrorWithError:error]);
41          return;
42      }
43
44      DEMotifRequest *request = [self.serverManager
        ↪  buildRequestForService:LOGIN_SERVICE
        ↪  operation:LOGIN_SERVICE_OP_ACTIVATE_USER
        ↪  isSecure:NO];
45
46      [request setHeader:activationDictionary];
47
48      [self.serverManager postRequest:request
        ↪  successBlock:^(DEMotifResponse *successResult,
        ↪  DEMotifRequest *serverRequest) {
49          successBlock();
50      } motifErroBlock:^(DEMotifResponse *failureResult,
        ↪  DEMotifRequest *motifRequest) {
51          failureBlock([DEError
             ↪  buildErrorWithMotifResponse:failureResult]);
52      } failureBlock:^(NSError *error, DEMotifRequest
        ↪  *motifRequest) {
53          failureBlock([DEError buildErrorWithError:error]);
54      }];
55  }
56
57  ...
58
59  @end
```

Fragmento de código 10: Código parcial de la implementación del servicio register para la librería de iOS

Debido a que el código de la implementación de los servicios se proporciona compilado, tan solo se muestra la implementación de la llamada a la operación *activateUser*, para seguir el mismo ejemplo empleado durante el documento. Todas las llamadas siguen el mismo patrón de todos modos. En primer lugar se serializan los datos de la llamada a MOTIF, contemplando el caso en el que la serialización falle, de modo que se interrumpiría la ejecución. En caso de que la serialización se produzca con éxito, se realiza la llamada a MOTIF y se procesa la respuesta mediante el uso de funciones inline.

En el caso de la operación *activateUser* no se recibe ningún tipo de información que deba ser deserializada, pero en el caso de las operaciones en lo que esto si ocurre, se

deserializa en la función inline correspondiente al éxito de la llamada, y se devuelve el objeto deserializado dentro del método `successBlock()`.

4.3. Desarrollo del plugin para Cordova

La principal utilidad del uso de Cordova es la abstracción que ofrece a la hora de ejecutar plugins que contienen código nativo. En tiempo de ejecución, Cordova es capaz de determinar sobre que plataforma se está ejecutando y determinar a que código nativo tiene que invocar, abstrayendo al programador de este problema.

4.3.1. Estructura del proyecto

Un plugin para Cordova tiene una estructura definida que puede ser consultada en su documentación online[3]. 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.

```
1  <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">
2    <name>SMEPayConnector</name>
3
4    <js-module src="www/CardControl.js" name="CardControl">
5      <clobbers target="DynamicEngine.plugins.CardControl"/>
6    </js-module>
7    <js-module src="www/LoginService.js" name="LoginService">
8      <clobbers
        ↪  target="DynamicEngine.plugins.CardControl.LoginService"/>
9    </js-module>
10   <js-module src="www/CardInfoService.js"
      ↪  name="CardInfoService">
11     <clobbers
        ↪  target="DynamicEngine.plugins.CardControl.CardInfoService"/>
12   </js-module>
13   <js-module src="www/ControlService.js"
      ↪  name="ControlService">
14     <clobbers
        ↪  target="DynamicEngine.plugins.CardControl.ControlService"/>
15   </js-module>
16   <js-module src="www/AlertService.js" name="AlertService">
17     <clobbers
        ↪  target="DynamicEngine.plugins.CardControl.AlertService"/>
18   </js-module>
```

```
19    <js-module src="www/VirtualCardService.js"
       ↪  name="VirtualCardService">
20      <clobbers
         ↪  target="DynamicEngine.plugins.CardControl.VirtualCardService"/>
21    </js-module>
22    <js-module src="www/SMEProfilingService.js"
       ↪  name="SMEProfilingService">
23      <clobbers
         ↪  target="DynamicEngine.plugins.CardControl.SMEProfilingService"/
24    </js-module>
25
26    <platform name="android">
27
28      ...
29
30      <source-file src="src/android/CardControlPlugin.java"
         ↪  target-dir="src/com/vipera/cardcontrolplugin" />
31      <source-file src="src/android/AlertServicePlugin.java"
         ↪  target-dir="src/com/vipera/cardcontrolplugin" />
32      <source-file
         ↪  src="src/android/CardInfoServicePlugin.java"
         ↪  target-dir="src/com/vipera/cardcontrolplugin" />
33      <source-file
         ↪  src="src/android/ControlServicePlugin.java"
         ↪  target-dir="src/com/vipera/cardcontrolplugin" />
34      <source-file src="src/android/LoginServicePlugin.java"
         ↪  target-dir="src/com/vipera/cardcontrolplugin" />
35      <source-file
         ↪  src="src/android/VirtualCardServicePlugin.java"
         ↪  target-dir="src/com/vipera/cardcontrolplugin" />
36      <source-file
         ↪  src="src/android/CardControlServicesProvider.java"
         ↪  target-dir="src/com/vipera/cardcontrolplugin" />
37      <source-file
         ↪  src="src/android/SMEProfilingServicePlugin.java"
         ↪  target-dir="src/com/vipera/cardcontrolplugin" />
38      <source-file src="src/android/Encode.java"
         ↪  target-dir="src/com/vipera/cardcontrolplugin" />
39      <framework custom="true"
         ↪  src="src/android/SMEPay.gradle"
         ↪  type="gradleReference" />
40
41    </platform>
42
```

```
43
44    <platform name="ios">
45
46      ...
47
48      <framework src="src/ios/SMEPay.framework" custom="true"
          ↪  embed="true"/>
49
50      <header-file src="src/ios/AlertServicePlugin.h" />
51      <source-file src="src/ios/AlertServicePlugin.m" />
52      <header-file src="src/ios/CardControlPlugin.h" />
53      <source-file src="src/ios/CardControlPlugin.m" />
54      <header-file src="src/ios/CardInfoServicePlugin.h" />
55      <source-file src="src/ios/CardInfoServicePlugin.m" />
56      <header-file src="src/ios/ControlServicePlugin.h" />
57      <source-file src="src/ios/ControlServicePlugin.m" />
58      <header-file src="src/ios/LoginServicePlugin.h" />
59      <source-file src="src/ios/LoginServicePlugin.m" />
60      <header-file src="src/ios/SmeProfilingServicePlugin.h"
          ↪  />
61      <source-file src="src/ios/SmeProfilingServicePlugin.m"
          ↪  />
62      <header-file src="src/ios/CardControlConfig.h" />
63      <source-file src="src/ios/CardControlConfig.m" />
64      <header-file
          ↪  src="src/ios/MotifConnectorConfiguration.h" />
65      <source-file
          ↪  src="src/ios/MotifConnectorConfiguration.m" />
66
67    </platform>
68
69  </plugin>
```

Fragmento de código 11: 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.

4.3.2. Módulos de conexión para Android

Debido a que los módulos de Android son usados únicamente de manera interna por Cordova para conectar la aplicación híbrida con la librería para Android desarrollada, estos no requieren una interfaz que exponga las funciones al exterior. De este modo tan solo

contaremos con una clase que extenderá a `CordovaPlugin` (clase abstracta proporcionada por Cordova) por cada módulo que se implemente. Esta clase tan solo contará con dos métodos, `initizalize` y `execute`, siendo el primero el que inicialice el módulo y establezca la conexión entre la aplicación y la librería, y el segundo el que se encargue de interceptar las llamadas que llegan desde la interfaz de Javascript.

```
1  package com.vipera.cardcontrolplugin;
2
3  import android.util.Log;
4
5  import com.vipera.de.cardcontrol.data.error.IDEError;
6  import com.vipera.de.cardcontrol.data.login.DELoginRequest;
7  import
    ↪  com.vipera.de.cardcontrol.data.login.DELoginResponse;
8  import
    ↪  com.vipera.de.cardcontrol.services.login.DELoginServiceCallback;
9  import
    ↪  com.vipera.de.cardcontrol.services.login.IDELoginService;
10
11 import org.apache.cordova.CordovaInterface;
12 import org.apache.cordova.CordovaPlugin;
13 import org.apache.cordova.CallbackContext;
14
15 import org.apache.cordova.CordovaWebView;
16 import org.apache.cordova.PluginResult;
17 import org.json.JSONArray;
18 import org.json.JSONException;
19 import org.json.JSONObject;
20
21
22 public class LoginServicePlugin extends CordovaPlugin {
23
24     private static final String EXCEPTION_RESPONSE =
       ↪  "header";
25
26     ...
27
28     @Override
29     public void initialize(CordovaInterface cordova,
       ↪  CordovaWebView webView) {
30         loginService = loginService();
31         super.initialize(cordova, webView);
32     }
33
34     @Override
```

```
35      public boolean execute(String action, JSONArray args,
         ↪  final CallbackContext callbackContext) throws
         ↪  JSONException {
36          switch (action) {
37
38              ...
39
40              case "activateUser": {
41                  cordova.getThreadPool().execute(new
                     ↪  Runnable() {
42                      public void run() {
43                          try {
44                              loginService.activateUser(
                                 ↪  args.getString(0),
                                 ↪  args.getString(1),
                                 ↪  args.getString(2),
                                 ↪  args.getString(3),
                                 ↪  args.getString(4), new
                                 ↪  DELoginServiceCallback<Void>()
                                 ↪  {
45                                  @Override
46                                  public void onSuccess(Void
                                     ↪  result) {
47                                      Log.i("activateUser",
                                         ↪  "success");
48                                  }
49                                  @Override
50                                  public void
                                     ↪  onError(IDEError error)
                                     ↪  {
51                                      handleException(error,
                                         ↪  callbackContext);
52                                      Log.e("activateUser
                                         ↪  error",error.getErrorCode()
                                         ↪  + "  -  " +
                                         ↪  error.getErrorMessage());
53                                  }
54                              });
55                          } catch (JSONException e ){
56                              Log.e("activateUser",
                                 ↪  e.toString());
57                          }
58                      }
59                  });
```

```
60                      break;
61                  }
62
63                  ...
64
65              }
66
67              return true;
68          }
69
70          private void handleException(IDEError error, final
             ↪  CallbackContext callbackContext) {
71              JSONObject JSONerror = new
                 ↪  Encode().encodeToJson(error,
                 ↪  EXCEPTION_RESPONSE);
72              PluginResult finalResult = new
                 ↪  PluginResult(PluginResult.Status.ERROR,
                 ↪  JSONerror);
73              finalResult.setKeepCallback(true);
74              callbackContext.sendPluginResult(finalResult);
75          }
76      }
```

Fragmento de código 12: Implementación del conector de Android del plugin para el servicio register

4.3.3. Módulos de conexión para iOS

Para la comunicación entre el plugin y el framework desarrollado para iOS como parte de la solución, se emplean unos conectores que siguen la misma estructura que cualquier implementación en Objective-C, siendo necesarios los ficheros de cabecera e implementación.

4.3.3.1. Cabecera del módulo Este fichero es el que consultará Cordova para ejecutar el código nativo, por lo que el nombre de las funciones debe ser igual al definido por la interfaz Javascript. Cada una de las funciones definidas recibirá siempre un único parametro del tipo `CDVInvokedUrlCommand`, el cual contendrá toda la información suministrada desde la aplicación.

```
1  #import <Cordova/CDVPlugin.h>
2  #import <DECardControl/DEActivateUserRequest.h>
3  #import "CardControlConfig.h"
4  #import "MotifConnectorConfiguration.h"
5  #import <DECardControl/DECardControlService.h>
```

```
6
7  @interface LoginServicePlugin : CDVPlugin {
8  }
9
10 // The hooks for our plugin commands
11 -(void) initialize:(CDVInvokedUrlCommand *)command;
12 -(void) loginOp:(CDVInvokedUrlCommand *)command;
13 -(void) activateUserOp:(CDVInvokedUrlCommand *) command;
14 -(void) changePwdOp:(CDVInvokedUrlCommand *) command;
15 -(void) enableFingerprintOp:(CDVInvokedUrlCommand
    ↪  *)command;
16 -(void) logoutOp:(CDVInvokedUrlCommand *) command;
17 -(void) loginFingerprintOp:(CDVInvokedUrlCommand *)
    ↪  command;
18 @end
```

Fragmento de código 13: Implementación de la cabecera del conector de iOS del plugin para el servicio register

4.3.3.2. Implementación del módulo En la implementación del módulo se encuentra el código que invocará a la librería para iOS desarrollada como parte de la solución, así como toda la lógica de procesamiento de los datos de entrada. Como se puede observar en el código, los parámetros de entrada se obtienen de la estructura `arguments`, contenida en el parámetro `command` que recibe la función. Al recogerse estos parametros por su posición en la estructura de datos, la posición de cada uno de los parámetros viene predefinida en la documentación asociada a la solución, que es suministrada a los clientes.

```
1  #import "LoginServicePlugin.h"
2
3  #import <Cordova/CDVAvailability.h>
4  #import <DECardControl/IDELoginService.h>
5  #import <DECardControl/DEActivateUserRequest.h>
6  #import "CardControlConfig.h"
7  #import "MotifConnectorConfiguration.h"
8  #import <DECardControl/DECardControlService.h>
9  #import "CardControlPlugin.h"
10
11 @interface LoginServicePlugin()
12
13 @property (nonatomic, strong) DECardControlService
    ↪  *cardControlService;
14
15 @end
```

```
16
17 @implementation LoginServicePlugin
18
19 ...
20
21 -(void) activateUserOp:(CDVInvokedUrlCommand *)command
22 {
23     NSString *userID = [command.arguments objectAtIndex:0];
24     NSString *userEmail = [command.arguments
       ↪  objectAtIndex:1];
25     NSString *otp = [command.arguments objectAtIndex:2];
26     NSString *password = [command.arguments
       ↪  objectAtIndex:3];
27     NSString *vPassword = [command.arguments
       ↪  objectAtIndex:4];
28
29     DEActivateUserRequest *activateRequest =
       ↪  [[DEActivateUserRequest alloc]
       ↪  initWithUserID:userID email:userEmail otp:otp
       ↪  password:password vPassword:vPassword];
30
31     id<IDELoginService> loginService =
       ↪  [self.cardControlService loginService];
32
33     [loginService activateUserRequest:activateRequest
       ↪  successBlock:^{
34         CDVPluginResult * result = [CDVPluginResult
           ↪  resultWithStatus:CDVCommandStatus_OK];
35         [self.commandDelegate sendPluginResult:result
           ↪  callbackId:command.callbackId];
36     } failure:^(id<IDEError> error) {
37         CDVPluginResult * result = [CDVPluginResult
           ↪  resultWithStatus:CDVCommandStatus_ERROR
           ↪  messageAsString:[error motifInternalError]];
38         [self.commandDelegate sendPluginResult:result
           ↪  callbackId:command.callbackId];
39     }];
40 }
41
42 ...
43
44 @end
```

Fragmento de código 14: Código de la implementación del conector de iOS del plugin para el servicio register

4.3.4. Interfaces Javascript

Las interfaces Javascript se ubican en la carpeta `www` del plugin de Cordova. Estas se comunican con el código nativo mediante la llamada a la operación `exec` incluida en la librería proporcionada por Cordova para comunicarse con el código nativo.

Cada interfaz contiene un JSON que define todas las funciones, definiendo como parámetros un callback de éxito, un callback de fallo y una lista que contendrá los datos a procesar por el código nativo. Estos parámetros son pasados a la función `exec` junto al nombre de la clase homóloga en código nativo y el nombre de la función a invocar.

Este JSON será exportado por la interfaz de manera que pueda ser invocado por la aplicación híbrida.

```
1  var exec = require('cordova/exec');
2
3  var PLUGIN_NAME = 'LoginServicePlugin';
4
5  var LoginServiceiOSPlugin = {
6
7    initialize: function(cb) {
8      exec(cb, null, PLUGIN_NAME, 'initialize', []);
9    },
10   login: function (successCallback, failureCallback, args)
     ↪  {
11
       ↪  exec(successCallback,failureCallback,PLUGIN_NAME,'login',args);
12   },
13   activateUser: function(successCallback, failureCallback,
     ↪  args){
14     exec(successCallback, failureCallback, PLUGIN_NAME,
       ↪  'activateUser', args);
15   },
16   changePwd: function(successCallback, failureCallback,
     ↪  args){
17     exec(successCallback, failureCallback, PLUGIN_NAME,
       ↪  'changePwd', args);
18   },
19   enableFingerprint: function (successCallback,
     ↪  failureCallback, args) {
20     exec(successCallback,failureCallback, PLUGIN_NAME,
       ↪  'enableFingerprint', args);
```

```
21    },
22    logout: function (successCallback, failureCallback, args)
      ↪  {
23      exec(successCallback, failureCallback, PLUGIN_NAME,
        ↪  'logout', args);
24    },
25    loginFingerprint: function (successCallback,
      ↪  failureCallback, args) {
26      exec(successCallback, failureCallback, PLUGIN_NAME,
        ↪  'loginFingerprint', args);
27    }
28  };
29
30  module.exports = LoginServiceiOSPlugin;
```

Fragmento de código 15: Implementación de una interfaz Javascript para el plugin de Cordova

Con la interfaz exportada, tan solo habrá que realizar una llamada a la operación elegida y definir el comportamiento de los callback.

```
1  (<any>window).LoginServicePlugin.activateUser((isSuccess)
   ↪  => {
2          /*Codigo a ejecutar en caso de exito*/
3        }, (isFailure) => {
4          /*Codigo a ejecutar en caso de error*/
5        }, ["user1@company1.com", "user1@company1.com",
          ↪  "123456", "1111", "1111"]);
```

Fragmento de código 16: Ejemplo de una llamada empleando el plugin desarrollado

5. CONCLUSIONES

Tras el desarrollo de la solución híbrida he sacado las siguientes conclusiones:

- El desarrollo móvil nativo proporciona claras ventajas respecto a las soluciones híbridas basadas en el uso un navegador web embebido dentro de una aplicación. Sin duda el desarrollo híbrido abarata mucho los costes, ya que no es necesario que el desarrollador controle 2 lenguajes y sus librerías asociadas como sí sucede con la programación nativa, pero aun sin ser necesario el conocimiento de estas tecnologías no deja de ser recomendable de cara a la posibilidad de realizar desarrollos similares al ejecutado en este Proyecto Fin de Grado.

- Las empresas del sector financiero tienen un miedo relativo al cambio. A pesar de que invierten una gran cantidad de recursos en mejorar y renovar sus sistemas y tecnologías, el tamaño de estas empresas y su necesidad de seguridad las hace implementar cambios en el negocio con una velocidad insuficiente en muchos casos. En el caso de este proyecto, se exploró la posibilidad de realizar el desarrollo en los lenguajes modernos para desarrollo nativo (Kotlin y Swift), pero se terminó descartando la idea debido a que la inmensa mayoría de potenciales clientes finales emplean aun Java y Objective-C en sus aplicaciones, y a pesar de que estas tecnologías son compatibles entre ellas en un principio, se sienten más cómodos empleando librerías desarrolladas en los lenguajes usados por ellos.

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] L. Welicki. (2018). El Patrón Singleton, dirección: `https://msdn.microsoft.com/es-es/library/bb972272.aspx` (visitado 01-01-2019).

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

7. ANEXO A - CÓDIGO

7.1. Objeto DEActivateUserRequest

```
1  package com.vipera.de.cardcontrol.data.login;
2
3  import com.google.gson.annotations.SerializedName;
4
5  public class DEActivateUserRequest {
6      public static final String DE_LOGIN_REQUEST_KEY_USER  =
       ↪  "user";
7      public static final String DE_LOGIN_REQUEST_KEY_EMAIL
       ↪  = "email";
8      public static final String DE_LOGIN_REQUEST_KEY_OTP  =
       ↪  "otp";
9      public static final String
       ↪  DE_LOGIN_REQUEST_KEY_PRIVATE_PWD  = "private:pwd";
10     public static final String
       ↪  DE_LOGIN_REQUEST_KEY_PRIVATE_VPWD  =
       ↪  "private:vpwd";
11
12     @SerializedName(value = DE_LOGIN_REQUEST_KEY_USER)
13     private String user;
14
15     @SerializedName(value = DE_LOGIN_REQUEST_KEY_EMAIL)
16     private String email;
17
18     @SerializedName(value =  DE_LOGIN_REQUEST_KEY_OTP)
19     private String otp;
20
21     @SerializedName(value =
       ↪  DE_LOGIN_REQUEST_KEY_PRIVATE_PWD)
22     private String private_pwd;
23
24     @SerializedName(value =
       ↪  DE_LOGIN_REQUEST_KEY_PRIVATE_VPWD)
25     private String private_vpwd;
26
27     public DEActivateUserRequest() {}
28
29     public DEActivateUserRequest(String user, String email,
       ↪  String otp, String private_pwd, String
       ↪  private_vpwd) {
30         this.user = user;
```

```
31          this.email = email;
32          this.otp = otp;
33          this.private_pwd = private_pwd;
34          this.private_vpwd = private_vpwd;
35      }
36
37      public String getUser() {
38          return user;
39      }
40
41      public void setUser(String user) {
42          this.user = user;
43      }
44
45      public String getEmail() {
46          return email;
47      }
48
49      public void setEmail(String email) {
50          this.email = email;
51      }
52
53      public String getOtp() {
54          return otp;
55      }
56
57      public void setOtp(String otp) {
58          this.otp = otp;
59      }
60
61      public String getPrivate_pwd() {
62          return private_pwd;
63      }
64
65      public void setPrivate_pwd(String private_pwd) {
66          this.private_pwd = private_pwd;
67      }
68
69      public String getPrivate_vpwd() {
70          return private_vpwd;
71      }
72
73      public void setPrivate_vpwd(String private_vpwd) {
74          this.private_vpwd = private_vpwd;
```

```
75        }
76    }
```

Fragmento de código 17: Implementación del objeto DEActivateUserRequest en Java

7.2. Interfaz del servicio register para Android

```
1  package com.vipera.de.cardcontrol.services.login;
2
3  import com.vipera.de.cardcontrol.data.login.DELoginRequest;
4  import
    ↪  com.vipera.de.cardcontrol.data.login.DELoginResponse;
5
6  /**
7   * Created by SME Project Team on 28/09/18.
8   * Copyright © 2018 Vipera. All rights reserved.
9   */
10
11 /**
12  * This service is needed for performing login/logut
    ↪  operations to the MOTIF Card Control Service.
13  *
14  * Before to performs Card Control SDK operations you need
    ↪  to logging into the remote MOTIF Card Control Service
    ↪  to open a valid session.
15  * The duration of this session depends on parameters set
    ↪  on MOTIF service side. When the session expires the app
    ↪  needs to make a new login.
16  *
17  * With checkSession method exposed by this service you can
    ↪  check if a session is still valid.
18  * For example you can check the session after the app
    ↪  returns in foreground.
19  *
20  */
21 public interface IDELoginService {
22
23      String COMMON_SERVICE_NAME = "COMMON";
24      String COMMON_SERVICE_OP_CHECK_SESSION ="checkSession";
25      String LOGIN_SERVICE_NAME = "register";
26      String LOGIN_SERVICE_OP_LOGIN = "login";
27      String LOGIN_SERVICE_OP_LOGOUT = "logout";
28      String LOGIN_SERVICE_OP_ACTIVATE_USER = "activateUser";
```

```
29      String LOGIN_SERVICE_OP_ENABLE_FINGERPRINT=
        ↪  "enableFingerprint";
30      String LOGIN_SERVICE_OP_LOGIN_FINGERPRINT =
        ↪  "loginFingerprint";
31      String LOGIN_SERVICE_OP_REGISTER_USER = "registerUser";
32      String COMMON_SERVICE_OP_CHANGE_PWD = "changePwd";
33
34      /**
35       * Send a Login request to the MOTIF Service.
36       * The operation allows registered users to log into
    ↪  the MOTIF Card Control Service.
37       * {@code DELoginServiceCallback.onError} is invoked
    ↪  when a problem is detected: for example no Network
    ↪  available, remote service not reachable, etc...).
38       * If the login success a new session starts. The
    ↪  session can be expire depending on configuraiton on
    ↪  MOTIF Server. When a session expires, a new login is
    ↪  required.
39       * Possible (applicative) error values:
40       * <ul>
41       *    <li> USER_NOTFOUND : User not found</li>
42       *    <li> ACTIVATION_REQUIRED : Activation
    ↪  required</li>
43       *    <li> AUTHENTICATION_FAILURE : Wrong password</li>
44       *    <li> APP_BLOCKED : Application instance has been
    ↪  blocked</li>
45       *    <li> ACCOUNT_SUSPENDED : User account has been
    ↪  suspended due to too many login failures</li>
46       * </ul>
47       *
48       *
49       * @param loginRequest the login request. See {@link
    ↪  DELoginRequest} for details
50       * @param callback the login callback. See {@link
    ↪  DELoginResponse} for more details if login is
    ↪  successfully completed
51       */
52      void login(DELoginRequest loginRequest,
        ↪  DELoginServiceCallback<DELoginResponse> callback);
53
54
55      /**
56       * Send a Logout request to the MOTIF Service and
    ↪  invalidate the current session.
```

```
57          * @param callback the logout callback. No result is
    ↪  provided in {@code onSuccess} method
58          */
59         void logout(DELoginServiceCallback<Void> callback);
60
61
62         /**
63          * Check the current session status.
64          * @param callback the checkSession callback. No result
    ↪  is provided in {@code onSuccess} method
65          */
66         void checkSession(DELoginServiceCallback<Void>
    ↪  callback);
67
68         /**
69          * Check the current session status.
70          * @param callback the checkSession callback. No result
    ↪  is provided in {@code onSuccess} method
71          */
72         void activateUser(String user, String email, String
    ↪  otp, String private_pwd, String private_vpwd,
    ↪  DELoginServiceCallback<Void> callback);
73
74         /**
75          * Check the current session status.
76          * @param callback the checkSession callback. No result
    ↪  is provided in {@code onSuccess} method
77          */
78         void changePwd(String private_pwd, String private_npwd,
    ↪  String private_vpwd, DELoginServiceCallback<Void>
    ↪  callback);
79
80         /**
81          * Check the current session status.
82          * @param callback the checkSession callback. No result
    ↪  is provided in {@code onSuccess} method
83          */
84         void enableFingerprint(String user,
    ↪  DELoginServiceCallback<String> callback);
85
86         /**
87          * Check the current session status.
88          * @param callback the checkSession callback. No result
    ↪  is provided in {@code onSuccess} method
```

```
89          */
90         void loginFingerprint(String user, String FINGER_TOKEN,
            ↪  DELoginServiceCallback<Void> callback);
91         /**
92          * Check the current session status.
93          * @param callback the checkSession callback. No result
    ↪   is provided in {@code onSuccess} method
94          */
95         void registerUser(String user, String maskedPhone,
            ↪  DELoginServiceCallback<Void> callback);
96     }
```

Fragmento de código 18: Implementación completa de la interfaz del servicio register para la librería de Android

7.3. Interfaz del servicio register para iOS

```
1  //
2  //  IDELoginService.h
3  //  DECardControl
4  //
5  //  Created by SME Project Team on 02/10/2018.
6  //  Copyright © 2018 Vipera. All rights reserved.
7  //
8
9  #import <Foundation/Foundation.h>
10 #import "IDEError.h"
11
12 @class DELoginRequest, DELoginResponse,
    ↪  DEActivateUserRequest, DEChangePwdRequest,
    ↪  DEEnableFingerprintRequest,
    ↪  DEEnableFingerprintResponse, DELoginFingerprintRequest;
13
14 /**
15  * Block invoked when a IDELoginService method has been
    ↪  failed
16  **/
17 typedef void(^DELoginServiceFailure)(id<IDEError> error);
18
19 /**
20  * Block invoked when the login method (@see
    ↪  IDELoginService:loginWithRequest:successBlock:failureBlock:)
    ↪  was successful
```

```
21   **/
22  typedef void(^DELoginServiceLoginSuccess)(DELoginResponse
    ↪  *loginResponse);
23
24  /**
25   * Block invoked when the logout (@see
    ↪  -logoutWithSuccessBlock:successBlock :failureBlock)
    ↪  method was successful
26   **/
27  typedef void(^DELoginServiceLogoutSuccess)();
28
29  /**
30   * Block invoked when the checkSession (@see
    ↪  -checkSessionWithSuccess:successBlock :failureBlock)
    ↪  method was successful
31   **/
32  typedef void(^DELoginServiceCheckSessionSuccess)();
33
34  /**
35   * Block invoked when the activateUser (@see
    ↪  -activateUserWithSuccess:successBlock :failureBlock)
    ↪  method was successful
36   **/
37  typedef void (^DELoginServiceActivateUserSuccess)();
38
39  /**
40   * Block invoked when the changePwd (@see -) method was
    ↪  successful
41   **/
42
43  typedef void (^DELoginServiceChangePwdSucces)();
44
45  /**
46   *  Block invoked when the enable fingerprint method was
    ↪  successful.
47   */
48  typedef void
    ↪  (^DELoginServiceEnableFingerprintSuccess)(DEEnableFingerprintRespon
    ↪  enableFingerprintResponse);
49
50  /**
51   *  Block invoked when the login fingerprint was
    ↪  successful.
52   */
```

```
53 typedef void
    ↪  (^DELoginServiceLoginFingerprintSuccess)(DELoginResponse*
    ↪  loginFingerprintResponse);
54
55 /**
56  *  Block invoked when the user registration was
    ↪  successful.
57  */
58 typedef void (^DELoginServiceRegisterUserSuccess)();
59
60 /**
61  * This service is needed for performing login/logut
    ↪  operations to the MOTIF SME Pay application.
62  *
63  * Before to performs SME Pay SDK operations you need to
    ↪  logging into the remote MOTIF SME Pay application to
    ↪  open a valid session.
64  * The duration of this session depends on parameters set
    ↪  on MOTIF service side. When the session expires the app
    ↪  needs to make a new login.
65  *
66  * With checkSessionWithSuccess method exposed by this
    ↪  service you can check if a session is still valid.
67  * For example you can check the session after the app
    ↪  returns in foreground.
68  *
69  */
70 @protocol IDELoginService <NSObject>
71
72 /**
73  * Send a Login request to the MOTIF Service.
74  * The operation allows registered users to log into the
    ↪  MOTIF SME Pay application.
75  *
76  * A failure block is invoked when a low level problem is
    ↪  detected: for example no Network available, remote
    ↪  service not reachable, etc...)
77  * If the login success a new session starts. The session
    ↪  can be expire depending on configuraiton on MOTIF
    ↪  Server. When a session expires, a new login is
    ↪  required.
78  *
79  * Possible error values:
80  *
```

```
81   *  **USER_NOTFOUND**:                    User not found
82   *
83   *  **ACTIVATION_REQUIRED**:              Activation required
84   *
85   *  **AUTHENTICATION_FAILURE**:           Wrong password
86   *
87   *  **APP_BLOCKED**:                      Application instance
     ↪  has been blocked
88   *
89   *  **ACCOUNT_SUSPENDED**:                User account has
     ↪  been suspended due to too many login failures
90   *
91   *  @param loginRequest The Login requet (@see
     ↪  DELoginRequest class)
92   *  @param successBlock The block invoked when the login
     ↪  has been completed successfully
93   *  @param failureBlock TThe block invoked when the login
     ↪  has been failed
94   */
95  - (void) loginWithRequest:(DELoginRequest *) loginRequest
     ↪  successBlock:(DELoginServiceLoginSuccess) successBlock
     ↪  failureBlock:(DELoginServiceFailure) failureBlock;
96
97  /**
98   * Send a Logout request to the MOTIF Service and
     ↪  invalidate the current session.
99   *
100  *  @param successBlock The block invoked when the login
     ↪  has been completed successfully
101  *  @param failureBlock The block invoked when the logout
     ↪  has been failed
102  */
103 - (void)
     ↪  logoutWithSuccessBlock:(DELoginServiceLogoutSuccess)
     ↪  successBlock failureBlock:(DELoginServiceFailure)
     ↪  failureBlock;
104
105 /**
106  * Check the current session status.
107  *
108  *  @param successBlock The block invoked when the session
     ↪  has been checked and remote validated successfully
109  *  @param failureBlock The block invoked when the login
     ↪  has been failed
```

```
110  */
111 - (void)
    ↪ checkSessionWithSuccess:(DELoginServiceCheckSessionSuccess)
    ↪ successBlock failure:(DELoginServiceFailure)
    ↪ failureBlock;
112
113 /**
114  *
115  * @param successBlock The block invoked when the user
    ↪ activation has been successfully
116  * @param failureBlock The block invoked when the user
    ↪ activation has been failed
117  */
118 - (void) activateUserRequest:(DEActivateUserRequest*)
    ↪ activateRequest
    ↪ successBlock:(DELoginServiceActivateUserSuccess)
    ↪ successBlock failure:(DELoginServiceFailure)
    ↪ failureBlock;
119
120 /**
121  *
122  * @param successBlock The block invoked when the change
    ↪ password operation has been successfully
123  * @param failureBlock The block invoked when the
    ↪ changepassword operation has been failed.
124  */
125 - (void) changePwdRequest:(DEChangePwdRequest*)
    ↪ changePwdRequest
    ↪ successBlock:(DELoginServiceChangePwdSucces)
    ↪ successBlock failure:(DELoginServiceFailure)
    ↪ failureBlock;
126
127 /**
128  *
129  * @param successBlock The block invoked when the enable
    ↪ fingerprint operation has been successfully.
130  * @param failureBlock The block invoked when the enable
    ↪ fingerprint operation has failed.
131  */
132 - (void) enableFingerprintWithRequest:
    ↪ (DEEnableFingerprintRequest *) enableFingerprintRequest
    ↪ successBlock:(DELoginServiceEnableFingerprintSuccess)
    ↪ successBlock failureBlock:(DELoginServiceFailure)
    ↪ failureBlock;
```

```
133
134 /**
135  *
136  *  @param successBlock The block invoked when the login
    ↪  fingerprint operation has been successfully.
137  *  @param failureBlock The block invoked when the login
    ↪  fingerprint operation has failed.
138  */
139 - (void) loginWithFingerprint: (DELoginFingerprintRequest
    ↪  *) loginFingerprintRequest
    ↪  successBlock:(DELoginServiceLoginFingerprintSuccess)
    ↪  successBlock failureBlock:(DELoginServiceFailure)
    ↪  failureBlock;
140
141 /**
142  *
143  *  @param userID The user ID
144  *  @param maskedPhone The phone number masked
145  *  @param successBlock The block invoked when the login
    ↪  fingerprint operation has been successfully.
146  *  @param failureBlock The block invoked when the login
    ↪  fingerprint operation has failed.
147  */
148 - (void) registerUserWithUserID:(NSString *)userID
    ↪  maskedPhone:(NSString *) maskedPhone
    ↪  success:(DELoginServiceRegisterUserSuccess)
    ↪  successBlock failure:(DELoginServiceFailure)
    ↪  failureBlock;
149
150
151 @end
```

Fragmento de código 19: Implementación completa de la interfaz del servicio register para la librería de iOS