ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

GRADO EN INGENIERÍA TELEMÁTICA

MODELADO DE COMUNICACIONES ENTRE SERVICIOS INTERNOS Y APLICACIONES EXTERNAS MEDIANTE APIs EN EL ENTORNO BANCARIO

Autor: Joaquín Liaño Díaz

Director: Atilano Fernández-Pacheco Sánchez-Migallón

Madrid

Julio 2017

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

MODELADO DE COMUNICACIONES ENTRE SERVICIOS INTERNOS Y APLICACIONES EXTERNAS MEDIANTE APIs EN EL ENTORNO BANCARIO

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2016/17 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: Joaquín Liaño Díaz Fecha: 07 / 07 / 2017

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Atilano Fernández-Pacheco Sánchez-Migallón Fecha: 07 / 07 / 2017

Vº Bº del Coordinador de Proyectos

Fdo.: David Contreras Bárcena Fecha: ……/ ……/ ……

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESINAS O MEMORIAS DE BACHILLERATO

1º. Declaración de la autoría y acreditación de la misma.

El autor D. Joaquín Liaño Díaz DECLARA ser el titular de los derechos de propiedad intelectual de la obra: MODELADO DE COMUNICACIONES ENTRE SERVICIOS INTERNOS Y APLICACIONES EXTERNAS MEDIANTE APIs EN EL ENTORNO BANCARIO, que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar "marcas de agua" o cualquier otro sistema de seguridad o de protección.
b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
e) Asignar por defecto a estos trabajos una licencia Creative Commons.
f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente).*

4º. Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

a) Que la Universidad identifique claramente su nombre como autor de la misma
b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
c) Solicitar la retirada de la obra del repositorio por causa justificada.
d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.

- El autor se compromete a:

a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e

intereses a causa de la cesión.

d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción de derechos derivada de las obras objeto de la cesión.

6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusive del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a 8 de julio de 2017

ACEPTA

Fdo: Joaquín Liaño Díaz

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

GRADO EN INGENIERÍA TELEMÁTICA

MODELADO DE COMUNICACIONES ENTRE SERVICIOS INTERNOS Y APLICACIONES EXTERNAS MEDIANTE APIs EN EL ENTORNO BANCARIO

Autor: Joaquín Liaño Díaz

Director: Atilano Fernández-Pacheco Sánchez-Migallón

Madrid

Julio 2017

Agradecimientos

Ante todo quiero dar las gracias a mi director de proyecto, Atilano, gracias a quien he tenido la oportunidad de realizar este proyecto. Gracias especialmente por toda su dedicación y por las valiosas aportaciones realizadas.

Mi más sincero agradecimiento también a todas las personas dentro de BBVA que con su ayuda han contribuido a la realización del proyecto.

Y muy especialmente, toda mi gratitud a mi familia por su apoyo constante, no solamente durante estos últimos meses, sino durante toda mi formación.

Muchas gracias a todos.

MODELADO DE COMUNICACIONES ENTRE SERVICIOS INTERNOS Y APLICACIONES EXTERNAS MEDIANTE APIS EN EL ENTORNO BANCARIO

Autor: Liaño Díaz, Joaquín.
Director: Fernández-Pacheco Sánchez-Migallón, Atilano.
Entidad Colaboradora: Banco Bilbao Vizcaya Argentaria, S. A.

RESUMEN DEL PROYECTO

Este proyecto tiene como principal objetivo demostrar la viabilidad del proceso de 'apificación' dentro de las arquitecturas IT actuales. Para ello, se ha realizado un estudio acerca de los beneficios de la adopción de *API Management* en las empresas, se ha colaborado en el proceso de 'apificación' de tres servicios de *backend* de BBVA y se ha desarrollado una aplicación Android consumidora de las APIs desarrolladas.

Palabras clave: API, API management, REST, Android

1. Introducción

Este proyecto está centrado en las APIs (*application programming inerface*), que son herramientas de las que disponen los desarrolladores para poder comunicar sus aplicaciones con otros servicios de forma ágil y sencilla. Este concepto ha evolucionado en los últimos años, dándose cuenta muchas empresas de su enorme utilidad, hacia el llamado API *Management*, en el cual se aportan servicios de valor añadido a las APIs como librerías, plantillas y patrones para los desarrolladores, monitorización de la ejecución de los servicios, configuración de políticas sobre gestión de eventos y errores, y configuración de políticas de seguridad.

2. Definición del proyecto

El proyecto persigue los siguientes objetivos:

- Realización de un estudio acerca de la solución de API *Management* adoptada dentro del departamento CIB de BBVA (donde se ha realizado el proyecto). Cuál es el ciclo de vida que siguen las APIs, qué actores intervienen y qué funciones tienen, y de qué se manera se securizan los servicios.
- Este estudio se aterrizará en el caso real de la apificación de 3 servicios de *backend* de BBVA, que se pretenden exponer como APIs *Restful*. Se ha adaptado la estructura de datos que devuelven los servicios de *backend* (XML con esquema FIX) para generar un JSON con esquema del estándar ISO 20022 (propuesto por la EBA, *european banking authority*) que devuelva la API.
- Desarrollo de una App Android consumidora de estos 3 servicios apificados que muestre la información devuelta por la API de manera visual para el usuario.

El objetivo último del proyecto consiste en estudiar la viabilidad de la solución de API Management para los requisitos de hoy

3. Descripción del sistema

La siguiente imagen muestra un esquema de la arquitectura del proyecto.

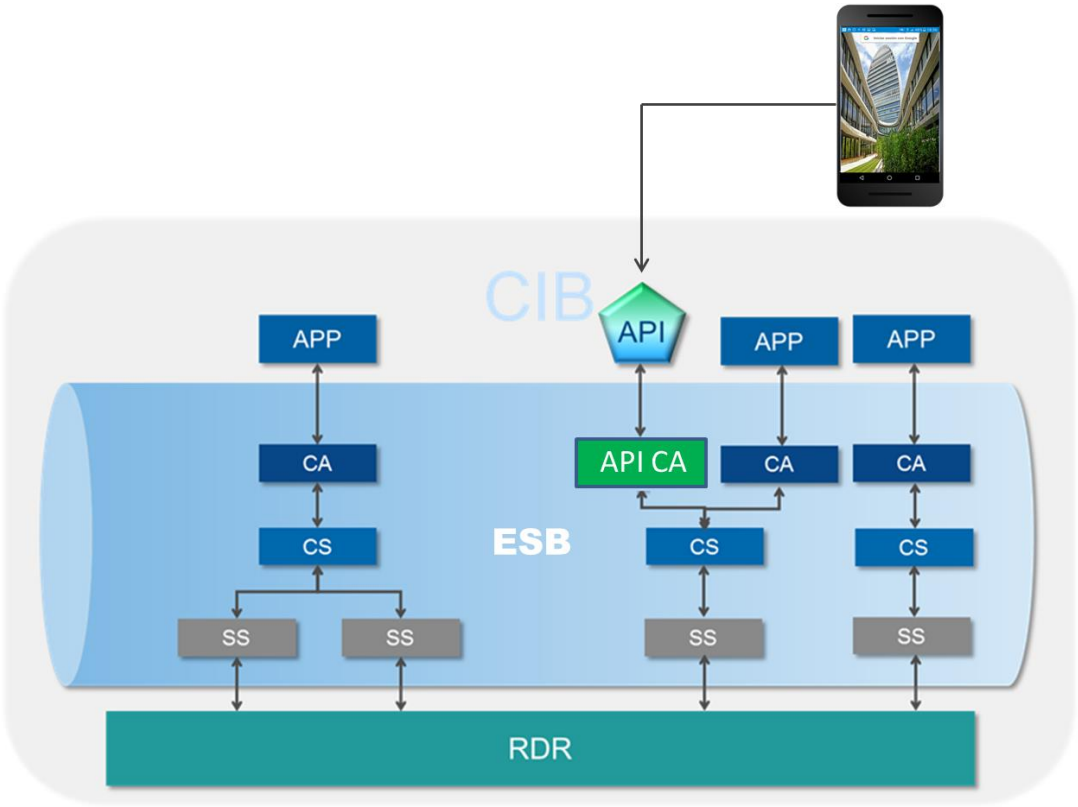


Ilustración 1 – Arquitectura del sistema

En la parte inferior del esquema se encuentra el sistema RDR, un sistema de *backend* en el que se encuentran los tres servicios apificados que se explican a continuación.

- Servicio de consulta de información de contrapartidas: Este servicio devuelve información (nombre de la entidad o persona, dirección fiscal, códigos bancarios, etc…) acerca de contrapartidas asociadas a movimientos de negocio efectuados por miembros del grupo BBVA.
- Servicio de consulta de jerarquía interna: devuelve todas las entidades, contrapartidas, grupos, etc... Asociados a una contrapartida conocida según la estructura interna de BBVA
- Servicio de consulta de calendarios: devuelve todas las fechas que son festivos o fines de semana en un calendario indicado mediante un código identificador del mismo, comprendidas en el rango de fechas que se indique.

Estos servicios se encuentran conectados al bus de conexiones del departamento CIB, el cual consiste en un sistema que permite la integración de servicios utilizando tecnología *middleware*, la cual incluía JMS, XML, Fix…

El sistema desarrollado ha consistido en presentar estos servicios mediante servicios REST (consumibles mediante simples peticiones HTTP) que devuelven la información en formato JSON.

Por último se ha desarrollado una aplicación Android que tras autenticarse el usuario, muestra un menú con las tres APIs desarrolladas. En cada uno de los tres servicios, se mostrará una pantalla en la que el usuario introducirá los datos de consulta, se realizará la petición a la API y se mostrará una pantalla con la información recibida.

4. Resultados

Por un lado se han logrado generar los documentos reales de modelado de tres APIs, consulta de información de contrapartidas, consulta de jerarquía interna y consulta de calendarios, exponiendo unos sistemas como servicios RESTful presentando los datos en formato JSON. Estas APIs se encuentran operativas, aunque faltaría por implementar, entre otras características, un sistema de securización de los servicios.

Por otra parte, una vez desarrollada la aplicación móvil, se ha probado su funcionalidad realizando una serie de peticiones a las APIs desarrolladas. A continuación se muestra una imagen con las pantallas de introducción de datos de consulta y de presentación de la información de la respuesta del servicio de consulta de información de contrapartidas.

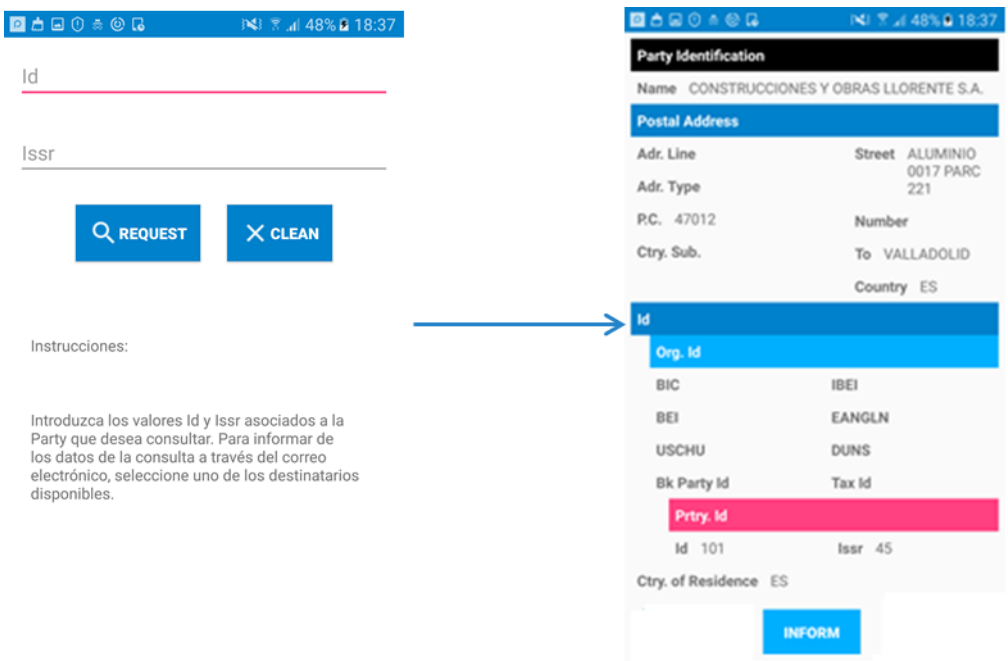


Ilustración 2 – Utilización del servicio de consulta de información de contrapartidas de la app móvil.

5. Conclusiones

Con la realización, en el tiempo de duración del proyecto (4 meses aproximadamente), de una prueba completa de apificación de unos servicios reales, cubriendo tanto la parte de desarrollo de los interfaces, como de una aplicación cliente, con un resultado funcional, las conclusiones son realmente positivas, ya que se han logrado demostrar algunos de los principales beneficios que ofrece el proceso de apificación: facilidad de integración con diferentes tecnologías (Android en este caso) y reducción del *time to market.*

COMMUNICATIONS MODELING BETWEEN INTERNAL SERVICES AND EXTERNAL APPLICATIONS USING APIS IN THE BANKING ENVIRONMENT

Author: Liaño Díaz, Joaquín.
Supervisor: Fernández-Pacheco Sánchez-Migallón, Atilano.
Collaborating Entity: Banco Bilbao Vizcaya Argentaria, S. A

ABSTRACT

This project chases the purpose of demonstrating the viability of the 'apification' process of IT architectures. For that purpose, it has been carried out a study on the advantages of adopting API Management by IT companies, it has been done a collaboration work in the process of 'APIfing' three BBVA backend services and it has been developed an Android application consumer of the mentioned APIs.

Keywords: API, API management, REST, Android

1. Introduction

This project is mainly focused in APIs (application programming interface), which consist in tools that developers can use for communicating their own applications with other services in an agile and simple manner. This concept has been developed in the last years, due to many companies which have realized of the enormous usefulness this elements can have, to a new concept called API Management, in which an added value is provided, for example libraries, templates and standards for developers, service execution monitoring, setting up events and errors management configuration and security policies.

2. Project definition

The project chases the following objectives:

- Make a study about the API Management solution implemented in the BBVA CIB department (in which the project has taken place). The study will contain topics such as the API life cycle, which actors take part in it and which kind of tasks they have, and how services get secured.

- This study will be followed with the apification of three BBVA backend services, which will be displayed as RESTful APIs. The structure of data which return the backend services (XML with a FiX schema) has been adapted for generating a JSON object using a ISO 20022 schema (proposed by the EBA) which is returned by the API.

- Development of an Android app which will make use of the 3 'apificated' services, showing the information returned by the API, in a visual manner for the user.

The main objective of the project consists in verifying the viability of the API Management solution for the actual requirements.

3. System description

The following image shows the project architecture scheme.

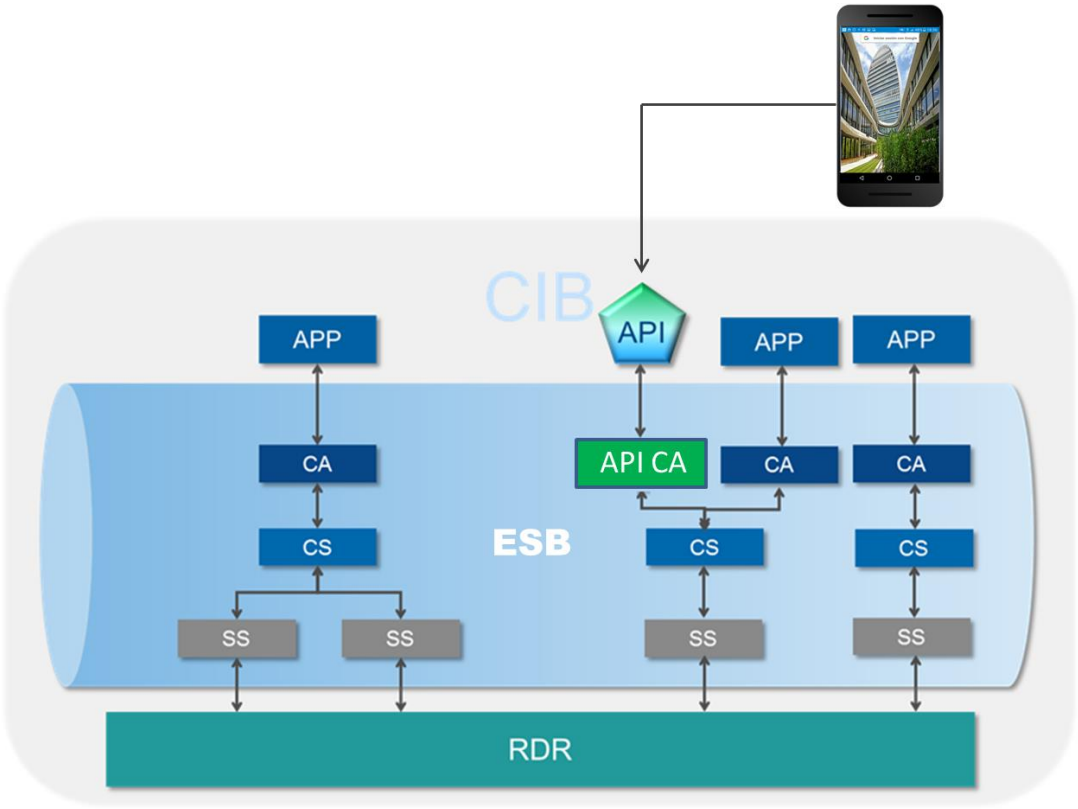


Illustration 1 – System architecture

At the bottom of the illustration it is placed the RDR system, a backend system in which are placed the three apificated services, which are explained below.

- Parties information request service: This service returns information (name of entity or person, postal address, bank codes…) about financial parties linked to financial movements done by BBVA members.
- Internal structure request service: Returns all entities, financial parties, groups… linked to a known party according to the BBVA internal structure.
- Calendars request service: Returns all dates which are weekends or holidays in a calendar specified by its identifier code, between a range of given dates.

These services are connected to the BBVA CIB enterprise service bus, which consists of a system which uses middleware technology, including JMS, XML, FiX…, for integrating services inside the department.

The system built consists of exposing these services as RESTful services (usable through HTTP requests) which return the information in a JSON format.

Finally, it has been developed an Android app which, once the user has been authenticated, it shows a menu for using the three developed APIs. Each of the three services will display a screen in which the user will introduce the parameters for sending the request to the API, after the request is done, a new screen will show the resulting information.

4. Results

First, it has been achieved to generate the modelling documents of the three APIs, parties information request, internal structure request and calendars request, implementing them as RESTful services which return the information requested as JSON objects. These APIs are operative at the end of the realization of the project, although some kind of securing system is missing in the model.

Once the Android app has been developed, its functionality has been proved by sending a series of requests to the developed APIs. The following image shows the two screens of request parameters input and the representation of the received data in a parties information service request.

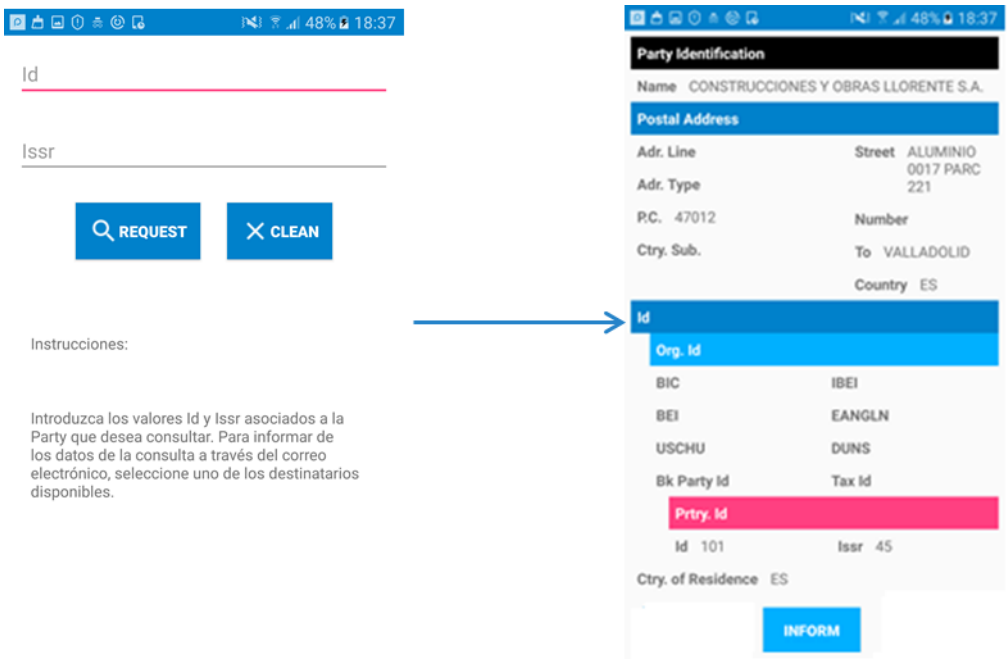


Illustration 2 – Usage result of the parties information service of the Android app

5. Conclusions

The realization of a complete apification test of three real backend services in the time the project has taken place (an approximately duration of 4 months), covering both parts of the API value chain, the development of the interfaces and the client app, obtaining a functional result, brings really positive conclusions, since some of the principal benefits the apification process brings have been proven, including the facility of integrating systems with different technologies (Android in this work) and the reduction of the time to market.

Índice de la memoria

Índice de figuras

Índice de tablas

1. INTRODUCCIÓN

En las tres últimas décadas, con la aparición de Internet y el consiguiente despliegue masivo de redes en todos los ámbitos ha quedado totalmente claro que la verdadera potencia de cualquier tecnología radica plenamente en su capacidad de comunicación con otras tecnologías, este enfoque hacia la compartición de datos ha provocado un crecimiento vertiginoso de una gran cantidad de tecnologías. Desde entonces, prácticamente la totalidad de las tecnologías desarrolladas se han centrado de una manera u otra en aprovechar la potencia de este nuevo paradigma centrado en la comunicaciones, desde la aparición de los lenguajes orientados a objetos cuya base radica en el traspaso de mensajes entre diferentes entidades y la invocación de métodos y procedimientos remotos; a las últimas tecnologías que prometen resultar más disruptivas como por ejemplo *big data* y el internet de las cosas, que tienen en común que ambas tienen su fundamento en el intercambio y procesamiento de cantidades masivas de datos.

El desarrollo de estas tecnologías ha seguido un proceso común o habitual por el cual, en el momento de su aparición, cada tecnología desarrolla su propia manera de compartir información y sus propios estándares, hasta que llega un momento en el que la cantidad de diferentes maneras de tratar la información se hace demasiado grande, además, la creciente complejidad de los sistemas y tecnologías va dificultando cada vez más el desarrollo desde cero de estas formas de manejar mensajes, por lo que va surgiendo la necesidad de crear estándares y plantillas que permitan el desarrollo de funcionalidad a más alto nivel, y por tanto, cada vez de mayor potencia.

Un proceso similar ha ido ocurriendo dentro de las propias empresas, en las cuales, gracias al desarrollo de las redes, se han ido sustituyendo progresivamente los grandes sistemas *mainframe* en favor de la partición de la funcionalidad en pequeños sistemas y la aparición de las llamadas arquitecturas orientadas a servicios, en las cuales, cada vez son más los departamentos dentro de las propias empresas que desarrollan sus propias aplicaciones, las

cuales a su vez van aumentando en cuanto a complejidad y especificación en sus propias tareas.

Ante esta fragmentación de la funcionalidad en diferentes aplicaciones, surge una necesidad de integración entre ellas prácticamente inabarcable con el enfoque punto a punto. Esto ha dado lugar al surgimiento de arquitecturas de IT caóticas en muchas empresas debido al creciente número de conexiones sin estándares bien definidos, que conducen a una pérdida de eficiencia cada vez más elevada dentro de las empresas.

Esta necesidad de comunicar tantos sistemas entre ellos, no solamente dentro de la propia empresa, sino incluso con sistemas externos, ha llevado a destinar una gran cantidad de recursos en desarrollar sistemas de integración que permitan a los distintos equipos poder centrarse en la propia funcionalidad de sus sistemas sin necesidad de preocuparse por aspectos relacionados con la comunicación entre procesos.

De estos sistemas de integración surgen las APIs (*application programming inerface*), que son herramientas de las que disponen los desarrolladores para poder comunicar sus aplicaciones con otros servicios de forma ágil y sencilla. El concepto de API se ha ido desarrollando en los últimos años, dándose cuenta muchas empresas de su enorme utilidad, evolucionando hacia la gestión de la API, en la cual se aportan servicios de valor añadido a la API como librerías, plantillas y patrones para los desarrolladores, monitorización de la ejecución de los servicios, configuración de políticas sobre gestión de eventos y errores, gestión de errores y servicios de auditoría.

El presente trabajo estará centrado en el desarrollo y la gestión de plataformas API dentro de las arquitecturas IT. En primer lugar se realizará un estudio de la evolución de estos sistemas, las razones que explican la súbita importancia que las empresas le han dado a su desarrollo, las implicaciones que tendrá este desarrollo y las posibles tendencias futuras. A continuación, siempre dentro de este marco y dentro del sector de arquitectura de integración del área CIB de BBVA, en el cual estoy realizando un periodo de prácticas y que explicaré más adelante en qué consiste, se pretende desarrollar una serie de piezas que permitan realizar llamadas API a unos servicios de *backend* del banco. Por último, y con el

fin de poder mostrar de una manera gráfica y visual la utilidad de la apificación de estos servicios, se desarrollará una aplicación en Android que llame a estos servicios vía API y muestre los resultados de la petición con alguna utilidad práctica.

1.1 MOTIVACIÓN DEL PROYECTO

La motivación del proyecto surge en primer lugar debido a que dada la novedad del concepto de gestión de la API, aún no está del todo claro a qué hace referencia y cabe realizar en este momento una investigación acerca de qué aspectos pueden resultar más beneficiosos y cuáles no. Este tipo de implementación cobra mayor relevancia dentro del mundo empresarial, en el cual, el desarrollo de servicios de integración que provean APIs puede agilizar enormemente los procesos de negocio de cualquier tipo de empresa. El interés es aún mayor estando dentro de una entidad financiera del volumen de BBVA, donde debido a una nueva regulación europea, que se explicará más adelante, se deberá desarrollar y ofertar una infraestructura de API pública de cara a ofertar los servicios requeridos por la normativa europea, cumpliendo las exigencias de seguridad requeridas para esta casuística, mientras que se busca la forma de rentabilizar y encontrar nuevas oportunidades dentro de este cambio de modelo de negocio.

En cuanto al aspecto técnico, la motivación reside en la necesidad de superar el escenario ya comentado de las conexiones punto a punto en el ámbito BBVA CIB, para ello, el área de arquitectura técnica de la entidad es el responsable del desarrollo e implementación de toda la infraestructura de integración de la empresa. Dentro de este proceso, se pretende proveer a un conjunto de servicios internos del departamento de RDR, un sistema propio del banco que proporciona datos maestros, con una API para facilitar y agilizar las comunicaciones con otros sistemas, dotándola de los servicios de valor añadido anteriormente comentados. Este proyecto, por otra parte, tiene la vista puesta en la posibilidad de ofertar algunos de estos servicios públicamente, buscando un beneficio de ello, tratándose de adaptar la entidad al nuevo modelo de pagos buscado por la Unión Europea.

1.2 INTRODUCCIÓN A LAS API

Es comentado frecuentemente como el volumen de información procesado en el mundo de las IT se ha visto incrementado exponencialmente en los últimos años, sin embargo, un factor igualmente importante que está dificultando enormemente el tratamiento de esta información es la multiplicación de los diferentes formatos en los que se presenta. Los recursos web ya no se consumen únicamente desde el navegador de un pc, sino que hoy en día contamos con *smart phones*, *tablets*, redes sociales, servicios *cloud*, IoT, etc… Es decir, nos encontramos ante un aumento muy elevado de los modos de consumo de servicios en muy poco tiempo, lo cual obliga a las empresas que ofrecen estos servicios a adaptarse muy rápidamente para poder cubrir este tipo de demanda tan heterogénea.

De cara al público, el modelo de negocio más visible es el de las API públicas, es decir, aquellas que las empresas ponen a disposición de los desarrolladores ajenos a la propia empresa. Las empresas tecnológicas líderes como Google, Twitter y Facebook han invertido grandes cantidades de recursos en desarrollar plataformas de integración capaces de generar beneficios para todo tipo de desarrolladores y empresas externas que quieran aprovechar los potentes servicios que ofrecen estas empresas, y a su vez, un beneficio a ellos mismos, ya que el despliegue de estas plataformas les ha permitido extender su negocio a una gran cantidad de diversas áreas.

El modelo de las API pública no sólo ha beneficiado a las empresas digitales como las mencionadas, sino que han sido muchas empresas en muy diversos sectores las que han sabido beneficiarse del mismo, como plataformas de *ecommerce* como Amazon y Ebay que se han adueñado del sector *retail*, o empresas como Uber y Airbnb que han puesto de moda el concepto de economía colaborativa también gracias a este modelo. [2]

Por otra parte nos encontramos con las API privadas, las cuales, si bien no han tenido tanta repercusión mediática, sí que tienen una gran importancia en los modelos de negocio de una gran cantidad de empresas en todo el mundo. Una API es privada cuando el dueño de los recursos o servicios que proporciona dicha API, solamente quiere que sean accesibles

para otros sistemas dentro de su propia empresa o para otras empresas con las que llega a un acuerdo para que puedan acceder a sus activos. En estas API cobra gran relevancia la seguridad.

El beneficio que busca la empresa al desarrollar APIs para sistemas propios es el de acabar con el escenario presentado en la introducción en el cual la gestión de todas las conexiones punto a punto se acaba volviendo insostenible. El objetivo primero que se busca, por tanto, es el de desacoplamiento de los servicios, es decir, establecer un contrato con el consumidor del servicio que establezca todo lo relacionado con la entrada y la salida del mismo, siendo posible cambiar la implementación interna del servicio, e incluso cambiar el propio servicio, siempre que se respete el contrato de entrada y salida.

1.3 ARQUITECTURA DE INTEGRACIÓN EN BBVA CIB

Este apartado pretende resumir los objetivos del departamento dentro del cual se ha realizado el periodo de prácticas en el cual se ha desarrollado este proyecto. Dentro de BVVA, el área CIB (Corporate & Investment Banking) está a cargo de las operaciones de banca de inversión, préstamos en un ámbito global y servicios transaccionales para clientes corporativos internacionales. Como resulta evidente, el desarrollo tecnológico capaz de soportar la realización efectiva de estas operaciones de elevada complejidad debe ser necesariamente muy grande y puntero.

Uno de los departamentos cuya labor resulta fundamental para el correcto desempeño de este sector es el de Arquitectura de Integración, cuya misión es la de definir la lógica de procesos de negocio y la infraestructura de IT. Para ello, este departamento se encarga de la orquestación de mensajes entre diferentes sistemas, para ello, son los encargados de la traducción de los mensajes a un modelo común de datos, del enrutamiento de los mensajes, de su gestión y de la configuración de políticas.

Arquitectura de Integración, por tanto, trabaja en torno a la integración orientada a datos y la integración orientada a servicios y aplicaciones. Para la primera ha desarrollado una

arquitectura llamada ETL (Extract, transform and load), añadiendo a su vez capacidades para auditoría, patrones de desarrollo y servicios comunes. Para la integración orientada a servicios y aplicaciones se ha desarrollado un bus de integración, un componente *middleware* diseñado para interconectar sistemas y aplicaciones desarrollados con diferentes tecnologías, transformando la información a un modelo común de datos con el que se puedan entender los distintos sistemas, más adelante dedicaré un párrafo a estudiar esta tecnología más en profundidad.

A partir de esta herramienta, el departamento tiene como objetivo la implementación de un *framework* de integración, que ofrezca escalabilidad, gracias al desacople entre sistemas y el empleo de un modelo común de datos; robustez, ya que este modelo busca asegurar mantenimiento y tratamiento de errores; y reducir la complejidad técnica a los equipos funcionales aportando plantillas, patrones y procedimientos de manera que estos puedan centrarse únicamente en desarrollar la funcionalidad asociada a su ámbito sin necesidad de preocuparse por aspectos de compatibilidad e integración. [3]

Vistas las responsabilidades y objetivos del departamento, parece lógico que sean ellos los encargados de diseñar e implantar la arquitectura de la plataforma API que pretende implantar el banco, y será en este ámbito en el cual se desarrollará el contenido del presente proyecto.

2. DESCRIPCIÓN DE LAS TECNOLOGÍAS

2.1 ATOM Y API WORKBENCH

Las APIs se documentarán mediante el lenguaje RAML 1.0. RAML (RESTful API Modeling Lenguage) está basado en YAML 1.2. Ha sido diseñado con el objetivo de definir APIs REST y se apoya en la notación *Markdown* para generar la documentación en formato HTML.

Los ficheros RAML se editarán haciendo uso de dos herramientas, Atom y API Workbench. Atom es un editor de texto orientado al desarrollo de código, con la ventaja de que se le pueden añadir funcionalidades y complementos que permiten configurar un espacio de trabajo que ofrece mucha más potencia que otros editores más convencionales, pero sin llegar al nivel de complejidad de un IDE de programación, lo que hace que sea una herramienta muy flexible.

API Workbench es una plataforma que proporciona un entorno de desarrollo para diseñar, construir y documentar APIs RESTful, al ser compatible con RAML, permite gestionar fácilmente todo el ciclo de vida de la API. La interfaz ofrecida por esta herramienta permite agilizar notablemente el desarrollo con este lenguaje al proporcionar el añadir recursos, métodos, su documentación y ejemplos de una manera muy ágil, esto permite al desarrollador centrarse en el diseño sin tener que preocuparse excesivamente por el aspecto más técnico.

Adicionalmente se instalará un paquete para validaciones semánticas que ayudará a escribir sintácticamente bien los archivos RAML.

2.2 ANDROID STUDIO

El entorno elegido para desarrollar las aplicaciones Android ha sido Android Studio, en concreto se hará uso de la versión 2.3.1. Se ha hecho esta elección, en vez de, por ejemplo, utilizar el IDE de eclipse, ya que, además de ser el entorno oficial para el desarrollo de aplicaciones Android, esta herramienta ofrece un potente editor de códigos y una serie de funciones que facilitan en gran medida el desarrollo de aplicaciones de este tipo. Android Studio está basado en IntelliJ IDEA y su sistema de compilación está basado en Gradle, una herramienta similar a Ant y Maven que automatiza la construcción de proyectos, simplificando al desarrollador tareas como la compilación, *testing*, empaquetado y la gestión de las dependencias y repositorios. [4]

El proyecto se desarrollará en una máquina que hace uso del sistema operativo Windows 7, sin embargo, Linux ofrece mucha más libertad y flexibilidad que será necesaria para el desarrollo de las aplicaciones que se pretende. Por tanto, se ha utilizado una máquina virtual para simular el entorno Linux, para ello se ha hecho uso del programa VirtualBox, una herramienta perteneciente a Oracle que permite simular "sistemas invitados" dentro de otro "sistema anfitrión", en nuestro caso Linux sobre Windows 7. Dentro de este entorno

virtual es donde se ha instalado la versión de Android Studio y se desarrollarán las aplicaciones.

2.3 SOAPUI Y RAML PLUGIN

Como se explicará más adelante, la aplicación Android desarrollada se conectará a las API desarrolladas que estarán ubicadas en unos servidores de BBVA, sin embargo, antes de invocar estos servicios resulta necesario poder disponer de servicios de prueba que permitan realizar las pruebas necesarias durante el desarrollo de la aplicación. Para ello se ha utilizado el programa SoapUI.

SoapUI es un programa *open-source* para realizar tests de servicios web tanto SOA como REST. Su funcionalidad permite simular estos servicios de manera que se pueden desplegar recursos web estáticos en el propio pc, pudiendo realizar pruebas de manera muy cómoda. SoapUI cubre además servicios JMS, AMF y puede realizar llamadas HTTP y JDBC.

Todas los servicios desarrollados en este proyecto devolverán datos en formato JSON sobre HTTP, por lo que para generar los servicios de prueba de manera sencilla se ha utilizado también un *plugin* de SmartBear para el programa que crea automáticamente una respuesta JSON estática de prueba a partir de un fichero RAML con la información del servicio. [5]

2.4 GIT

El presente proyecto no se ha llevado a cabo de manera individual, sino que la parte aquí desarrollada se engloba dentro de un proyecto en el que están involucradas varias personas de diferentes departamentos, para poder compartir y ensamblar el trabajo de las distintas partes de una manera eficiente se ha utilizado el programa Git.

Este programa es un sistema de control de versiones distribuido (DVCS), que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que se puedan recuperar versiones específicas más adelante. Git permite revertir archivos a un estado anterior, revertir el proyecto entero a un estado anterior, comparar cambios a lo largo del tiempo, ver quién modificó por última vez algo que puede estar causando un problema, quién introdujo un error y cuándo.

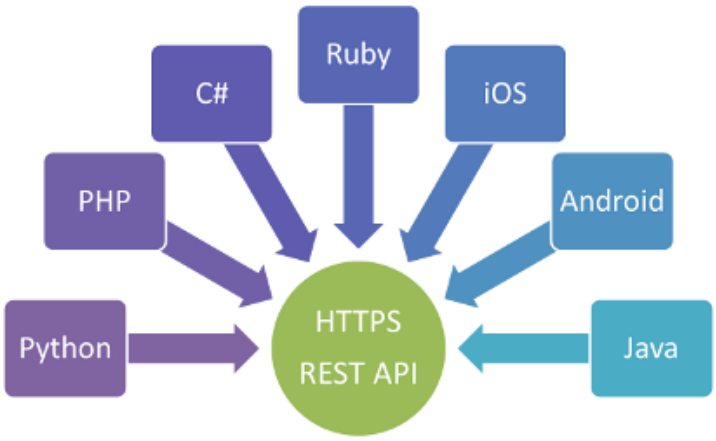
Dentro de un proyecto en Git se trabaja en tres secciones diferentes: el directorio de Git, que es donde el programa almacena los metadatos y la base de datos de objetos del proyecto, que es lo que se copia al clonar un repositorio desde otro ordenador; el directorio de trabajo, que es una copia de una versión del proyecto en local; y el área de preparación, que es un archivo almacenado en el local que guarda la información que se va a enviar al directorio de Git en la próxima confirmación.

2.5 ISO 20022

El estándar internacional ISO 20022 (*Universal Financial Industry message scheme*) es un *framework* propuesto por el ISO para el desarrollo de mensajes financieros. Este *framework* no ofrece un esquema fijo para la creación de los mensajes como tal, sino que describe una plataforma común para el desarrollo de mensajes estandarizados. Para ello hace uso de una metodología de modelado basada en UML para esquematizar los flujos y procesos asociados al área financiera. El estándar provee además un diccionario de términos habitualmente empleados en comunicaciones entre entidades financieras, y un conjunto de reglas para convertir los mensajes descritos en UML a esquemas XML. [6]

Los mensajes ISO 20022 se han diseñado con independencia del protocolo de transporte que se utilice y no incluyen convenciones propias para el mensaje de transporte. Los usuarios de estos mensajes tienen libertad para definir el transporte del mensaje, de acuerdo con los estándares y prácticas de la red o de la comunidad donde se implante su utilización. [7]

2.6 SERVICIOS RESTFUL



REST (*REpresentational State Transfer*) es un modelo de arquitectura para definir interfaces entre servicios basado en HTTP para obtener datos o realizar acciones sobre los mismos. Estos datos pueden estar en todos los formatos posibles, típicamente XML y JSON, aunque la tendencia actual es utilizar JSON. Al estar basado en HTTP, la principal característica de los servicios REST radica en el hecho de que todas las comunicaciones se realizan sin estado, ni por parte del cliente, ni del servidor, si bien es cierto que algunas implantaciones incorporan memorias cache. Este modelo está experimentando un auge muy importante en los últimos años, desbancando a otros protocolos como SOAP, convirtiéndose en la opción predominante a la hora de construir APIs para servicios de Internet, debido a su sencillez de implantación y la escalabilidad que permite.

3. ESTADO DE LA CUESTIÓN

En los últimos años, cada vez es mayor el número de empresas que se está dando cuenta de la necesidad que supone implementar una infraestructura API que cumpla con las necesidades cada vez mayores de flexibilidad, integración y escalabilidad que tienen, así como de las posibilidades y oportunidades que plantea. Estos sistemas, a medida que añaden nuevas y mejores funcionalidades, se vuelven cada vez más complejos, por lo que no todas las empresas se pueden permitir desarrollar sus propias soluciones, es por ello que desde hace algunos años, están apareciendo empresas que aportan soluciones a otras para este tipo de implantaciones, algunas de ellas muy nuevas, y otras líderes ya consolidados que están comenzando a ofrecer sus propias soluciones, tanto de APIs internas como externas. Entre ellas destacan empresas como ApiGee, 3scale de RedHat, Mashery de TIBCO, WSO2 API *Management*, grandes empresas como IBM y Microsoft también ofrecen soluciones de API *Management*.

También resulta necesario comentar la importancia de los servicios *web* desde hace algunos años, pero cada vez con mayor relevancia. Estos consisten en objetos cuya funcionalidad es accesible a través de protocolos de red y son la base de la funcionalidad de *bakend* de muchas arquitecturas. En concreto, en el departamento CIB, la funcionalidad interna se consume a través de *web services*, estos elementos hacen por tanto el rol de los recursos que se está tratando de apificar.

3.1 LA NUEVA DIRECTIVA EUROPEA DE SERVICIOS DE PAGO

Uno de los sectores que se está viendo transformado en mayor medida es el financiero, especialmente a raíz de una nueva regulación europea que está forzando a muchas entidades financieras a ofertar servicios de API pública, la nueva directiva europea de servicios de pago (PSD2), cuyo objetivo consiste en establecer un espacio único de servicios de pago dentro del continente, tratando de alcanzar una igualdad de condiciones y

derechos en los servicios ofrecidos en el mercado. Para ello, se obligará a los bancos a dar acceso libre a información sobre cuentas y servicios de pago a proveedores de los mismos, los cuales son terceros no vinculados con las entidades. A pesar de que la regulación no habla explícitamente de gestión de APIs, la única manera de lidiar con los requerimientos de seguridad, escalabilidad y monitorización que surgen en este nuevo modelo, es desplegar una infraestructura de API pública, la cual, a su vez, abre la posibilidad a las entidades financieras de beneficiarse de un contacto directo con agentes como proveedores de servicios, servicios de *ecommerce* y minoristas, con los cuales, hasta este momento, habían tenido un trato mucho menos directo. [9]

En palabras de la Comisión Europea "las nuevas directrices protegerán mejor a los consumidores a la hora de efectuar pagos, promoverá el uso y el desarrollo de formas innovadoras de realizar pagos *online* y móviles y hará que los servicios de pago europeos sean más seguros". [8]

Aunque habrá que ver si realmente se acaban cumpliendo las expectativas de la Comisión en los próximos años, en lo que sí parece estar todo el mundo de acuerdo es en que esta directiva provocará grandes cambios en el sector de los pagos electrónicos. La primera consecuencia que tendrá será la aparición de nuevos actores, los llamados *Third Party Providers* o TPPs, que serán las entidades encargadas de aprovechar la apertura de la información de cuentas y los servicios de los bancos para ofrecer nuevos servicios a los usuarios. Habrá dos tipos de TPP, los proveedores de servicios de iniciación de pagos o PISP y los proveedores de servicios de información sobre cuentas o AISP.

Los proveedores de inicio de pago son servicios que inician procesos de pago al vendedor, desde una plataforma de comercio online y previa autorización del cliente, accediendo directamente a la cuenta bancaria del comprador, lo cual deberán autorizar los bancos sin ningún tipo de discriminación. Para ello crearán un puente software entre el comercio y el banco. Este tipo de servicio eliminará la necesidad de intermediación por parte de agentes como plataformas de tarjetas *online*.

Esta idea se ilustra en las siguientes dos imágenes. La primera corresponde a la manera de efectuar pagos *online* actualmente, donde como se puede ver, mediante peticiones HTTP generalmente, los datos de las cuentas de los clientes, donde se incluyen datos sensibles como datos personales, pines de cuentas…, siguen un flujo largo en el que intervienen muchos actores como el vendedor, proveedor de la tarjeta…, lo cual puede generar riesgos para la seguridad de los datos.

En los pagos efectuados tras la apificación, sin embargo, el flujo de datos es menor, reduciendo el tiempo en que estos datos están expuestos, además de que una buena gestión de las APIs permitirá mayor robustez en dicho intercambio de datos.

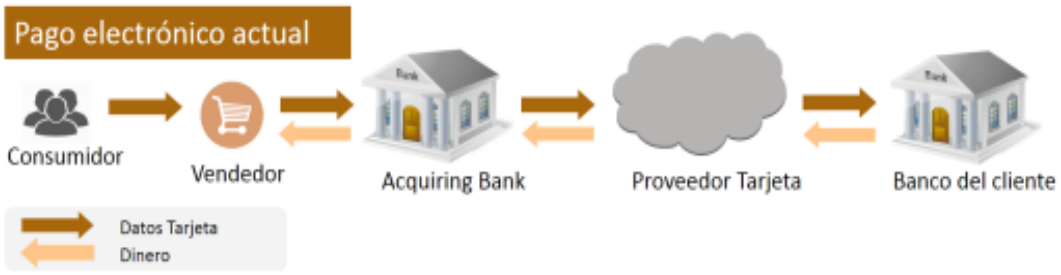


Figura 1 – Modelo de pagos online actualmente [9]

Figura 2 – Modelo de pagos después de la PSD2 [9]

Los AISP consisten en servicios *online* que facilitarán información sobre una o varias cuentas de pago de las que es titular el usuario del servicio de pago, bien en otro proveedor de servicios de pago, o bien en varios. La ventaja de la que se pueden aprovechar este tipo de aplicaciones para ofrecer a sus usuarios es que podrán recopilar la información de fondos y cuentas que en muchas ocasiones estos tienen dispersas en varias entidades o incluso en la misma, pero que los bancos siempre han sido reticentes a ofrecerla de una manera conjunta y ordenada, y ofrecerla de esta manera, además de poder analizarla, ofreciendo recomendaciones a los usuarios.

Como se ha comentado, en el texto de la directiva PSD2 no se habla en ningún momento de que las entidades financieras estén obligadas a ofrecer los servicios exigidos mediante infraestructuras de API pública, sin embargo, son varias las razones que hacen pensar que esta es la única manera que tendrán para cumplir la normativa de manera eficiente y sin quedarse atrás en el mercado. Una de las principales razones, y quizá la que más preocupa a los usuarios finales que utilizarán las aplicaciones que accedan a los servicios abiertos por los bancos, es el tema de la seguridad. Actualmente, cuando se realiza un pago *online*, o una consulta, uno de los principales problemas que surgen es que para que se autentique el usuario en el momento de realizar la acción, éste debe enviar sus credenciales bancarias al proveedor de tarjetas, el cual a su vez deberá enviarlas al banco en el que el usuario tiene la cuenta, y realizar este proceso cada vez que se quiera realizar cualquier acción, aunque sea simplemente de consulta. A pesar de que en los últimos años los sistemas de seguridad han evolucionado notablemente, en temas de cifrado de datos por ejemplo, la aparición de https, sigue existiendo el riesgo de que los datos sean interceptados durante la comunicación, sufriendo ataques de tipo *man-in-the-middle*. Con el objeto de mitigar este riesgo se han desarrollado ciertos estándares y protocolos para securizar los procesos de autenticación y autenticación de las comunicaciones vía APIs, especialmente en llamadas a servicios REST, este hecho ha posicionado este tipo de comunicaciones como la opción idónea para implantar servicios en los que se manejan datos sensibles del tipo de información de cuentas bancarias.

Otra de las razones que forzará la implementación de servicios de API pública es la escalabilidad. La normativa dice que los bancos deberán permitir el acceso a sus servicios a cualquier servicio externo, del tipo que sea, una vez que hayan obtenido el consentimiento del propietario de la cuenta, sin ningún tipo de discriminación. La única manera a día de soportar la comunicación con tal cantidad de entidades será desplegando una plataforma API con su capa de gestión.

Otra razón de peso es la económica, la inversión que la nueva directiva obligará a realizar a las entidades financieras para poder cumplimentarla, por no mencionar cómo afectará a sus ingresos la aparición de todo tipo de competidores que podrán beneficiarse de sus servicios de manera gratuita obligará a los bancos a encontrar la manera de sacar un beneficio del cumplimiento de esta normativa, y la única solución posible a este problema parece que está en la capa de la gestión de las APIs.

En su afán por posicionarse como un auténtico líder en el marco de la transformación digital del mundo financiero, BBVA está realizando un gran esfuerzo para, no solamente cumplir con las exigencias de la Unión Europea, sino aprovechar el gran cambio que se espera que sacuda al mercado de los pagos electrónicos en los próximos años. Para ello, ya está disponible el BBVA API *market* [10], una plataforma web que ofrece en estos momentos 7 servicios con conexión vía API pública para los clientes en España, y 4 para los de EEUU, mientras que la mayoría de bancos europeos aún se están preguntando cómo hacer frente a la normativa y están exigiendo a la Unión Europea pautas y estándares para construir los servicios necesarios.

El modelo de las APIs que ofrece BBVA es REST, buscando sencillez para acceder a sus servicios, además se ofrece también de manera gratuita una consola para realizar pruebas. La autenticación en las llamadas se gestiona empleando tokens en formato JSON, lo que supone una manera eficiente y segura de gestionar este aspecto en las llamadas a servicios REST, como se comentará más adelante. Según la criticidad de los servicios a llamar el modelo de autenticación puede estar basado en el modelo de dos patas, de tres patas de la utilización de *one-time-passwords*, lo que resulta muy práctico para los usuarios, ya que en

servicios de consulta de datos, por ejemplo, basta con hacer una petición a un servidor de autenticación y recibir un *token* para obtener los servicios, sin embargo, si se pretende, por ejemplo, realizar una transferencia en nombre de un usuario, la seguridad es más fuerte y se debe pedir el consentimiento directo del usuario final en cada llamada para obtener una *one-time-password*.

El objetivo que tienen estos servicios es en última instancia el de aportar facilidades y nuevas experiencias al usuario final con respecto a los pagos en la red. Los servicios ofrecidos a día de hoy en el API market son:

- *Customers*: consiste en una API que provee información personal relativa a un cliente BBVA, tal como nombre, fecha de nacimiento, sexo, email, dirección fiscal, documento de identidad y teléfono, de modo que se sustituyan los *logins* que hay que rellenar en muchas aplicaciones por una simple validación.
- *Accounts*: lista las cuentas de un cliente, verifica la titularidad, comprueba el saldo y recupera el histórico de sus movimientos, permitiendo por ejemplo, consultar el saldo en tiempo real para confirmar una transferencia, evitando complicaciones de esta manera.
- *Cards*: permite usar datos de tarjetas de usuarios que hayan autorizado su uso, agilizando las compras de los clientes al recuperar de manera sencilla su PAN, CVV y fecha de caducidad.
- *Payments*: permite realizar traspasos y transferencias nacionales e internacionales, de manera segura gracias al segundo factor de autorización según el cual el cliente debe confirmar la operación a través de un código SMS que llega a su móvil.
- *PayStats*: ofrece estadísticas agregadas de transacciones realizadas con tarjetas BBVA que permite conocer los hábitos de consumo de los clientes.
- *Loans*: permite conocer si tus clientes tienen o no un préstamo preconcedido en BBVA y las condiciones del mismo.
- *Notifications*: permite recibir información enriquecida sobre diferentes eventos de la operativa bancario de los usuarios de una aplicación en tiempo real.

Como se ha comentado, este tipo de servicios por parte de los bancos se han empezado a ofrecer recientemente, es por ello que aún existen pocas aplicaciones que se sirvan de los mismos para aportar algún tipo de servicio a sus clientes. Sin embargo han aparecido ya algunos ejemplos con cierta repercusión, el caso más notable en España es el de Fintonic, aplicación que ofrece a sus clientes la información de todas sus cuentas de manera organizada en un único servicio, además ofrece servicios de valor añadido a partir de esta información, como aportar recomendaciones al usuario para rentabilizar los ahorros o dónde invertir. [11]

Existen además otras aplicaciones que han surgido recientemente que proporcionan servicios para ordenar gastos, fijar objetivos de ahorro y, en definida, permitir controlar las finanzas personales de una manera más rápida y sencilla de lo que era hace poco tiempo atrás. Entre ellas destacan apps como Myvalue, Money Wiz, Toshl Finance y Whallet. En cuanto a los llamados proveedores de servicios de pago en Europa los servicios más conocidos son Trustly y Sofort, el primero, por ejemplo, define así su funcionamiento: "Seleccione su banco e inicie sesión con sus códigos de acceso habituales a través de una conexión cifrada y segura. Elija la cuenta desde la cual desea pagar (por ejemplo, su cuenta de ahorros o cuenta corriente). Verifique su compra." [12]

La ventaja del presente proyecto consiste en que, a diferencia de las aplicaciones analizadas, las cuales tienen que obtener los datos de los servicios que los bancos ponen a su disposición, en nuestro caso, tenemos la oportunidad de desarrollar una aplicación que se conecte directamente a servicios de uso interno dentro del propio banco, pudiendo ofrecer de esta manera una funcionalidad diferente a la ofrecida hasta ahora por este tipo de aplicaciones.

3.2 COMPONENTES DE LA ARQUITECTURA CIB QUE INTERVIENEN EN EL SISTEMA DESARROLLADO

En este apartado se pretende dar una visión de los diferentes sistemas y componentes que intervienen en el sistema desarrollado.

3.2.1 PLATAFORMA RDR

RDR es un sistema de la arquitectura interna de *backend* de BBVA que tiene desplegados una serie de servicios de consulta, alta y modificación de datos maestros, estos datos corresponden a detalles de pagos realizados por la entidad, contrapartidas financieras, la estructura interna de la entidad, etc…

A pesar de que a día de hoy hay más servicios de RDR desplegados, a continuación se detalla la funcionalidad únicamente de los servicios sobre los que se va a trabajar en el proyecto, aunque como se comentará en el apartado de trabajos futuros, estos son solamente una pequeña parte de los disponibles y en función de los resultados obtenidos se podría decidir realizar este trabajo con algunos de los restantes.

- Consulta de información de contrapartidas:

  Este servicio devuelve información acerca de contrapartidas asociadas a movimientos de negocio efectuados por miembros del grupo BBVA. Entendemos contrapartida como el otro actor involucrado en cualquier movimiento financiero respecto al actor principal que se considere, en este caso las entidades pertenecientes al banco. Estos actores pueden ser de muchos tipos: personas físicas, instituciones financieras, empresas, etc.., por lo que una estructura de datos en la que se pueda representar cualquier tipo de contrapartida no puede ser sencilla como se verá.

- Consulta de información de estructura interna

  Este servicio devuelve los datos referidos a la estructura interna de BBVA de sus contrapartidas. Al servicio se le introducen los datos que identifican a la contrapartida deseada y se indica el tipo de jerarquía (estructura superior, inferior, grupo, entidad…), el servicio devuelve como respuesta un *array* con los datos de las contrapartidas asociadas a la de entrada y el tipo de jerarquía.

- Consulta de información de calendarios

Este servicio devuelve todas las fechas que son festivos o fines de semana en la Bolsa de un calendario indicado mediante un código identificador del mismo, comprendidas en el rango de fechas que se indique. Los datos de entrada son un identificador del calendario y las fechas de inicio y de fin del rango, y los datos devueltos serán un *array* con los valores de las fechas indicando su tipo (vacaciones, fin de semana u otro tipo de día de no trabajo). El valor de este servicio se entiende en el contexto de un grupo grande como es BBVA, el cual tiene presencia en muchos países de muy diferentes partes del mundo, con diferentes calendarios laborales, y en un mundo como es el financiero resulta de gran importancia tener un acceso rápido a información acerca de los días laborables de cualquier país o región.

Los datos de estos servicios se encuentran en formato FIX, (*Financial Information eXchange*), que consiste en un protocolo de intercambio de información financiera. Este protocolo se compone de una serie de especificaciones de mensajes para comunicaciones financieras. [21]

En concreto, los mensajes se intercambian mediante FIXML, que consiste en un sistema para intercambiar mensajes en formato XML utilizando el lenguaje FIX para crear estos mensajes. [22]

3.2.2 GESTOR DE NOTIFICACIONES

El Gestor de Notificaciones es una herramienta para la gestión y envío de notificaciones entre los sistemas de CIB. Su objetivo consiste en dotar a los distintos sistemas existentes en CIB de una pieza de arquitectura para la gestión y envío de notificaciones en diferentes formatos. Permite garantizar la homogeneidad en el formato de las diferentes notificaciones enviadas y realizar el seguimiento de entrega de las notificaciones.

Los objetivos que buscan cubrirse con esta herramienta serán los siguientes:

- Administración de plantillas: (Alta, Baja, Modificación y Consulta) que posteriormente pueden ser utilizadas para homogeneizar las notificaciones enviadas.
- Consulta estado notificaciones: Posibilidad de consultar el estado en cada momento de las notificaciones enviadas a otros sistemas.
- Envío de notificaciones: Se permite establecer planificaciones de envío (basadas en fecha inicio/fecha fin, numero de envíos o frecuencia) de las notificaciones.
- Recepción SMS: Las recepciones pueden deberse a confirmaciones de que una notificación SMS ha sido entregada correctamente o a SMS enviados por los clientes como respuesta a una notificación anterior.
- Gestión de Reintentos: Para cada aplicación solicitante y tipo de mensaje se puede establecer el número de veces que se debe reintentar el envío de una notificación en caso de error.
- Administración: Modulo a través del cual el propio aplicativo puede configurar los diferentes aspectos necesarios las los envíos de las notificaciones correspondientes.

El Gestor de Notificaciones tiene como base ser la herramienta mediante la cual el resto de aplicaciones de CIB realizarán el envío de notificaciones.

Los servicios del Gestor estarán expuestos en el ESB en diferentes vías de comunicación (MQ, JMS, *WebService*,…) basadas en las capacidades de las diferentes aplicaciones solicitantes.

3.2.3 EL BUS DE INTEGRACIÓN (ESB)

El bus de integración consiste en un conjunto de capacidades, implementadas mediante el uso de tecnología *middleware*, que permiten la integración de servicios dentro de una arquitectura orientada a servicios. Podría definirse como un conducto de información en donde diferentes sistemas desarrollados con diferentes tecnologías en distintos formatos comparten un entorno donde depositar y recoger su información para interactuar entre ellos.

La principal aportación que tiene este elemento para un sistema es el desacoplamiento del que dota a sus elementos, lo que quiere decir que se podrá tratar o modificar cada uno de ellos sin necesidad de modificar en absoluto el resto de elementos. Esta característica se consigue gracias a la utilización de tecnologías tales como JMS (Java *Message Service*), mientras que los datos suelen estar en formato XML.

Las principales características del ESB son:

- Orientación a Servicios: las aplicaciones se comunican a través de servicios reutilizables, previa definición de una serie de interfaces.
- Dirección por Mensajes: las aplicaciones envían mensajes a un destino concreto.
- Dirección a Eventos (EDA): similar a la dirección por mensajes pero, en este caso, cada aplicación genera y consume mensajes de manera independiente al resto de sistemas.

Uno de los principales elementos que conforman el bus son los adaptadores, que consisten en componentes que permiten la transformación de la información que utilizan los servicios que utilizan el bus. Estos adaptadores se conectan directamente con las aplicaciones de *backend*, transformando la información que comparten con otros sistemas a un formato común de datos, enrutando los mensajes y añadiendo otra funcionalidad como gestión de errores y securización. Estos componentes se basan en estándares como JCA (Java EE *Conector Architecture*).

Este componente se implanta en entornos con numerosos puntos de integración, requerimientos elevados de escalabilidad o mediación entre extremos (para transformación o securización de datos) o con múltiples protocolos de comunicación.

En el departamento CIB de BBVA, se utiliza *BusinessWorks* de TIBCO para la implantación del ESB, que cuenta con prestaciones de valor para la entidad como transformadores y adaptadores, que responden a numerosas necesidades de integración de la entidad y otra funcionalidad como permitir la transformación del mensaje durante el proceso de comunicación con diferentes objetivos (seguridad, regulaciones, perfilado…).

En cuanto al aspecto técnico, la implementación de la tecnología JMS se ha realizado mediante TIBCO EMS (*Enterprise Message Service*). Esta solución soporta los dos principales tipos de mensajería:

- Mensajería punto-punto. Una aplicación de origen envía un mensaje a un destino concreto, especificando una cola de mensajería destino dentro de su cabecera JMS. Se aplica en los proyectos en los que se debe asegurar la llegada al destinatario (muy común en procesos con requerimientos de baja latencia).
- Publicación-subscripción. El mensaje se envía a varias aplicaciones destino, que en este caso reciben el nombre de *topics*. Toda aplicación destino que quiera recibir los mensajes de este tipo debe subscribirse a dicho servicio.

El principal componente del bus de integración es TIBCO *BusinessWorks*, una herramienta que permite desarrollar procesos y realizar los mapeos necesarios de datos usando una interfaz gráfica, sin necesidad de programar a bajo nivel. Este proceso realiza una transformación de los datos de entrada a un formato conocido como MCD (Modelo Canónico de Datos). En el área CIB este modelo se conoce como FMM (*Financial Messaging Model*), y se compone de cuatro formatos de datos: FpML, FIXM, ISO 20022 y XML.

4. DEFINICIÓN DEL TRABAJO

4.1 JUSTIFICACIÓN

En el apartado anterior se ha tratado de explicar la envergadura que tendrá el cambio provocado por la nueva normativa europea y, en general, por el nuevo enfoque que se está tomando en las arquitecturas IT, especialmente en sectores como el de la banca, que es en el cual se enmarca este proyecto.

Para que este cambio experimentado por la banca se materialice, resulta necesario que las instituciones financieras realicen un gran esfuerzo, de manera que no se queden atrás dentro del proceso de digitalización que está experimentando prácticamente la totalidad de los ámbitos de negocio hoy en día y puedan ofrecer un servicio eficaz y adaptado a las necesidades del usuario de hoy en día, usuario ya acostumbrado a tener una cantidad de información y de servicios impensable hace unos pocos años al alcance de su móvil.

Como se irá comentando, una de las claves de esta nueva aproximación recae en el proceso de gestión, análisis y securización de los servicios apificados. La implementación de este tipo de arquitectura, por esta razón, no debe realizarse de manera precipitada y sin previsión, a pesar de lo rápida e inminente que se prevé que deberá ser. Es por ello que resulta de enorme importancia el realizar un trabajo de estas características, que contribuya a sentar unas bases que considerar a la hora de la implantación de las arquitecturas API.

4.2 OBJETIVOS

En primer lugar se llevará a cabo una investigación acerca del concepto de gestión de la API en la que se tratará de responder a una serie de preguntas pata tratar de concretar un concepto aún tan ambiguo pero con tanto futuro, ¿En qué consiste? ¿Qué valor aporta al

concepto tradicional de API? ¿Qué beneficios le supondría a una organización adoptar este modelo? ¿Qué implicaciones tendría para la misma? ¿Qué barreras de entrada debe superar para lograr implantar este modelo? ¿En qué contextos se debería destinar mayores recursos a unos aspectos que a otros, como por ejemplo seguridad, gestión de errores o auditoría?

Posteriormente se procederá a hacer un análisis acerca de las soluciones adoptadas a día de hoy en la empresa BBVA para integrar sus sistemas internos, concretamente dentro del área de arquitectura técnica e integración del área CIB (*Corporate and Investment Banking*) de la empresa. Se efectuará, asimismo, una investigación acerca del modelo del sistema de API pública que pretende ofertar el banco para dar acceso a terceros a servicios e información interna, así como el modo por el cual pretende beneficiarse de esta liberación de información.

Paralelamente a este estudio se trabajará dentro de un proyecto de apificación de la arquitectura existente en el área CIB de BBVA. En concreto, el objetivo de este proyecto consiste en exponer una serie de servicios proveedores de datos maestros de la arquitectura *backend* de BBVA mediante APIs *restful* que devuelvan la información en formato JSON y ser capaces de implantar las características del API *management* como seguridad, monitorización, monetización, etc… El objetivo concreto del presente proyecto dentro del proyecto explicado consiste en el estudio y adaptación del formato de datos existente en tres de estos servicios de *backend* al estándar propuesto por la regulación PSD2, el ISO 20022, y adaptarlo al formato JSON. A continuación se pretende desarrollar los ficheros RAML de estos tres servicios, que describan sus respectivas APIs, y que serán utilizados para el desarrollo de las mismas.

Por otra parte, con la intención de comprobar el funcionamiento, utilidad y beneficio de la apificación de estos servicios, se desarrollará una aplicación cliente consumidora de estos tres servicios. La aplicación se desarrollará para Android, tratando de demostrar de esta manera la facilidad que proporcionan las APIs para la integración con diferentes tipos de tecnologías. El objetivo consiste en que esta aplicación sea capaz de presentar a los

usuarios finales la información expuesta en los servicios de una manera clara y con una utilidad real, es decir, cubriendo necesidades específicas.

El objetivo global, por tanto, de este proyecto, consiste en, al abarcar un rango tan amplio de la cadena de valor de la API, que se explicará más adelante detalladamente, poder obtener una visión clara y objetiva que permita valorar si realmente la apificación es una solución efectiva a largo plazo de cara a las necesidades de integración de tecnologías que se demandan hoy en día.

4.3 METODOLOGÍA Y PLANIFICACIÓN

Se pretende emplear una metodología ágil para desarrollar el proyecto, dentro de la cual se irán realizando *sprints* quincenales de cara a alcanzar los objetivos especificados. Este tipo de metodología puede ser adecuada para la realización de este proyecto debido a que está pensado para poder adaptarse rápidamente a los cambios que puedan surgir durante el proceso, y en este proyecto en concreto los objetivos técnicos se irán especificando más a medida que se avance en las cuestiones de investigación y se puedan extraer conclusiones de cara al desarrollo técnico.

Se planifica asimismo una reunión de seguimiento con el director del proyecto una vez cada dos semanas.

Aunque se emplea una metodología ágil, se incluye un diagrama de Gantt a alto nivel con las tareas más generales que se llevarán a cabo con unas fechas provisionales, aunque en la práctica se utilizará una metodología ágil.

	Nombre	Duracion	Inicio	Terminado
1	Análisis	20 days	6/02/17 8:00	3/03/17 17:00
2	Estudio de la gestión de la API	25 days	6/03/17 8:00	7/04/17 17:00
3	Desarrollo de los servicios	30 days	17/04/17 8:00	26/05/17 17:00
4	Pruebas integradas	10 days	29/05/17 8:00	9/06/17 17:00
5	Pruebas de usuario	31 days	12/05/17 7:00	23/06/17 17:00

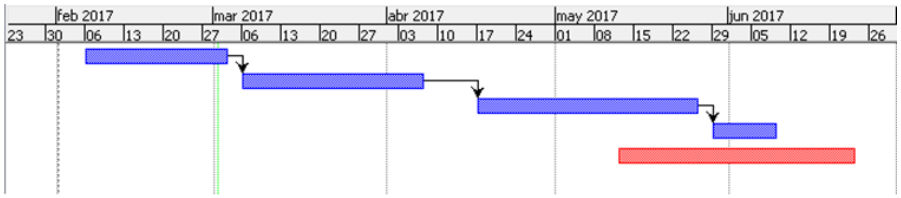


Figura 3 – Cronograma del proyecto.

4.4 ESTIMACIÓN ECONÓMICA

A continuación se presenta una tabla que muestra una estimación económica aproximada del proyecto, el cual ha tenido una duración aproximada de tres meses.

Aunque el proyecto se engloba dentro de uno de mayor envergadura, y por tanto, con un coste notablemente mayor, en esta estimación se consideran únicamente los costes derivados del proyecto individual desarrollado en este documento.

Concepto	**Justificación**	**Coste**
Equipo	Ordenador portátil con procesador i7, 8GB de memoria RAM y disco duro de 500 GB.	600 €
Dispositivo móvil	Realización de pruebas durante el desarrollo de la app móvil. Dispositivo móvil con SO Android, versión 7.	300 €
Android Studio y Android Emulator	Licencia gratuita.	0 €
Atom, SoapUI y Git	*Software* libre.	0 €
Coste personal	Coste de analista y desarrollador: 50 €. Duración del proyecto: 240 horas (3 meses a media jornada).	12 000 €
Total		**12 900 €**

Tabla 1 – Estimación económica.

El proyecto, por tanto, supondrá una inversión aproximada de 12 900 €.

5. LA GESTIÓN DE LA API

En este capítulo se pretende hacer una descripción del proceso de apificación que están llevando a cabo las empresas que se quieren posicionar como líderes en el mundo IT, haciendo especial hincapié en este proceso dentro del mundo de la banca, mundo en el cual, como ya se ha comentado, este proceso resultará imprescindible de cara a la transformación hacia el llamada *open banking*, el futuro modelo en que operará la banca.

Se comenzará realizando un análisis acerca de los principales aspectos de la gestión de las nuevas plataformas API como la necesidad de establecer un gobierno que dirija la toma de decisiones en cuanto a definición, desarrollo, publicación,…; el ciclo de vida de las APIs y la seguridad en el proceso de autenticación de los usuarios. Posteriormente se realizará un análisis acerca del funcionamiento de las API, en qué modelos se basa actualmente y de qué tecnologías hace uso su estructura más funcional.

5.1 ¿QUÉ SE ENTIENDE HOY EN DÍA POR APIS?

Como ya se ha comentado, una API, entendida simplemente como un interfaz de comunicación entre dos sistemas diferentes no es ningún concepto nuevo, sino que se ha venido desarrollando de una manera u otra desde que comenzaron a surgir las redes informáticas. Lo que se entiende hoy en día por API va mucho más allá, una API ha pasado a considerarse un producto en sí mismo, producto que constituye el nuevo canal de relación tanto a nivel interno dentro de las empresas como entre diferentes empresas y entre toda clase de socios y clientes. El factor que ha hecho que las APIs hayan cobrado tanta relevancia en los últimos años es que han pasado de ser el canal de comunicación entre dos sistemas a convertirse en el capaz de conectar un sistema con cualquier sistema de un entorno con una cantidad de datos cada día más elevada y más heterogénea, de una manera rápida, ágil y flexible gracias a los estándares que se están desarrollando. Las APIs son los elementos que permitirán la evolución y el desarrollo de la llamada economía

colaborativa digital, permitiendo la aparición de nuevas empresas que presten servicios del estilo de los ofrecidos por Uber, AirBnb, Blablacar o Wallapop por poner algunos de los ejemplos más conocidos. La siguiente figura pretende mostrar el modelo de interconexión de diferentes sistemas gracias a las APIs.

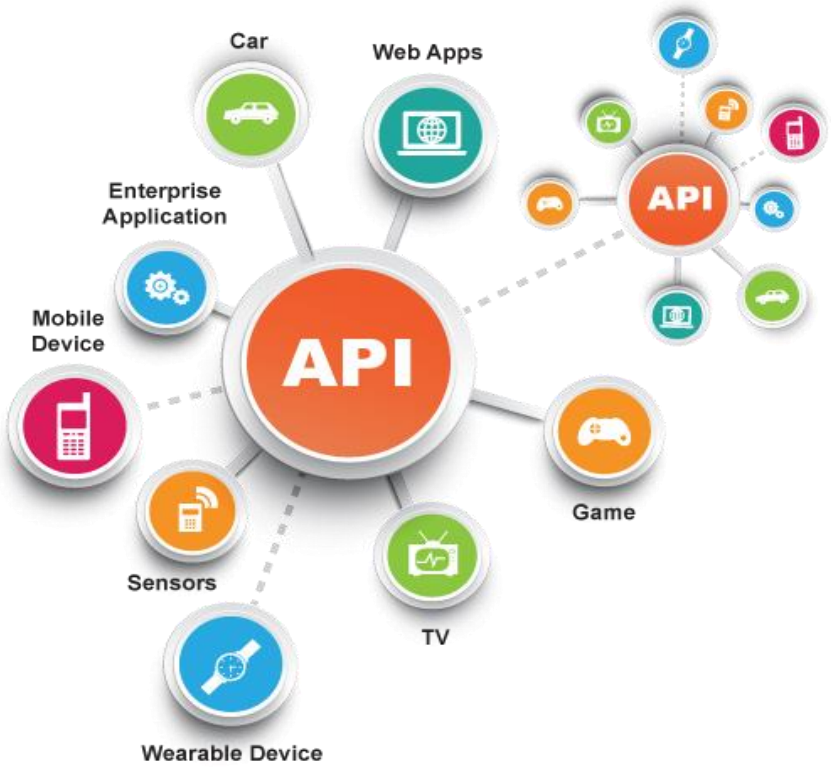


Figura 4 – Las API permitirán interconectar todas las nuevas tecnologías que surjan en los próximos años

Podemos clasificar los diferentes tipos de APIs en tres grandes grupos:

- Las APIs públicas: son aquellas que las empresas ponen a disposición de los desarrolladores ajenos a la propia empresa, ofreciendo a través de ellas datos o servicios. Normalmente son grandes empresas tecnológicas como Google, Facebook, Twitter o Amazon las que abren a los usuarios este tipo de infraestructuras, aunque también son muchas empresas con negocios en principio más alejados de las tecnologías de la información las que están invirtiendo en estos

sistemas, es el caso, por ejemplo, de BBVA, con el ya comentado desarrollo del BBVA API *Market* [10].

La inversión en este tipo de sistemas tiene como objetivos la expansión de la firma hacia otros modelos de negocio sin la necesidad de realizar una inversión directa, ya que el objetivo es que sean los propios desarrolladores los que al crear sus aplicaciones haciendo uso de estas APIs en diferentes entornos y modelos de negocio, alcancen nuevos de nicho, generando un beneficio para la empresa que abre sus servicios, para los desarrolladores, que podrán ofrecer servicios a través de sus propias aplicaciones que solamente son capaces de ofertar empresas con grandes recursos o muy especializados, y para los usuarios finales que se beneficien del uso de estas aplicaciones. Además, el despliegue de este tipo de aplicaciones permite aumentar el número de usuarios finales de la empresa, sin que esta tenga que hacer una inversión adicional en el desarrollo de nuevos servicios.

Las empresas tienen, por otra parte, dos formas de rentabilizar la inversión que requiere desarrollar una infraestructura de API pública, la primera, consiste simplemente en cobrar a los desarrolladores por prestarles sus servicios. El modelo que más están utilizando las empresas a día de hoy es el de ofrecer una versión gratuita que permita realizar un número limitado de llamadas a su API en un tiempo determinado, y ofrecer versiones de pago en el caso en que se necesite superar este límite de llamadas. Para llevar a cabo este modelo, resulta fundamental gestionar de manera eficaz la monitorización del tráfico API, uno de los aspectos más importantes del *API management*, de manera que se pueda controlar quién está realizando llamadas API, y cuantas.

El otro modelo de monetización de las API públicas consiste en ofrecer los servicios de manera completamente gratuita sin importar el número de peticiones que se realicen, esperando obtener un beneficio únicamente a través de la expansión de la empresa gracias a las aplicaciones desarrolladas que hagan uso de sus servicios. Esta segunda opción entraña más riesgos y el tiempo de retorno de la inversión se ve notablemente incrementado por lo que únicamente se lo pueden

permitir empresas muy asentadas en el mercado, como es el caso de los gigantes tecnológicos como Google, Facebook y Twitter.

- Las APIs privadas o internas: son aquellas en las que el dueño de los recursos o servicios a ofertar pone los mismos a disposición únicamente dentro de la propia empresa, generalmente para integrar de manera eficaz sistemas pertenecientes a diferentes departamentos. Este tipo de APIs no están teniendo tanta repercusión como las públicas, sin embargo, están teniendo una importancia cada vez mayor en los modelos de negocio de empresas de todo el mundo.

  Desarrollando este tipo de servicios se busca el desacoplamiento de los servicios, es decir, el establecimiento de un contrato con el consumidor del servicio que establezca todo lo relacionado con la entrada y la salida del mismo, siendo posible cambiar la implementación interna del servicio, e incluso cambiar el propio servicio, siempre que se respete el contrato de entrada y salida. Gracias a este desacoplamiento de los servicios se pretende simplificar las operaciones internas, favorecer la colaboración entre diferentes departamentos y entidades dentro de las propias empresas y consolidar los canales de distribución.

  Un aspecto que se debe tener muy en cuenta a la hora de desplegar servicios de este tipo, consiste en que, al ofrecerse los servicios únicamente dentro de la propia empresa, en muchas ocasiones no se tiene en cuenta si los diferentes departamentos o los desarrolladores de la empresa tienen conocimiento de la existencia de estos servicios, desaprovechando así la inversión realizada en el desarrollo de los servicios. Es por esta razón que resulta igual de necesario al desarrollar tanto APIs privadas como públicas, un sistema o entorno al que los desarrolladores tengas fácil acceso y en el cual puedan informarse de los servicios disponibles y de su funcionamiento, el desarrollo de este tipo de entorno es un aspecto clave de la gestión de las APIs.

- Las APIs para *partners*: son aquellas que se desarrollan con el objetivo de desacoplar servicios entre diferentes empresas que actúan como *partners*.

Resulta especialmente importante en este tipo de APIs la seguridad, ya que las comunicaciones se realizan externamente, entre diferentes empresas, pero los datos o servicios se ofrecen a un único usuario, el *partner*, por lo tanto, el proceso de autenticación y autorización debe ser muy robusto.

Este tipo APIs permitirán agilizar y flexibilizar en gran medida los procesos de colaboración entre diferentes empresas, un proceso que está cobrando gran importancia en los últimos años, ya que la tendencia consiste cada vez en mayor medida en que las empresas se especialicen cada vez más en temas más concretos, y cada vez se deleguen más procesos a empresas externas. Según este modelo de trabajo, logrando que los procesos de colaboración resulten rápidos y eficientes, el resultado final no puede ser otro que el lograr un producto de mayor calidad.

En el apartado siguiente, una vez se haya explicado la cadena de valor de las APIs, se explicará un posible caso real de utilización de las APIs de *partners* dentro de los procesos financieros.

5.2 LA CADENA DE VALOR DE LAS APIS

Como vemos, cada uno de los grupos en los que se engloban las APIs persigue diferentes objetivos y está dirigido a distintos tipos de usuarios, sin embargo, todas comparten un mismo esquema que explica su cadena de valor y que se ilustra en la siguiente figura.

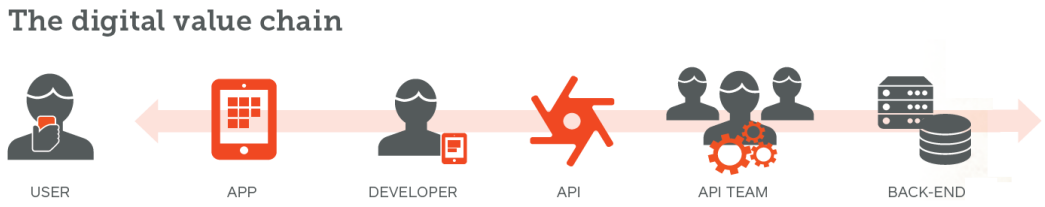


Figura 5 – La cadena de valor de las APIs [13]

En uno de los extremos de la cadena tenemos el llamado *backend*, que son los servicios, programas y datos de los que disponen las empresas, desarrollados en todo tipo de tecnologías y en cualquier formato. La empresa dueña de estos recursos puede pretender hacerlos más accesibles dentro de la propia empresa, accesibles para otras empresas con las que tenga acuerdos o directamente, pretende poner estos recursos a disposición de desarrolladores totalmente ajenos a la empresa tal y como se ha explicado.

En el caso de que se tome cualquiera de estas decisiones, el siguiente paso consiste en hacer que estos recursos, desarrollados en cualquier tecnología, en muchas ocasiones en sistemas *legacy* propios de la empresa pesados y muy poco flexibles, estén disponibles sin necesidad de tener que adaptarse a estas tecnologías o tener que crear complicados sistemas de traducción entre formatos para cada servicio, como ocurría en los modelos de las conexiones punto a punto. Para llevar esto a cabo se apifican los servicios mencionados, de manera que estos recursos internos, ya sean servicios o información sean accesibles a través de APIs.

Una vez que se publican estas APIs, para continuar la cadena de valor, resulta necesario que éstas mismas lleguen al conocimiento de algún grupo de desarrolladores, ya sean internos o externos, y que éstos les den algún uso mediante las aplicaciones que desarrollen, aplicaciones web, aplicaciones móviles o de cualquier tipo dependiendo del caso.

Por último, y para completar la cadena de valor, resulta igualmente necesario que una vez creadas las aplicaciones, lleguen de alguna manera a las manos de los usuarios finales que harán uso de estas. Para que esta cadena sea realmente exitosa, además de que los usuarios encuentren algún beneficio del uso de estas aplicaciones, éstos a su vez deben proporcionar algún tipo de beneficio a los desarrolladores de la aplicación, a los proveedores de la API y al dueño de los recursos del *backend*. [1]

5.2.1 LA CADENA DE VALOR DE LAS APIS DENTRO DEL MUNDO FINANCIERO

Tradicionalmente los bancos han ocupado toda la cadena de valor de negocio financiero, ellos mismos eran los encargados de analizar las necesidades de los clientes, de diseñar los nuevos productos para ofertarles, de realizar el *marketing*, y evidentemente se han encargado de absolutamente todos los aspectos del *backoffice* y la gestión de los riesgos, es por ello que tradicionalmente se ha considerado a los bancos como entidades muy herméticas.

En el siguiente esquema se explica más en detalle el proceso típico que sigue una entidad financiera, en este caso BBVA, a la hora de crear un producto financiero y ofrecerlo al cliente.

En primer lugar, el banco se encarga de analizar las necesidades y demandas de los clientes y a partir de estos resultados se encarga de crear nuevos productos financieros y de ofertarlos a sus clientes. El propio banco se encarga de gestionar los aspectos de la comercialización del producto, es decir, de hacer que el producto llegue al cliente, como por ejemplo el *marketing*. Gestiona los riesgos derivados de la puesta en marcha del producto, garantiza la liquidez de los activos y se encarga de la contabilidad. Como ilustra la imagen, todos estos procesos se llevan a cabo dentro de la entidad.

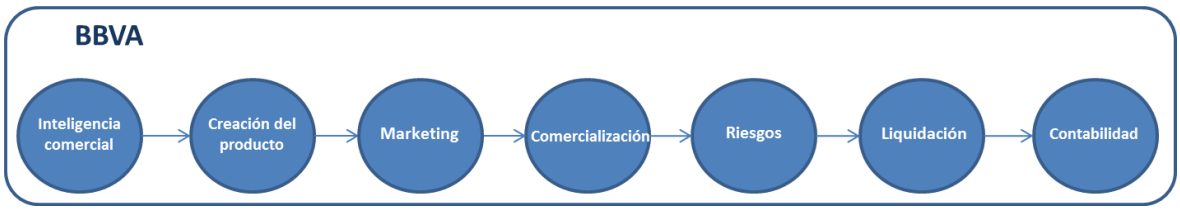


Figura 6 – Esquema del flujo tradicional de tareas realizadas por los bancos.

En los últimos años han aparecido nuevos actores en esta cadena, que se centran en partes muy específicas de ella, tratando de aportar un valor añadido de cara al cliente. Es el caso por ejemplo de PayPal o de las nuevas *fintech*, empresas nacidas normalmente con formato

de *startups*, que tratan de cubrir partes específicas del proceso total que siguen los bancos. Por ejemplo, empresas que se encargan únicamente de la realización de pagos *online*, o los mencionados proveedores de información de cuentas que aparecerán cuando se implante la PSD2, y que de hecho ya están apareciendo.

Los bancos, por tanto, si no quieren perder completamente su nicho de negocio deben ser capaces de adaptarse a esta nueva coyuntura. Debido a sus características, generalmente los bancos son empresas muy grandes, en algunos de los puntos de la cadena, no pueden competir con la capacidad de innovación que puede tener una *startup*, sin embargo, sí que hay otros procesos, por ejemplo, la gestión de toda la infraestructura de *bakoffice*, con la que este nuevo tipo de empresas no puede lidiar, debido a la cantidad de recursos necesarios para llevar a cabo estas tareas. La clave, por tanto, está en llegar a un punto de colaboración entre empresas, que sean capaces, centrándose cada una en las tareas que mejor sabe realizar, ofrecer en conjunto un producto final de máxima calidad.

Un ejemplo de esta nueva manera de operar es el caso de la compra del banco digital americano Simple por parte de BBVA. Este banco fue fundado en 2009 en Oregón y tiene una manera de operar bastante alejada de los bancos más tradicionales, está enfocado principalmente a una clientela joven a la que ofrece pequeñas cuentas de ahorro y aplicaciones con la idea de gastar de manera inteligente y ahorrando lo máximo posible. [14] Simple es, por decirlo de alguna manera, una entidad bastante más ligera que los bancos tradicionales, sin embargo, a pesar de ello, no deja de ofrecer el mismo tipo de servicios que cualquier otra entidad financiera, aunque esté enfocado a un tipo de clientela mucho más específica y los servicios tengan un enfoque diferente, es por ello que debe cubrir la totalidad de la cadena de negocio financiera para poder dar un servicio fiable. Para una empresa nacida con formato de *startup*, resulta prácticamente inalcanzable el desarrollo de una infraestructura de *backend* operativa. Para hacer frente a este problema, la filial de BBVA americana, BBVA Compass compró Simple a principios de 2014. Con esta compra, se produce un beneficio mutuo enorme, por un lado, permite a Simple no depender de sí mismo, una empresa pequeña y con pocos recursos en comparación a otros bancos, para realizar las tareas de negocio más pesada, y poder centrarse en su verdadero nicho de

negocio, mucho más centrado en el contacto directo con el cliente. Por otro lado, a BBVA le permite expandir su negocio llegando a toda esta nueva clientela, además de beneficiarse de todas las ideas y nuevas formas de negocio que puede crear una entidad del tipo de Simple.

Esta nueva forma de operar queda ilustrada en el siguiente esquema, el cual representa el mismo proceso que el del esquema anterior, sin embargo, en este caso, participan nuevos actores dentro de él. El recuadro verde de la parte inferior representa los nuevos actores especializados en tareas concretas, como podrían ser la creación del producto y la contabilidad, y que las realizarían en lugar de BBVA. Por otra parte, el recuadro naranja representa a otros bancos, los cuales podrían llegar a acuerdos con BBVA para que éste llevase la gestión de los riesgos, tarea en la cual éste se habría especializado.

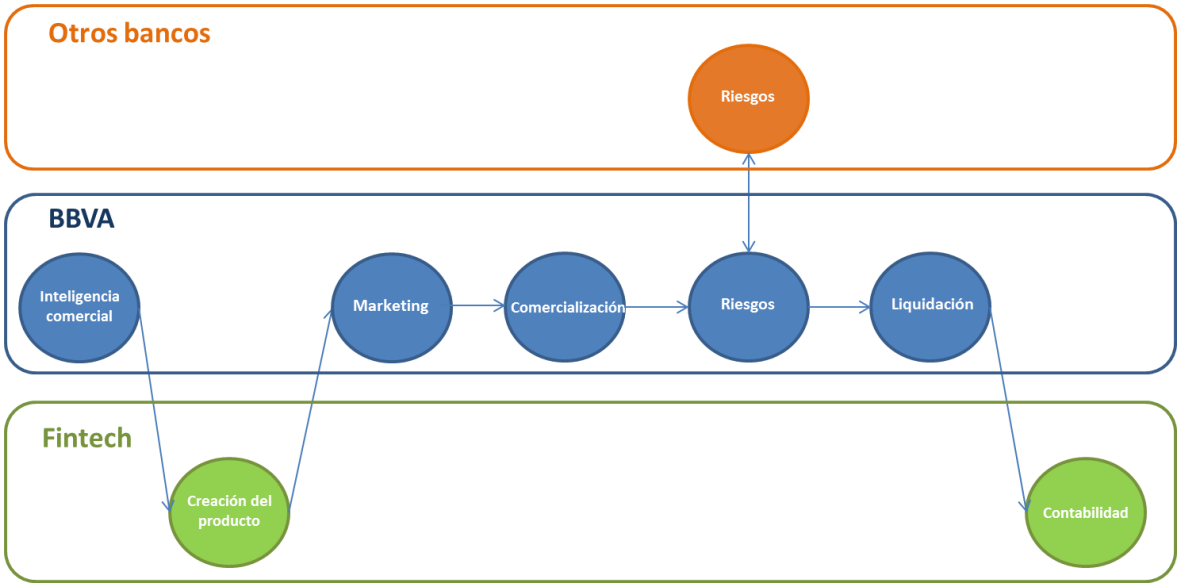


Figura 7 – Esquema del flujo de tareas realizadas por los bancos integrando procesos gracias a la apificación.

Este nuevo escenario permite que, al aumentar la competencia al permitir que entren nuevos actores donde antes no era posible, debido a que esta cadena estaba completamente cerrada a la colaboración, las empresas puedan especializarse lo máximo posible en las tareas que mejor saben realizar y así ofrecer un servicio de gran calidad.

El principal impedimento para la realización de este escenario consistía en que al ser un proceso complejo, la colaboración directa entre entidades era muy complicada de llevar a cabo. La apificación de las infraestructuras, sin embargo, permitirán llevar a cabo este escenario gracias a que permiten agilizar y flexibilizar en gran medida estos procesos al ofrecer el consumo de recursos de manera sencilla.

5.3 EL GOBIERNO DE LAS APIS

Es importante entender que uno de los cambios más importantes y que han llevado dotar a las APIs de tanta importancia en los últimos años ha sido que éstas han pasado de ser simples objetos de *software*, a convertirse además en auténticos elementos de negocio. Es por ello que la gestión de estos elementos ha dejado de ser únicamente un proceso centrado puramente en los aspectos más tecnológicos como pueden ser la seguridad o la gestión de eventos y de errores para pasar a ser igualmente necesaria la gestión de aspectos más orientados al aspecto estratégico y de negocio.

Por otra parte, echando un vistazo a la cadena de valor explicada en el apartado anterior, vemos como para que llegue a tener éxito el proceso de apificación de cualquier sistema, es necesario que entren en juego muchos actores diferentes, en ocasiones completamente ajenos a la entidad que proporciona los recursos, y que cada uno de los procesos que componen la cadena de valor funcionen correctamente. Es por ello que, si ya resultaba necesaria la adopción de un gobierno para dirigir eficientemente procesos de *software* más tradicionales, en los procesos de apificación, el gobierno resulta una pieza aún más fundamental.

Dentro de un departamento de IT, el gobierno es el equipo encargado de definir y establecer políticas y procedimientos para gobernar el desarrollo, despliegue y gestión de los servicios y procesos de negocio que se definan y desarrollen en el área de IT. El gobierno es responsable de qué decisiones se toman, quién toma estas decisiones y cómo se implementan. [15]

Dentro de la planificación de una API, el gobierno es el equipo encargado de decidir si para una necesidad concreta se debe reutilizar una API existente o se debe crear una nueva versión con otra funcionalidad; debe definir su ciclo de vida; decide qué tecnologías se utilizan para el desarrollo, pruebas y mantenimiento; define los roles y reparte las responsabilidades dentro del proceso; se encarga de determinar la manera de hacer la API accesible a los posibles desarrolladores estableciendo herramientas como un gestor documental; y decide qué APIs se deben publicar a qué público y la manera de medir el éxito de esta publicación. Por tanto, las principales decisiones respecto a la gestión de la API dependen directamente de este organismo.

5.4 EL CICLO DE VIDA DE LAS APIS

El ciclo de vida de las APIs tiene como principales características el ser relativamente corto, ágil y flexible en comparación a otros servicios y aplicaciones como los *web services* por ejemplo. El hecho de que cada vez se demande un *time to market* más reducido para cualquier tipo de servicio *web* es el que está forzando que el ciclo de vida de las APIs sea cada vez menor, a la vez que es una de las razones por las cuales las APIs se están posicionando como herramientas imprescindibles dentro del entorno IT. Otra de las características actuales de los productos digitales es la rapidez con la que cambian sus requisitos, lo que obliga a disponer de herramientas que se adapten a ellos a una velocidad que no podrían soportar los tradicionales servicios *legacy*. La flexibilidad es una de las características más importantes de las APIs, pues una de las claves de su auge es que son capaces de proporcionar capacidades muy elevadas de integración a prácticamente cualquier tipo de dispositivo, servicio o entidad que las requiera con los servicios a apificar. Para que las APIs implantadas cumplan con estas características absolutamente imprescindibles para cumplir con los requisitos tecnológicos demandados actualmente, es importante incluir a los usuarios posicionados al otro lado de la cadena de valor tanto en el proceso de diseño e implementación como en el de publicación, para ello, las empresas tratan cada vez en mayor medida de recibir y atender comentarios y propuestas tanto de desarrolladores como de usuarios finales, logrando que se dé esta participación y

gestionándola eficientemente permite poder adaptar el ciclo de vida de la API a las necesidades que se estén produciendo externamente en cada momento.

Las APIs no deberían ser únicamente productos informáticos que puedan entender únicamente los expertos en IT, por el contrario, se está evolucionando con mucha rapidez hacia un escenario donde las tecnologías están presentes en cada vez más hábitos de la vida cotidiana, aunque no todo el mundo necesariamente debe tener los conocimientos más técnicos de los procesos que hay detrás. Es por ello, que para que una API pueda formar realmente parte de este fenómeno de informatización de tantos procesos ajenos al ámbito de la informática, resulta necesario que tenga unos objetivos bien definidos que estén directamente relacionados con los procesos de negocio, los cuales sean capaces de responder de manera clara a preguntas alejadas de la jerga más tecnológica como por ejemplo ¿Qué problema se pretende solucionar?, ¿Qué oportunidades se están tratando de aprovechar?, o ¿Qué ventajas se les están proporcionando a los potenciales usuarios que se beneficien de estos servicios?. [1]

El ciclo de vida definido, y que se emplea como referencia para la definición del ciclo de vida de las APIs en BBVA, se basa en un conjunto de fases divididas en dos grupos. Estas fases contemplan todas las tareas a realizar sobre las APIs desde la identificación, hasta su publicación y posterior monitorización y evolución.

Las fases del ciclo de vida se clasifican en función de su ámbito de acción. Por este motivo existen dos grupos, uno para las fases de desarrollo que contiene las fases que permiten identificar, especificar tanto funcional como técnicamente y construir la API o producto digital. Y un segundo grupo para las fases de ejecución que comprenden las tareas necesarias para la publicación, puesta a disposición de los consumidores, monitorización y evolución de las APIs. Entre este segundo grupo cabe destacar las fases dedicadas a promover el uso de las APIs entre los consumidores y analizar el uso que se hace de ellas con objeto de tomar las mejores decisiones en cuanto a su evolución.

A continuación se listan y describen detalladamente las nueve fases que comprenden el ciclo de vida de las APIs *end-to-end* dentro de BBVA.

Figura 8 – El ciclo de vida de las APIs [15]

- Identificación: La Identificación de APIs consiste en el proceso de analizar las capacidades y/o necesidades del negocio para obtener un conjunto de APIs candidatas que puedan aplicar valor en el nuevo canal de negocio formado por los productos digitales. El proceso de identificación propuesto usa dos aproximaciones complementarias como son *Topdown* (técnica de identificación orienta a negocio) y *Botton-up* (técnica orientada al departamento de IT). Como resultado de este proceso se obtiene un listado de las APIs candidatas a ser construidas junto con una breve descripción funcional de las mismas y la justificación de su necesidad.
- Especificación: Esta fase consiste en filtrar las APIs candidatas obtenidas en la fase anterior para obtener únicamente aquellas que aportan valor a la organización y por lo tanto deben ser construidas. Como parte de esta fase también se debe especificar el contrato o interfaz de cada una de estas APIs mediante el cual se define el comportamiento de la API.
- Realización: Una vez definida la interfaz de la API, en esta fase se toman las decisiones técnicas necesarias para la implementación de la API. En esta fase, en

base a la arquitectura técnica de referencia, se definirá entre otras características técnicas la tecnología a utilizar para la implementación de la API, protocolos a exponer y criterios de seguridad y no funcionales a aplicar.

- Implementación: Esta fase comprende el desarrollo de la API a partir de la definición de la interfaz realizada en la fase de especificación y de las decisiones técnicas adoptadas en la fase de realización. El resultado final de esta fase debe consistir en una versión de la API testeada y lista para ser desplegada en el entorno productivo y publicada en el catálogo de APIs para que empiece a ser utilizada por los desarrolladores.
- Configuración: Esta fase tiene el objetivo de habilitar desde el punto de vista técnico el nuevo producto digital. Para ello, en esta actividad se asignan los permisos técnicos necesarios para la ejecución de la API: permisos de acceso a los servicios de negocio, permisos de ejecución a los usuarios implicados, etc. En esta actividad también se configuran los acuerdos de nivel de servicio (SLAs) acordados con los consumidores de las APIs. Como resultado de esta actividad se obtiene la API preparada para ser consumida por los desarrolladores de aplicaciones.
- Publicación: En esta fase, una vez que la API ya se encuentra disponible para ser utilizada, y una vez que ha sido desplegada y dada su visión externa, se debe registrar en el catálogo de APIs corporativo. De esta forma se pone a disposición de la comunidad de usuarios que son los desarrolladores de aplicaciones consumidoras de APIs. Para favorecer la utilización de las APIs, la publicación debe ir acompañada de documentación adicional que facilite su uso por parte de los desarrolladores de Apps. Esta documentación debe incluir información como el detalle funcional del comportamiento de la API, ejemplos prácticos de uso, buenas prácticas y modelos de uso de la API (gratuita, pago por uso, incluye publicidad, cobro por uso, etc.
- Socialización: El principal objetivo de esta actividad consiste en fomentar el uso de la API y mejorar la experiencia de uso de la misma por parte de los desarrolladores, que son los consumidores de las mismas. Para ello, esta fase recoge todas las tareas necesarias para crear y mantener la comunidad de usuarios en torno a las APIs.

Bajo esta actividad del ciclo de vida se debe dar respuesta a las consultas realizadas por los consumidores relacionadas con el uso y operativa de la API.

- Monitorización: Una vez que la API ya se encuentra publicada en el catálogo de APIs y ha sido debidamente documentada y catalogada, se realiza la monitorización de su uso, se realizan informes y se indica el grado de cumplimiento de las SLAs. La monitorización de las API tiene 2 vertientes diferenciadas, con distintos objetivos y distinto público. Por un lado se considera la monitorización de la API como elemento técnico de la arquitectura, este tipo de monitorización consiste en generar informes donde mostrar el rendimiento de la API: tiempo de ejecución, tiempo de espera a los servicios invocados, etc. Por otro lado se encuentra la monitorización de negocio de la API, encargada de recoger información que permita comprender a los clientes, su relación con el producto y la monetización de la API como puede ser: número de ejecuciones, número de aplicaciones que usan esta aplicación, número de ejecuciones de la aplicación, etc.
- Monetización: La elección del modelo de monetización de la API se realizará en la fase de Identificación de la API, al principio del ciclo de vida de la misma y a la vez que la identificación de los beneficios que produce. El modelo de monetización final podrá ser: gratuito, con pago por parte del usuario o incluso con cobro por parte del usuario de la API, opción que puede resultar adecuada en casos de optar por dar publicidad al catálogo de APIs existentes para aumentar la cuota de popularidad entre los desarrolladores.

5.5 API MANAGER

API *management* es el proceso de gestión, publicación, promoción y supervisión de las APIs en un entorno seguro y escalable. [16]

La siguiente figura pretende esquematizar un modelo de solución de API *manager*.

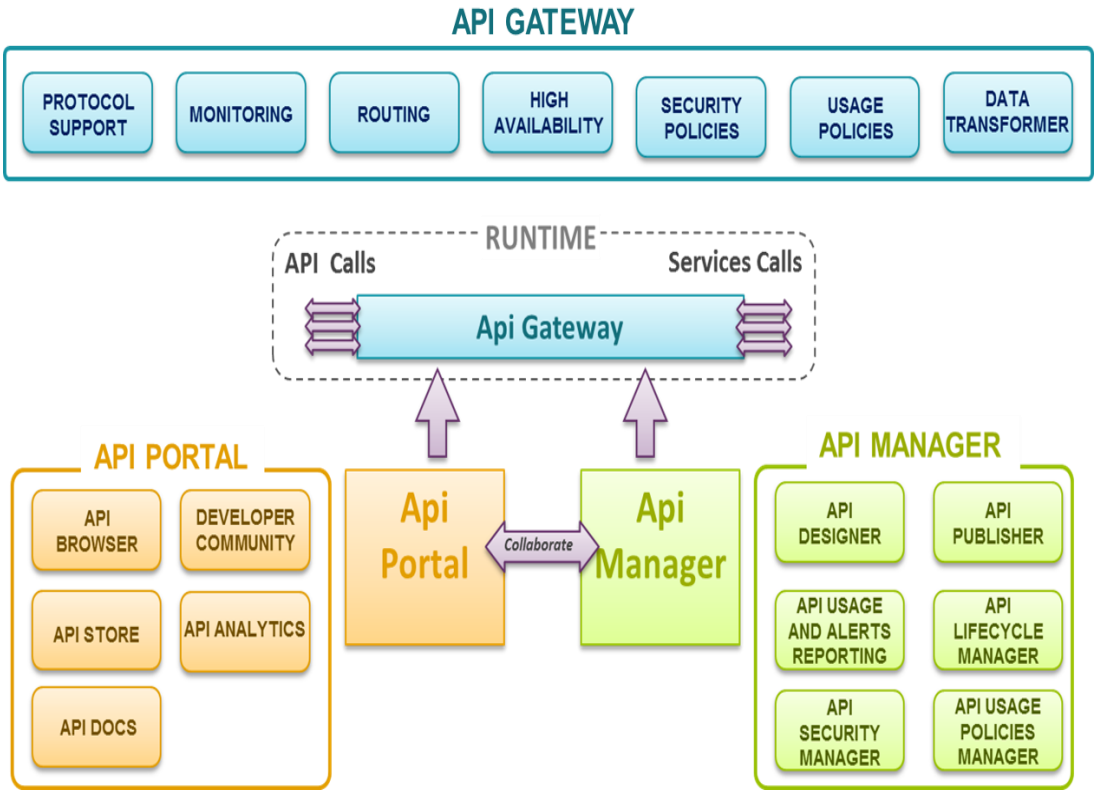


Figura 9 – Esquema de una solución API Manager [15]

El API *manager* se compone de tres piezas principales, el API *Gateway*, el API *Manager* y el API *Portal*.

- El API *Gateway* es el elemento que actúa como puente entre las llamadas a los recursos internos y las aplicaciones que realizan estas llamadas.
- El API *Manager* es el elemento encargado de garantizar que se lleve a cabo correctamente el ciclo de vida de las APIs, es donde se configuran las políticas de seguridad y es elemento que realiza tareas como la monitorización del consumo de los recursos.
- El API *Portal* es el entorno donde se publican las APIs y su documentación, es el elemento encargado de detectar a los potenciales consumidores y desarrolladores, y

hacerles llegar a estos últimos el conocimiento de la existencia de las APIs y su información.

La adopción de un sistema de estas características por parte de las instituciones que pretenden apificar sus servicios resulta crucial para poder realizar correctamente la gestión de sus APIs.

5.6 LA SEGURIDAD

Uno de los errores más habituales de los proveedores de APIs consiste en confiar en que los desarrolladores de las aplicaciones que se conecten a sus servicios sean los encargados de los aspectos de la seguridad, especialmente en el caso de las API abiertas. Esto puede parecer lo más lógico en un principio ya que en general, serán las propias aplicaciones finales que consuman los usuarios las principales interesadas en general un nivel de confianza sólida que permita que los usuarios continúen utilizándola. Sin embargo, la realidad es que la seguridad no es uno de los primeros aspectos que se tienen en cuenta a la hora de desarrollar aplicaciones, especialmente por parte de grupos pequeños de desarrolladores o *startups* que no cuentan con los *frameworks* con los que típicamente cuentan las grandes compañías y especialmente con los medios necesarios para implantar las medidas necesarias.

Ya que la tendencia de ofertar APIs de manera cada vez más libre y con menores restricciones provocará cada vez más el desarrollo de aplicaciones como las mencionadas en el párrafo anterior, y el riesgo existente para los propios proveedores de APIs debido a que lo que queda expuesto son sus propios datos internos y servicios, parece claro que la seguridad debe ser un factor muy a tener en cuenta en las propias plataformas API.

A continuación se expone unos de los principales problemas para la seguridad a la que se exponen actualmente los servicios REST, y que está estrechamente relacionado con el método de paginación que se expondrá en apartado siguiente. Como se detallará más adelante, si se pretende transmitir una cantidad elevada de información, se deberá realizar también una cantidad elevada de llamadas al servicio, siendo un servicio REST, la

conexión se debe realizar sin estado y por tanto cada llamada debe tener toda la información necesaria para identificar la acción a realizar o el recurso a pedir, y también cada llamada debe ser capaz de identificar quién está realizando la llamada. La primera aproximación para resolver este problema es enviar el identificador y la contraseña del usuario en cada petición, este método no solamente resulta ineficiente para el servicio *backend*, ya que en cada una de las peticiones debe realizar una consulta a base de datos y autorizar o denegar el acceso a la petición, sino que resulta insegura, ya que estos datos pueden ser interceptados mediante ataques tipo *man-in-the-middle*. Si bien es cierto que las técnicas para evitar este tipo de ataques han evolucionado notablemente en los últimos años, gracias a las nuevas técnicas de cifrado y encriptación, y especialmente HTTPS, también han evolucionado las técnicas para interceptar datos sensibles y por ello, el riesgo continúa estando presente.

Entre los principales riesgos que afectan a este tipo de comunicaciones se encuentran la exposición a ataques de tipo XSS y ataques CSRF. De manera resumida, los ataques XSS (o *Cross-site scripting*) consisten en la ejecución de un código por parte del cliente, generalmente JavaScript, el cual puede ser inyectado por un usuario en una página web, o incluso encontrárselo incluido dentro del propio código de la misma, con el objetivo de obtener información del usuario almacenada en forma de cookies, datos incluidos al rellenar un formulario, etc. [17]

Los ataques CSRF (*Cross-site request forgery*) consisten en el engaño por parte del atacante para que el cliente realice una consulta o una petición indeseada, por ejemplo, incluir en una petición que pretenda realizar el cliente una opción para que cambie su contraseña de manera involuntaria. [17] A pesar de que existen otros muchos riesgos para la seguridad, entre ellos otros tipos de ataques comunes como las inyecciones SQL, nos centramos en estos dos por tener en común que ambos consisten en ataques en el lado del cliente, siendo el objetivo del apartado 6 exponer un modelo seguro en este lado de la aplicación.

5.6.1 EL ESTÁNDAR OAUTH

La creciente complejidad de las plataformas API ha provocado como consecuencia una complejidad muy elevada a la hora de establecer medidas efectivas de seguridad, lo cual ha provocado que resulte prácticamente inasumible para empresas pequeñas desarrollar un sistema de seguridad robusto partiendo desde cero. Es por ello que se ha tratado de desarrollar estándares y marcos de actuación para poder establecer sistemas lo más seguros posibles. De cara a algunos aspectos de la seguridad como la identificación, autenticación y autorización de acciones a realizar, el estándar más importante es OAuth.

OAuth es un protocolo abierto definido en la RFC 5849 que define las pautas para permitir que aplicaciones de terceros se comuniquen con otros servicios que provean una API sin necesidad de compartir contraseñas. El estándar OAuth tiene dos principios fundamentales, el primero consiste en otorgar tan solo los permisos necesarios, esto quiere decir que una aplicación debería poder realizar únicamente las acciones que el propietario del recurso considere. El otro principio consiste en dar solamente la responsabilidad necesaria, es decir, permitir a los desarrolladores de las aplicaciones que se van a conectar a los recursos que se centre en desarrollar su funcionalidad, sin preocuparse por temas como los de la seguridad. [28]

OAuth, por tanto, resuelve algunas de las necesidades más importantes que tienen los proveedores de APIs, como son la delegación y la revocación de acceso y la reducción del intercambio de contraseñas entre proveedores y terceras aplicaciones. Otra de las ventajas que entraña no intercambiar contraseñas en las llamadas a servicios, además de la ya mencionada de evitar la interceptación de las mismas, consiste en que si el proveedor quiere denegar el acceso de una aplicación a un recurso específico, la única manera que tiene siguiendo el patrón de intercambio de contraseñas es cambiar las suyas propias, mientras que de lo contrario, se puede denegar el acceso a un recurso específico sin denegarlo a la aplicación entera. [29]

Para gestionar las autorizaciones, OAuth define el concepto de *scope*, algo así como el alcance del usuario. Los *scopes* podrían ser definidos como los permisos o los derechos

que el propietario de un recurso otorga al cliente que quiere acceder a ellos para que realice alguna acción que le beneficie. El cliente solicita los derechos que considera, pero el proveedor puede concederle sólo algunos de los que haya pedido, o incluso concederle alguno que no haya solicitado expresamente, esto según las políticas definidas por el propietario de los recursos.

5.6.2 EMPLEO DE *TOKENS* PARA LA AUTENTICACIÓN EN SISTEMAS REST

A continuación se realiza una introducción a la solución propuesta por OAuth para sustituir el modelo de intercambio de contraseñas, la autenticación basada en *tokens*. Esquemáticamente, este modelo tiene el siguiente funcionamiento: el usuario se autentica en un llamado servidor de autenticación basado en OAuth, empleando el usual par usuario/contraseña, y el servidor le devuelve un *token*, que consiste en una cadena de caracteres con cierta información que veremos más adelante. OAuth además define dos tipos diferentes de *tokens*, los de acceso, que se utilizan para la autenticación en sí, y los de refresco, mediante los cuales, una vez ha caducado el anterior, se puede obtener uno nuevo de acceso. Mediante este *token*, el API debe ser capaz de identificar al usuario en cada petición que realice, petición en la cual adjuntará el *token* en la cabecera. Ya que este *token* se almacena únicamente en el lado del cliente, no hay ninguna información de estado en el lado del servidor, respetando de esta manera el fundamento de REST y permitiendo que el servicio sea mucho más escalable, ya que, por ejemplo, esto permite que se pueda conectar una misma API a cualquier tipo de aplicación, como web, móvil, Android o iOS, con la única condición de que los *tokens* para todos los distintos tipos de aplicación sean del mismo formato, lo cual es posible gracias a ciertos estándares como se verá a continuación.

Existen además varios tipos de *tokens* y OAuth no especifica que se deban usar unos u otros, lo cual proporciona mucha flexibilidad, ya que, teniendo cada uno sus ventajas, se pueden utilizar de diferentes tipos en una misma aplicación, según la situación lo requiera. Entre los tipos más comunes se encuentran los WS-Security *tokens*, entre los cuales destacan los SAML, los JSON web *tokens* o JWT y los *legacy tokens*. [28]

SAML o *Security Assertions Markup Language* es un estándar abierto que utiliza el lenguaje de marcado XML para intercambiar datos de autenticación y autorización. Los *tokens* definidos por SAML consisten en representaciones XML de los denominados "*claims*", que podrían traducirse como afirmaciones, en este caso de la identidad de un usuario. [30]

Los JSON Web *Tokens* los define la propia especificación de JSON como una manera compacta de representar los *claims* mencionados en el párrafo anterior, para ser transferidos entre dos entidades. Estos *claims*, o el identificador y los permisos que son concedidos al usuario se codifican como objetos JSON para ser posteriormente firmados digitalmente y encriptados. [31] [32]

La ventaja de los *tokens* SAML respecto a los JWT consiste en que se pueden configurar con más opciones de seguridad al ser estos más flexibles. Esto supone la desventaja de que resultan más pesados y complejos. [32] SAML además emplea el lenguaje XML, mientras que JWT JSON, lo que a día de hoy supone una ventaja para el segundo, pues JWT está siendo cada vez más extendido, al suponer el empleo de XML una barrera técnica más elevada. Parece, por tanto, a menos que se tengan unas necesidades muy específicas de autenticación, que JWT no pueda asumir con eficiencia, para cuyo caso tenemos la especificación SAML, lo recomendable es el empleo de *tokens* en formato JSON.

5.6.3 ANÁLISIS DE UN MODELO DE AUTENTICACIÓN EN SISTEMAS APIS REST

A continuación se realiza un análisis sobre un proceso de autenticación mediante *tokens* JSON real, el empleado con las APIs abiertas de BBVA. Debido a que cada tipo de servicio tiene una criticidad diferente, por ejemplo, no es igual de peligroso que un usuario no autorizado realice una llamada a un servicio de consulta de saldo de otro usuario, por ponernos en el ejemplo de las APIs de BBVA explicadas en el capítulo 3, que un usuario no autorizado utilice un servicio para realizar un pago con la cuenta de otro usuario. Por esta razón se han definido 3 modelos de autorización con *tokens* JSON, cada uno de mayor

seguridad que el anterior, pero a la vez más lento y pesado, de manera que cada servicio tendrá un nivel de seguridad ajustado a sus características.

- Autenticación de dos patas:

La autenticación de dos patas es el método más ligero y consiste en que la aplicación cliente hace una llamada al servidor de autenticación y recibe como respuesta un *token* de acceso en formato JSON.

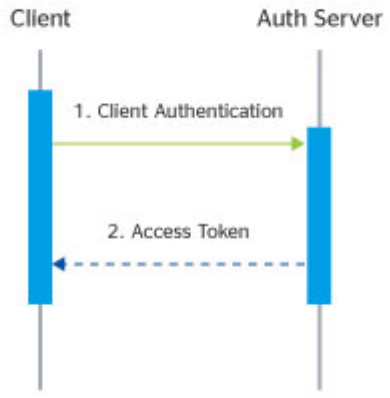


Figura 10 – Proceso de autenticación de 2 patas [10]

Para poder realizar este proceso, el desarrollador de la app debe tener creada previamente una cuenta en el portal, con la cual recibe unas credenciales en forma de parámetros mediante el protocolo OAuth. Teniendo estas credenciales, se puede realizar una petición HTTP POST al servidor de autenticación, incluyendo las credenciales como cabeceras, el servidor devolverá el *token* correspondiente que estará en forma de objeto JSON. Este objeto será similar al que se muestra a continuación, y en él se incluyen el propio *token*, el tipo de *token* (JSON *web token* en este caso), el tiempo en el que expira en segundos y el *scope*, o permisos que se otorgan al cliente que obtenga el *token*.

```
{
        "access_token":
"eyJhbGciOiJSU0EtT0FFUCIsInppcCI6IkRFRiIsImVuYyI6IkExMjhH…",
        "token_type": "jwt",
        "expires_in": 3599,
        "scope": "accounts_full_sbx_1 account_detail_full_sbx_1
account_transactions_sbx cards_full_sbx_1 card_detail_full_sbx_1
me_full_sbx_1 data_manager_sbx transfer_sbx paystats_sbx_test",
    }
```


Una vez que se ha obtenido el *token*, se podrán realizar llamadas a la API REST en cuestión con resultado satisfactorio siempre que se incluya el mismo como cabecera de la petición HTTP con el nombre *Authorization*.

- Autenticación de 3 patas:

Mediante la implantación de este sistema, para llamar a la API se realizará un proceso de autenticación por parte de la app cliente como en el caso anterior y también por parte del propio usuario.

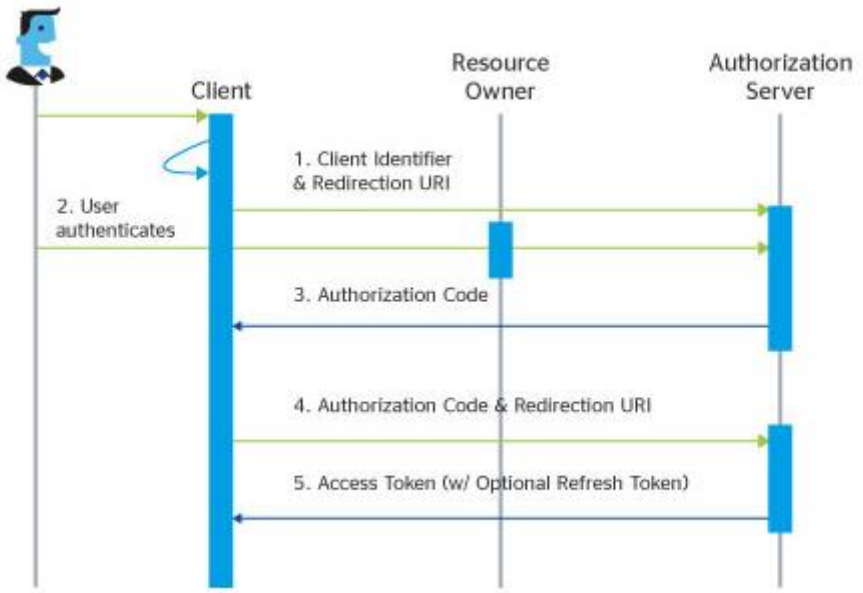


Figura 11 – Proceso de autenticación de 3 patas [10]

En primer lugar, antes de realizar la petición, la aplicación mandará una petición al servidor de autenticación de la misma manera que en el caso anterior, sin embargo, el servidor, antes de devolver el *token* de acceso, devolverá un código que el usuario recogerá desde una página de redireccionamiento que el desarrollador de la aplicación haya configurado previamente. El propio usuario deberá recoger este código y rellenarlo en un formulario para que la aplicación pueda generar una nueva petición al servidor de autenticación añadiendo este código como cabecera, el servidor devolverá el JSON *token* con el que se podrá realizar la llamada a la API de la misma manera que en el caso anterior. En este caso, en el JSON devuelto se incluirá también un *token* de refresco con un tiempo de caducidad mayor al *token* de acceso, una caducado este último, se podrá realizar una nueva petición al servidor de autenticación usando el *token* de refresco para recibir uno nuevo de acceso.

Este modelo es, como se ha dicho, más robusto que la autorización de dos patas al entrar en el proceso un tercer actor, el propio usuario de la aplicación, pero a la vez es más lento y pesado por esta misma razón, por tanto tiene sentido emplearlo únicamente en servicios con información especialmente sensible para el usuario.

- *One Time Password*:

Para acciones donde la seguridad prima por encima de la comodidad, como por ejemplo una API para realizar una transferencia, existe un tercer modelo de autenticación de especial robustez, la contraseña de un único uso.

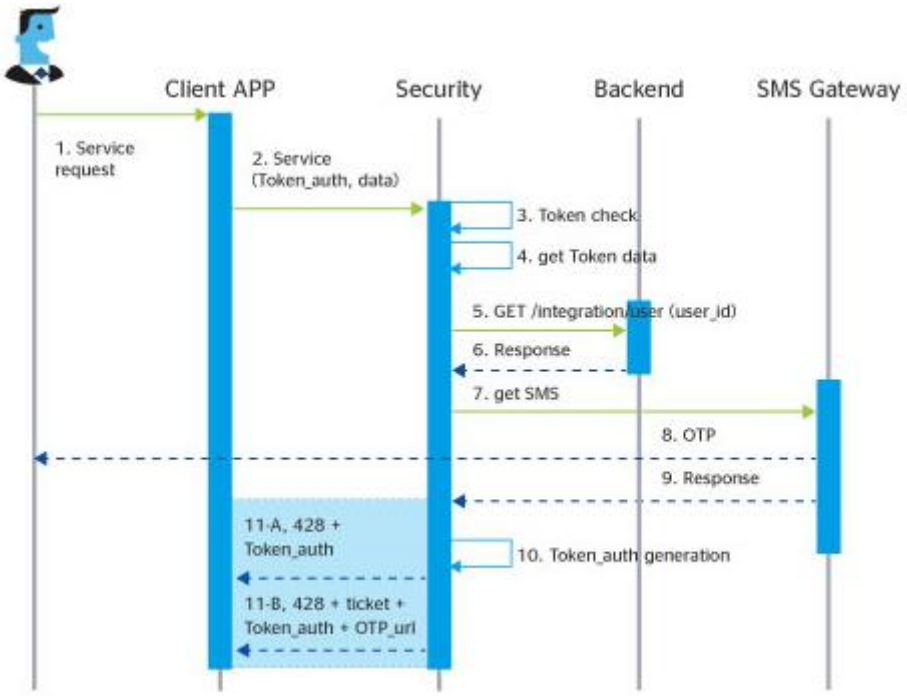


Figura 12 – Proceso de autenticación mediante one time password [10]

Este modelo hace uso en primer lugar de la autenticación de tres patas, ya que hay que seguir el proceso de obtención del código y del *token* explicado en el apartado anterior. Una vez obtenido el *token* se realiza una primera llamada a la API usando el *token* como cabecera y especificando en el cuerpo de la petición HTTP la acción que se pretende realizar, en el caso de la transferencia habría que introducir un objeto JSON con el id de las cuentas de origen y destino, la cantidad de dinero a transferir, etc… Esta petición devolverá como respuesta un código de estado HTTP 428, en lugar del 200 (ok), que significa *Second factor required*, y en el cuerpo de la respuesta un nuevo *token*, una url y otro código. El usuario deberá acceder a esta nueva URL donde se muestra información con la acción que se pretende realizar y un campo que se deberá rellenar con un código enviado vía SMS al propio usuario, de manera que se valida el *token* anteriormente recibido. Con este proceso se ha conseguido validar el *token* como la contraseña de uso único a utilizar, por lo que sólo falta realizar una nueva llamada a la API añadiendo el

token en la cabecera, recibiendo como respuesta del servidor un código 201 que significa que la acción se ha realizado correctamente.

Este proceso, como se puede observar, resulta muy tedioso, por lo que debería emplearse únicamente en aquellas APIs que manejen datos realmente sensibles para el usuario.

5.7 EL MODELO REST

Entrando en el aspecto más tecnológico, existen diferentes modelos para desarrollar una API, a día de hoy los más empleados son SOAP, XML-RPC, JSON-RPC y REST. La ventaja que presenta REST respecto a los demás es que ofrece sencillez para la integración de otras tecnologías gracias a la utilización del protocolo HTTP, SOAP, por el contrario, aunque tiene mucha capacidad, presenta una complejidad elevada. Debido a esta razón, el modelo REST ha cobrado una gran importancia en los últimos años, pasando a ser considerado como la principal opción para la implementación de APIs. Debido a esto y a que los servicios apificados en este proyecto seguirán el modelo REST se va a explicar en qué consiste este modelo.

REST (*REpresentational State Transfer*) es un modelo de arquitectura para definir interfaces entre servicios basado en HTTP para obtener datos o realizar acciones sobre los mismos. Estos datos pueden estar en todos los formatos posibles, típicamente XML y JSON, aunque la tendencia actual es utilizar JSON. Al estar basado en HTTP, la principal característica de los servicios REST radica en el hecho de que todas las comunicaciones se realizan sin estado, ni por parte del cliente, ni del servidor, si bien es cierto que algunas implantaciones incorporan memorias cache.

A continuación se explican las principales características de este modelo:

- Es un protocolo cliente/servidor sin estado: REST funciona exclusivamente mediante el intercambio de mensajes HTTP, que tienen la característica de ser mensajes sin estado. Esto quiere decir que en cada petición que se realiza debe estar contenida toda la información necesaria para ejecutarla, sin necesidad de que ni el

cliente, ni el servidor tengan que recordar ningún estado previo. Existe, de todas maneras, la posibilidad de configurar una memoria caché para configurar el llamado protocolo cliente-caché-servidor sin estado, que permite definir ciertas respuestas por parte del cliente como cacheables, de manera que éste pueda responder con mayor rapidez.

- El formato URI (*Uniform Resource Identifier*) es una secuencia de caracteres que identifica el recurso que se apifica, es el *endpoint* que se utiliza para acceder al recurso. En el modelo REST, los recursos solamente se deben consultar o manipular a partir de su URI asociada. Resulta de vital importancia organizar los recursos disponibles de una manera jerárquica, de forma que se pueda extraer cierta información directamente al leer su URI correspondiente.

- Las operaciones que se pueden realizar con los recursos corresponden a los métodos del protocolo HTTP. Estos, además, tienen dos características muy importantes que obligarán a establecer unas medidas de seguridad u otras en función del tipo de llamada.
    - La idempotencia: Se define como idempotente aquel proceso en el cual la repetición de la misma operación varias veces genera el mismo resultado, en el caso de las APIs, el envío de una petición N veces produce el mismo resultado que el envío de la petición de la primera vez.
    - Método seguro: Se define como método aseguro aquel que no produce cambios en los recursos.

A continuación se muestra una tabla con los diferentes métodos HTTP, su funcionalidad y si presentan las mencionadas características.

Verbo	Funcionalidad	Idempotente	Seguro
GET	Obtener un recurso o listado de recursos	Sí	Sí
DELETE	Eliminar un recurso	Sí	No
PUT	Actualizar un recurso completo	Sí	No
POST	Crear un recurso	No	No
PATCH	Actualización parcial de un recurso	Sí	No

Tabla 2 – Métodos HTTP [18]

- El modelo facilita mucho una de las características principales que debe cumplir una API, el interfaz debe ser uniforme. Esto también se denomina desacoplamiento y significa que el funcionamiento interno del servicio de *backend* puede ser modificado de manera totalmente trasparente para el usuario, las características del interfaz del servicio deben ser una especie de contrato con el usuario que no se debe modificar aunque los procesos internos cambien. El usuario puede realizar únicamente las acciones concretas que permite HTTP sobre los recursos que definan las URI.

- Uno de los principios característicos de los servicios REST, y que ha supuesto una de las claves de su auge, es el concepto que se ha denominado como HATEOAS (*Hypermedia As The Engine Of Application State*), al cual podríamos considerar como una evolución del clásico concepto del hipertexto, ya que HATEOAS define que al realizarse una petición a un servidor, parte de la información que se devuelva consistirá en hipervínculos de navegación asociada a otros recursos. Gracias a este modelo, un cliente, a partir de una URI de entrada genérica al servicio del servidor, podrá descubrir otros recursos utilizando únicamente la información devuelta en las respuestas. [19]

- La última característica a mencionar es la paginación, esto no es una característica como tal, sino una herramienta más de la que se puede hacer uso a la hora de

desarrollar servicios REST. Se emplea en el caso de que al realizar una llamada *get*, la respuesta devuelva contenga una cantidad de información elevada, en este caso, al trabajar sobre un protocolo ligero como es HTTP, la respuesta del servicio puede verse penalizada. La herramienta se sirve del método HATEOAS explicado en el punto anterior para fragmentar la información de manera que para recibir la información de un solo recurso se deban hacer varias llamadas *get* más ligeras en lugar de realizar una sola excesivamente pesada. A continuación se explica el funcionamiento de este método.

En primer lugar, al cliente realiza una petición *get* normal a un recurso, en caso de que este sea demasiado grande, el servidor, en lugar de enviar un código 200, envía un 206, que significa *partial content*. En este caso, en la respuesta, junto a este contenido parcial, se devolverá cierta información adicional como las URIs de acceso al primer fragmento de la información, al último, a la página anterior, a la siguiente, el número total de páginas y el número de la página actual. Gracias a esta información, el cliente no tiene más que ir accediendo a las URIs que se le vayan devolviendo en cada llamada hasta completar la información. A continuación se incluye un ejemplo de llamada paginada:

```
"pagination": {
        "links": {
            "first":
"https://www.bbva.com/accounts/v0/accounts/1234/transactions?orderBy=postedDate",
            "last":
"https://www.bbva.com/accounts/v0/accounts/1234/transactions?orderBy=postedDate&p
aginationKey=gdfw254sf",
            "previous":
"https://www.bbva.com/accounts/v0/accounts/1234/transactions?orderBy=postedDate&p
aginationKey=dwer342sf",
            "next":
"https://www.bbva.com/accounts/v0/accounts/1234/transactions?orderBy=postedDate&p
aginationKey=qazx687sf"
        },
        "page": 1,
        "totalPages": 3,
        "totalElements": 6,
        "pageSize": 2
    }
```

5.8 LENGUAJES DE DISEÑO DE APIS

Los lenguajes de diseño de APIs son una herramienta fundamental para el desarrollo de las mismas. Un correcto modelado mediante estas herramientas permite diseñar, desarrollar, probar y documentar las APIs. A continuación se nombran las principales.

- Swagger

Consiste en un *framework* de especificación e implementación que permite describir, producir, consumir y visualizar servicios RESTful. Su objetivo general es que la documentación y los sistemas clientes se actualicen al mismo tiempo que la parte del servidor

- RAML

Es un lenguaje para describir RESTful APIs basado en YAML. Proporciona una especificación para describir APIs. Es capaz de definir APIs que no obedecen todas las restricciones de RESTFul.

Su objetivo es fomentar la reutilización, permitir el descubrimiento y la definición de patrones compartidos.

- Apiblueprint

Consiste en un lenguaje de descripción de APIs web de alto nivel. Su especificación es sencilla y permite describir las APIs, generar documentación de forma automática, definir el ciclo de vida y crear prototipos de las APIs.

A pesar de que los tres lenguajes están cobrando una gran importancia en el mundo de las APIs, la herramienta elegida por el departamento para el modelado de APIs es RAML, por ello se hará uso de ella para modelar las APIs de los servicios del proyecto y a continuación se profundiza algo más en ella.

- Ventajas: Especificación simple, legible y de alta calidad. Soporte para la mayoría de esquemas JSON y W3c XML. Soporte para la mayor parte de lenguajes. Soporta patrones que ayudan a la reutilización.
- Desventajas: No tiene una iniciativa de estandarización como en caso de Swagger. Carece de herramientas a nivel de código.

	Swagger	**RAML**
Formato	JSON y YAML	YAML
Representación de metadatos	Subconjunto de JSON Schema	Cualquiera
Autenticación	Basic, API-Key, OAuth 2	Basic, Digest, OAuth 1 y 2
Especificación	Difícil de leer y larga	Fácil de leer
Lenguajes soportados	Clojure, Go, JS, Java, .Net, Node, PHP, Python, Ruby, Scala	JS, Java, Node, PHP, Python, Ruby
Open Source	100% Open Source	Respaldado por una gran comunidad de código abierto.

Tabla 3 – Comparativa entre Swagger y RAML [20]

6. SISTEMA DESARROLLADO

6.1 APIFICACIÓN DEL SISTEMA RDR

En este apartado se explica la manera en la que se han expuesto los servicios comentados mediante servicios RESTful.

A continuación se incluye un esquema con el diseño global del sistema de apificación de los servicios de RDR.

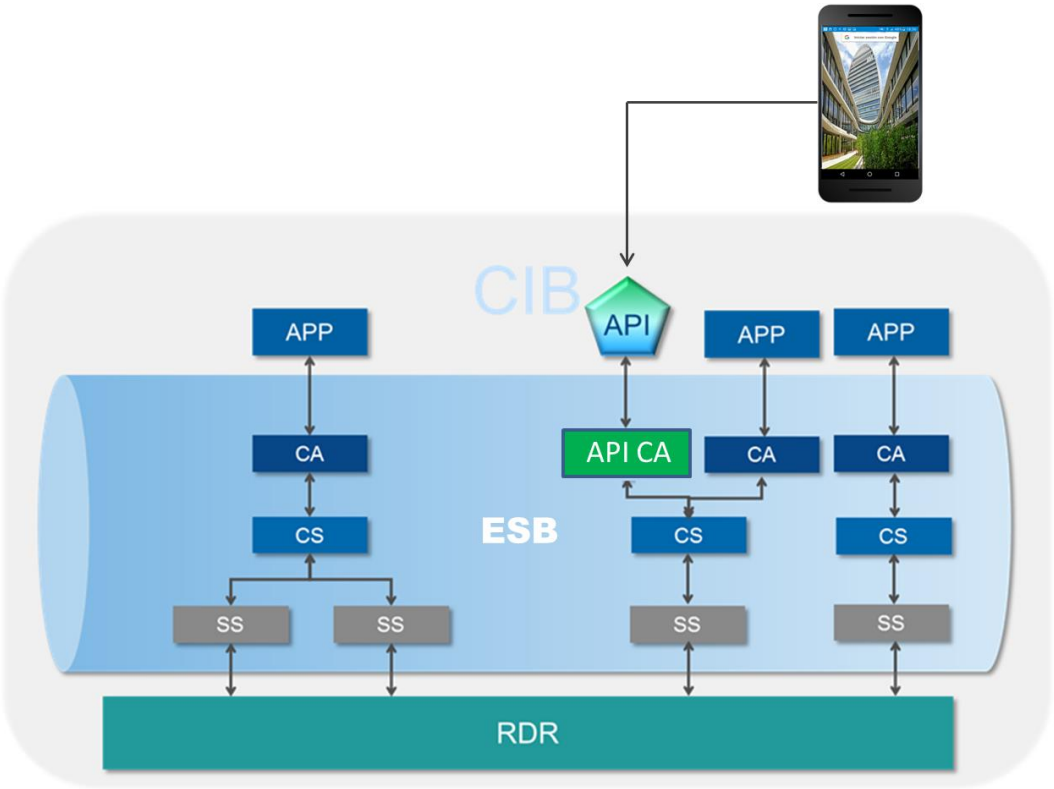


Figura 13 – Esquema del diseño global del sistema.

En la parte inferior se representa el sistema de *backend* RDR que se pretende apificar. Este sistema se encuentra conectado directamente al bus de integración, que es la manera en la que a día de hoy se puede hacer uso a estos servicios, sin embargo, los mensajes en el ESB, como se ha comentado, se encuentran en un formato propio del BBVA, conocido como el modelo canónico de datos (MCD), sin embargo, el objetivo consiste en que la API devuelva los datos en formato JSON mediante un sistema REST, por ello, se deben desarrollar unos adaptadores de canal que transformen los mensajes provenientes del ESB del MCD al formato JSON que será expuesto mediante la API, y viceversa. Estos adaptadores de canal se implantarán en unos servidores de desarrollo del banco a los cuales se conectarán las aplicaciones que hagan uso de esta API. Es el caso de la aplicación Android que se desarrollará, la cual se representa con la pantalla situada en la parte superior derecha de la figura, y que hará uso de la API desarrollada y por tanto, de los servicios de *bakend*, que dentro de la cadena de valor de la API corresponderían a los activos apificados.

Una de las principales características que se debe lograr que tenga una API que se diseñe es que sea lo más sencilla posible de integrar con las aplicaciones que hagan uso de ella, para ello siempre se deben buscar soluciones que se adapten lo máximo posible a estándares definidos en el caso de que estos existan. Esta característica se hace aún más necesaria en el caso de manejar datos financieros, los cuales suelen ser complejos y están atados a muchas regulaciones.

A pesar de que los datos de los servicios tratados en este proyecto se encuentran estructurados según el formato FiXML, dentro del proyecto se ha optado por traducir estos datos a otro estándar, el ISO 20022, debido a que este supone una solución más de conjunto ya que surge como una solución única a diferentes formatos utilizados para diferentes procesos de negocio como FIX, FpML, MDDL, FiXML o RIXML. [23]

6.1.1 SERVICIO DE CONSULTA DE CONTRAPARTIDAS

El principal obstáculo para convertir este servicio de *backend* en un servicio REST es el tamaño de la respuesta. Ante una petición, el servicio original devuelve una cantidad

enorme de campos con la información referente a la contrapartida especificada, relacionados con la información general de la contrapartida, los códigos internos y externos asociados a la misma, la jerarquía interna, información acerca de los grupos a los que pertenece, información asociada a la oficina a la que pertenece e información regulatoria y sobre datos fiscales entre otros. El intercambio de tal cantidad de información utilizando el protocolo HTTP, a pesar de la posibilidad de la utilización del método de paginación, puede provocar una ralentización bastante elevada del servicio, ya que se trata de un protocolo ligero, repercutiendo este hecho en la experiencia del cliente.

Por otra parte, se ha pretendido utilizar el estándar ISO 20022 para modelar los mensajes de intercambio de datos de estos servicios. En concreto, para el servicio de consulta de contrapartidas se ha empleado el esquema tsin.002.001.01. [24], (*trade service initiation*), el cual, según la página del ISO se emplea para informar a un demandante financiero acerca del estado de peticiones o cancelaciones de peticiones de carácter financiero.

Este esquema es un archivo XML que describe la estructura de datos que define a las entidades correspondientes, en nuestro caso las contrapartidas. El esquema es notablemente más ligero que las respuestas que devuelve el servicio, ya que tiene muchos menos campos. Se han descartado, por tanto, muchos de los campos del servicio original.

A partir de este esquema se ha modelado el cuerpo de la respuesta que devolverá la API. A continuación se incluye un diagrama de clases que representa un esquema del cuerpo devuelto por una petición.

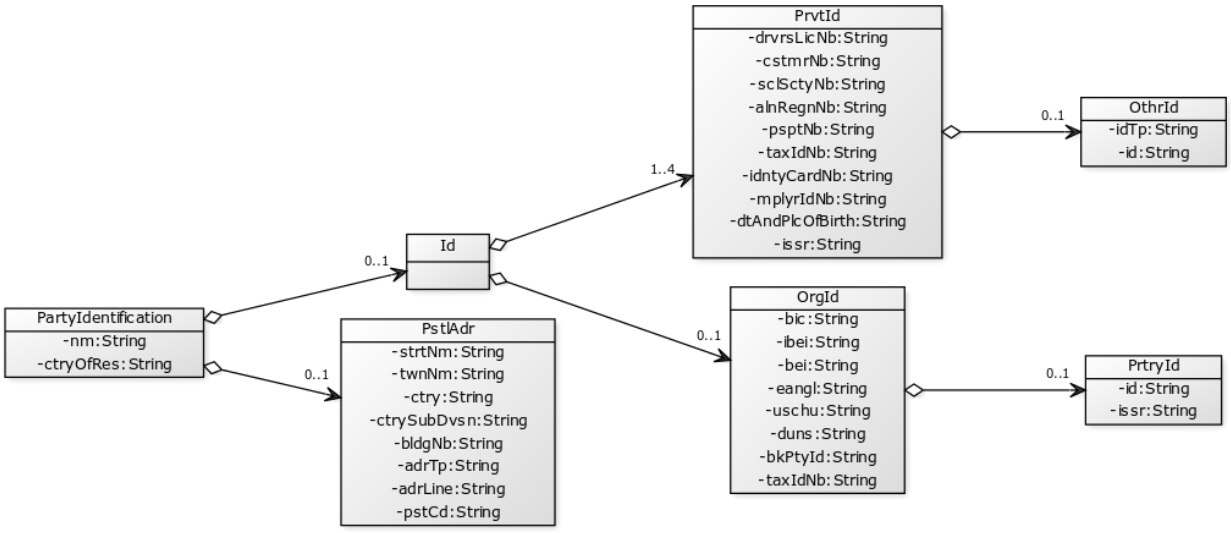


Figura 14 – Diagrama de clases UML de la respuesta del servicio de consulta de contrapartidas.

Como se puede apreciar en la imagen, la respuesta *PartyIdentification* tiene 4 campos principales, el nombre de la persona, entidad, etc…, su país de residencia, un identificador y una dirección postal. Los dos últimos a su vez se componen de otros campos, en concreto, en función de si trata por ejemplo de un banco, se devolverá un código, o si trata de una persona física se devuelve el número del documento de identidad o el carnet de conducir. Debido a que cada contrapartida puede ser de un tipo diferente y por tanto

La API de *parties* ofrece a los desarrolladores la manera de recuperar la información de los clientes (nombre, email, dirección fiscal, documento de identidad, teléfono…) de forma actualizada y fiable evitando tener que pedir que se rellenen formularios por su parte.

A continuación se incluye un ejemplo de la respuesta a una petición hecha al servicio utilizando los valores de Id de la contrapartida “101” e Id del *Issuer* “45”. Como se ha comentado, los campos no son obligatorios, y dependerá del tipo de contrapartida que se devuelvan unos u otros. En este caso se obtienen el nombre, país, algunos campos de la dirección postal y los IDs de la propia contrapartida y su entidad asociada según la propia estructura de datos de BBVA.

```
{
 "PartyIdentification": {
   "Nm": "CONSTRUCCIONES Y OBRAS LLORENTE S.A.",
   "PstlAdr": {
     "StrtNm": "ALUMINIO 0017 PARC 221",
     "PstCd": "47012",
     "TwnNm": "VALLADOLID",
     "Ctry": "ES"
   },
   "Id": {
     "OrgId": {
       "PrtryId": {
         "Id": "101",
         "Issr": "45"
       }
     }
   },
   "CtryOfRes": "ES"
 }
}
```


Los archivos RAML completos de descripción de las tres APIs se encuentran adjuntos en el anexo C.

6.1.2 SERVICIO DE CONSULTA DE JERARQUÍA INTERNA

El servicio de jerarquía interna se ha modelado según el diagrama de clases que se muestra a continuación.

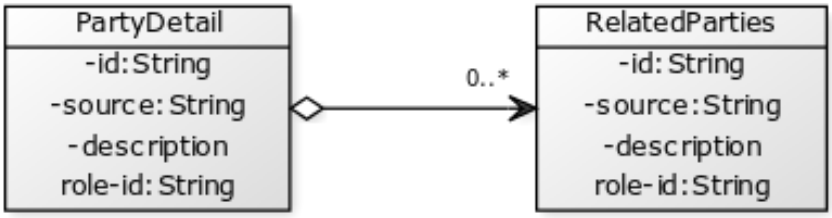


Figura 15 – Diagrama de clases UML del objeto Java bean de las instancias de internal structure.

Los mensajes, por tanto, estarán compuestos por cuatro datos de la propia contrapartida pedida (Id, fuente, descripción y rol), y las contrapartidas relacionadas que tendrán los mismos campos. La relación entre ambas consiste en que la contrapartida pedida podrá

tener 0, 1 o varias contrapartidas relacionadas. Comentar que en esta prueba de apificación se ha descartado el campo de tipo de jerarquía de las contrapartidas relacionadas que podría resultar interesante.

Los parámetros de consulta son tres: el código identificador de la contrapartida pedida, su rol, que podrá ser un valor de una lista dada: grupo, matriz, entidad legal, empresa, *branch*, oficina y agente de cálculo; y el tipo de jerarquía que se solicita: empresa, *branch*, oficina, *portfolio*¸ jerarquía superior o inferior.

Se debe comentar, por último, que en este caso no se ha logrado adaptar los datos del servicio al estándar ISO 20022 debido a la inexistencia de un esquema que se parezca al modelo de datos del servicio.

A continuación se incluye un ejemplo de la respuesta a una petición hecha al servicio utilizando los valores de Id de la contrapartida "PORT00000067", Rol "*Legal entity*" y tipo de jerarquía inferior. Se reciben en primer lugar los 4 campos con la información referente a la propia contrapartida pedida. En este caso se recibe una contrapartida relacionada, se puede ver que los datos se encuentran dentro del elemento de un *array* del objeto JSON, en caso de que hubiese más elementos relacionados, se situarían a continuación del mismo. Los datos recibidos del elemento relacionado son los mismos que los de la contrapartida padre: Id, fuente, descripción y rol.


```
{
  "ID" : "PORT00000067",
  "Source" : "O",
  "Description" : "ALBBVA",
  "Role" : "PORTFOLIO",
  "RelatedParties" : [{
    "ID" : "455",
    "Source" : "O",
    "Description" : "TESOR. MADRID",
    "Role" : "CIBOFFI"
  }]
}
```

6.1.3 SERVICIO DE CONSULTA DE CALENDARIOS

Se ha utilizado el esquema que se aprecia en el siguiente diagrama de clases para modelar los datos del servicio de consulta de calendarios.

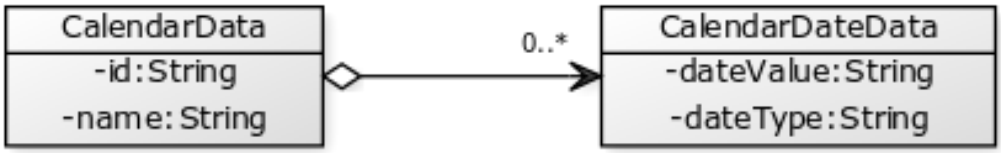


Figura 16 – Diagrama de clases UML del objeto Java bean de las instancias de calendars.

En este caso, el esquema presenta pocos datos, por lo que el servicio resultará muy ligero, lo cual resulta ideal para un servicio RESTful.

Se tienen dos elementos, el primero corresponde a los datos del propio calendario pedido, en concreto un código identificador del mismo, y un nombre identificativo.

El siguiente elemento corresponde a las fechas no laborables recibidas, cada una de las fechas tendrá dos valores, uno correspondiente al valor de la propia fecha (en formato YYYYMMDD), y el segundo al tipo de fecha no laborable (los valores pueden ser 1: fin de semana, 2: festivo, 3: otro día no laborable). Estos elementos aparecerán como un *array* de tamaño ilimitado.

Los parámetros de consulta son: un código identificador del calendario, las fechas de inicio y fin del rango de búsqueda (en formato YYYYMMDD también).

A continuación se incluye un ejemplo del objeto JSON de respuesta a una consulta al servicio habiendo introducido los valores “EUR” correspondiente al calendario europeo, y las fechas 24/02/2017 y 27/02/2017. Se puede observar que se reciben en primer lugar los dos datos asociados al calendario, seguidos de un *array* con las fechas no laborables devueltas.

```
{
  "ID" : "EUR",
  "Name" : "EUR",
  "CalendarDateData" : [{
    "DateValue" : 20170225,
    "DateType" : 2
  },{
    "DateValue" : 20170226,
    "DateType" : 2
  }]

}
```

6.2 DESARROLLO DE LA APLICACIÓN ANDROID CONSUMIDORA DE LOS SERVICIOS APIFICADOS

6.2.1 SISTEMA DE AUTENTICACIÓN

El primer paso para el desarrollo de una app de estas características, es decir, orientada a un tipo muy específico de usuario, en este caso serían usuarios internos del banco o de empresas colaboradoras, será la implantación de un sistema de autenticación por parte del usuario. Al ser los servicios de consulta, tal y como se ha explicado en el capítulo 5 en el apartado de seguridad, el proceso no es crítico y por tanto, el proceso de autenticación podría ser ligero, utilizando un *token* de acceso por ejemplo. Sin embargo, la configuración de un servidor de autenticación y la implantación de un protocolo como OAuth es un proceso complejo que queda completamente fuera del alcance de un proyecto de estas características. De todas maneras, al tratarse de un proyecto de prueba de concepto, con la intención de hacerlo lo más parecido posible a un caso de uso práctico real, se ha decidido emplear el término medio entre un sistema de seguridad que pudiera implementar una entidad de las características de BBVA y no realizar proceso de autenticación alguno, utilizar el servicio de Google de autenticación.

El servicio de Google Sign-In se describe en su página como un sistema de autenticación seguro que permite reducir la carga del proceso de *log-in* a los usuarios permitiéndoles que realicen este proceso usando su propia cuenta de Google. [25]

A continuación se explica el proceso de integración de este servicio con la app. El primer paso consiste en registrar la cuenta del desarrollador y la app en la página de desarrolladores de Google, para ello se necesita generar el SHA-1 (*Secure Hash Algorithm*) de un certificado de acceso para poder crear un cliente OAuth2 y una API *key* para la app.

Este paso es sencillo y tan solo se necesita un ordenador con el java *development kit* y Android Studio instalados. El certificado necesario se crea automáticamente en un directorio al instalar Android Studio en el ordenador. Para cifrarlo y obtener el SHA-1 se utiliza la herramienta *keytool* provista por java.

Para evitar mayores complicaciones se ha utilizado el certificado de *debug* para lo cual se evita tener que configurar una cuenta.

Para obtener el SHA-1 se abre una ventana de comandos y se ejecuta la siguiente instrucción.

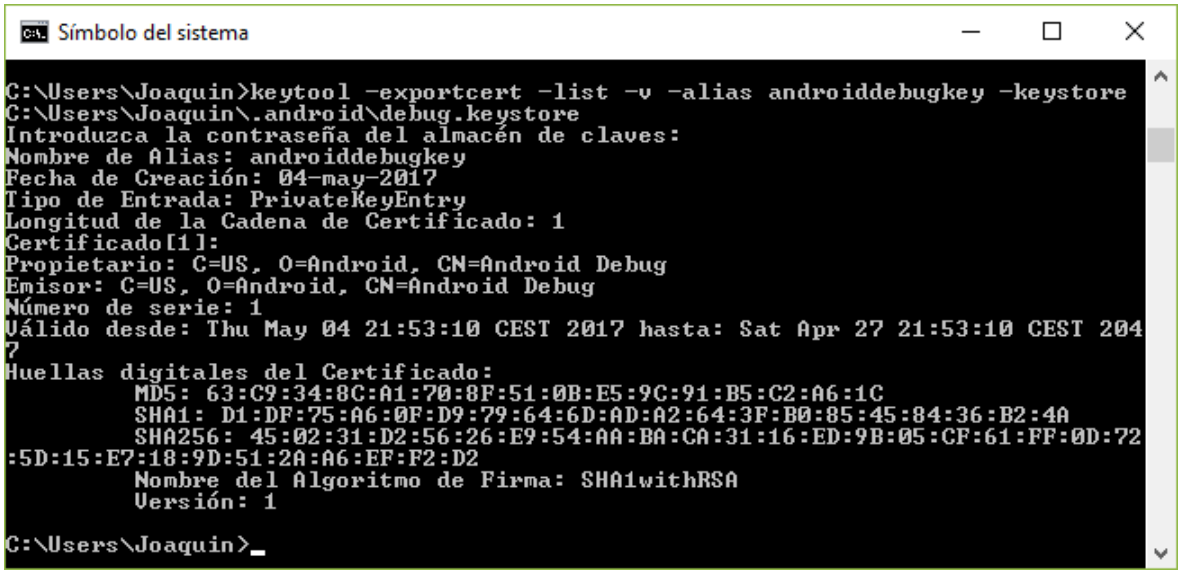


Figura 17 – Obtención del SHA-1 del certificado para la creación del cliente OAuth2 y la API key de los servicios de Google.

Como se ve en la imagen, se obtienen varias firmas, entre las cuales seleccionamos la SHA-1, a continuación se introduce la cadena en el siguiente campo de la página de desarrolladores de Google.

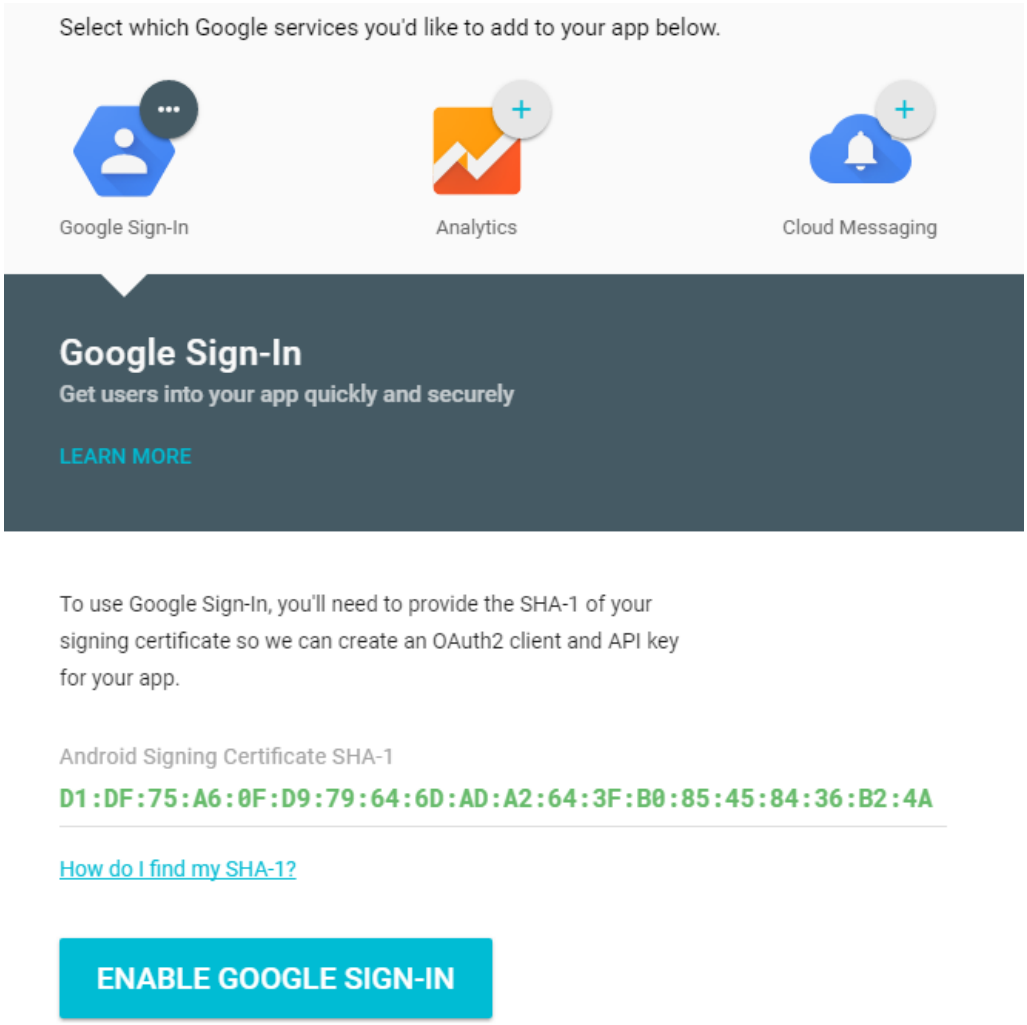


Figura 18 – Obtención del JSON de configuración de la API de Google.

Pulsando en *Enable Google Sign-in* se registra el nombre de la aplicación en el servicio de Google (en este caso RDR Pilot), y a continuación se da la opción de descargar un fichero JSON de nombre *google-services.json*, que contendrá información entre la que se encuentran el nombre e id de la aplicación, el nombre del paquete que se ha utilizado, información del cliente OAuth, los permisos de autenticación y la API *key*. Este fichero se deberá incluir dentro de la estructura del proyecto y lo utilizará el servicio de Google para identificar la aplicación y el tipo de autenticación a efectuar al realizar las llamadas contra sus servidores.

6.2.2 ESQUEMAS DE CASOS DE USO Y DE SECUENCIA

A continuación se incluye un diagrama de casos de uso UML con la funcionalidad desarrollada.

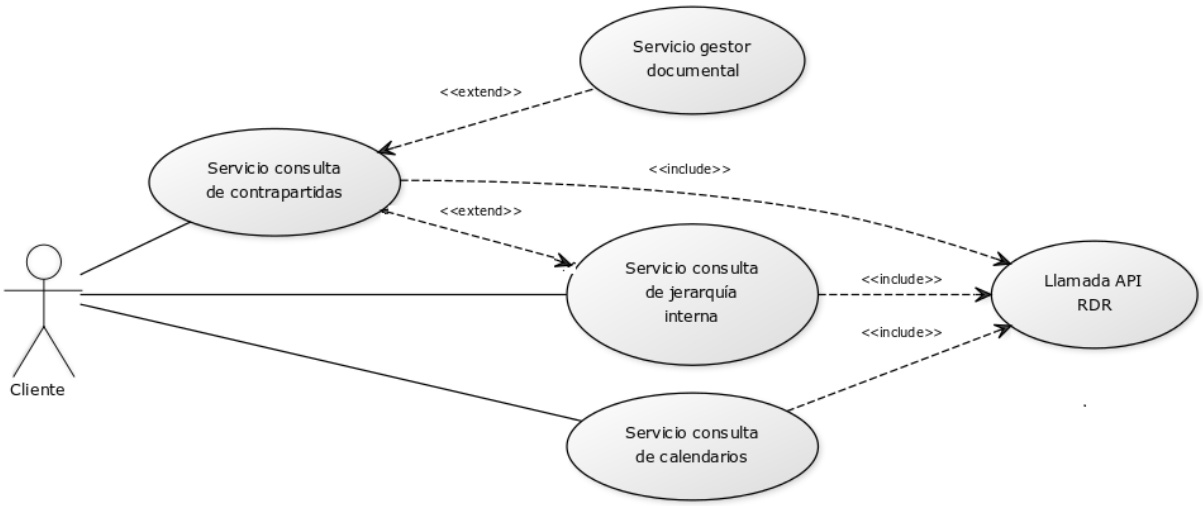


Figura 19 – Diagrama de casos de uso de la app.

El usuario, una vez se ha autenticado, tendrá tres casos de uso diferentes, como se puede ver en el esquema, cada uno de ellos correspondiente a uno de los servicios apificados, estos casos de uso corresponden a realizar una consulta a los servicios de RDR por medio de las APIs desarrolladas.

El servicio de consulta de contrapartidas, además, incorpora algo de funcionalidad añadida, la utilización del gestor documental para informar mediante el envío de un correo electrónico sobre la consulta realizada a un usuario determinado.

El servicio de jerarquía interna también presenta funcionalidad añadida respecto a la ofrecida por la propia API, pues una vez recibida la información de las contrapartidas relacionadas, el servicio permite realizar una consulta de información más detallada de la misma realizando una llamada al servicio de contrapartidas con los datos obtenidos.

También se incluye a continuación un diagrama de secuencia UML que representa el flujo de datos que se produce en una llamada a uno de los tres servicios.

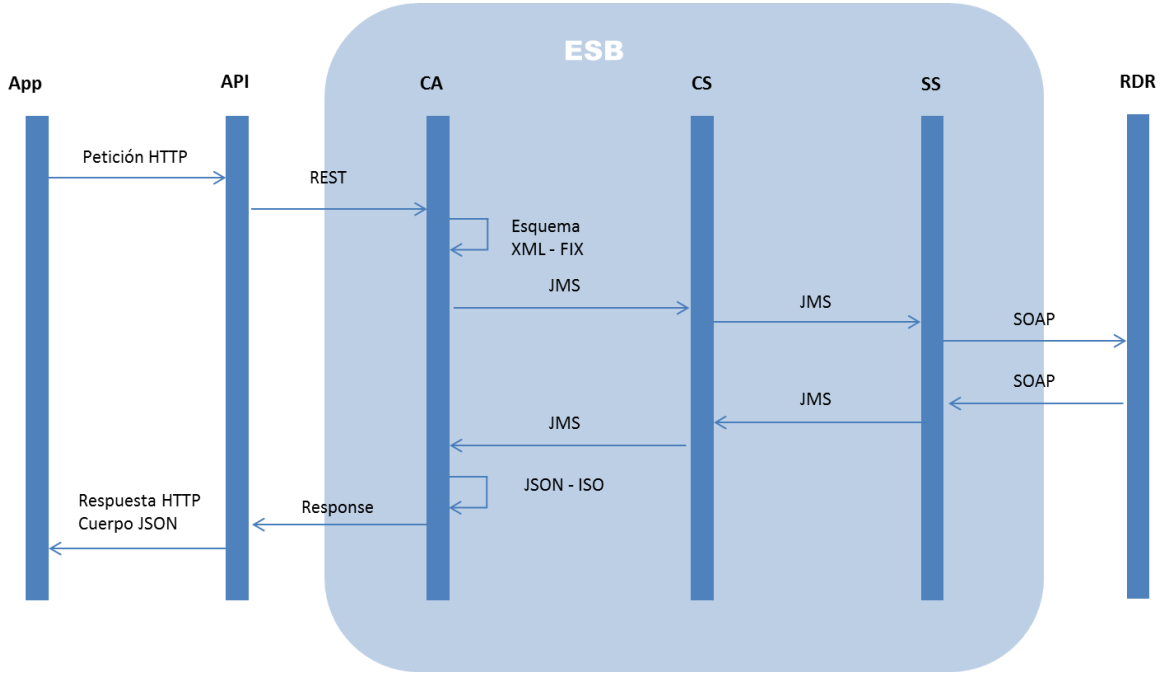


Figura 20 – Diagrama de secuencia de consulta de una petición a la API de consulta de contrapartidas.

El flujo de datos representado es el siguiente, una vez que el usuario ha introducido los parámetros de búsqueda y pulsado el botón de realizar la petición, la app se conecta a la API mediante una petición HTTP concatenando la URL del servidor donde se ha instalado la API, el puerto TCP en el que está escuchando, el nombre del recurso correspondiente al recurso, y los parámetros de búsqueda. La API a su vez se conectará al bus de integración

por medio de un elemento llamado el *channel adapter*, este elemento traduce el formato de datos REST al formato de datos usado en el ESB (XML y FiX). Una vez realizada la transformación, el adaptador de canal envía los datos a una serie de servicios internos del bus que dirigirán la información a los servicios de *backend*. Antes de hacer la petición directa al servicio de RDR el bus debe realizar una nueva transformación del mensaje, en este caso a la tecnología SOAP utilizada por el servicio. Una vez realizada la petición, la respuesta seguirá el camino inverso hasta el adaptador de canal de la API, en el cual se transformará la respuesta XML recibida al objeto JSON definido en los RAML. Este objeto con los datos de la respuesta se incluirá en la respuesta de la conexión HTTP entre la app y el adaptador de canal.

6.2.3 IMPLEMENTACIÓN DE LAS ACTIVIDADES

A continuación se explicará la implementación de cada una de las actividades más en detalle.

6.2.3.1 Pantalla de inicio

El diseño de la pantalla que aparece al iniciar la aplicación se compone únicamente de una imagen de fondo y un botón para iniciar el proceso de autenticación. A continuación se incluye el diseño de la pantalla.

Figura 21 – Diseño de la pantalla de autenticación de la aplicación móvil.

A continuación se incluyen y explican las principales líneas de código que explican la lógica de la actividad.

Al iniciar la actividad simplemente se inicializan las opciones de autenticación de google, utilizando la opción estándar y pidiendo el *email*, mediante la siguiente línea de código.

```
GoogleSignInOptions gso = new
GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
                .requestEmail()
                .build();
```

Y se instancia la variable de clase GoogleApiClient en la que a continuación se almacenará la cuenta que intente validarse.

```
GoogleApiClient mGoogleApiClient = new GoogleApiClient.Builder(this)
                .enableAutoManage(this /* FragmentActivity */, this /*
OnConnectionFailedListener */)
                .addApi(Auth.GOOGLE_SIGN_IN_API, gso)
                .build();
```

Una vez realizados estos pasos, la actividad se queda a la espera de que el usuario pulse el botón de iniciar sesión con Google.

Una vez que se pulsa, el programa lanza una actividad distinta, perteneciente a Google y que se encargará de validar la cuenta, mediante las siguientes dos líneas.

```
Intent signInIntent = Auth.GoogleSignInApi.getSignInIntent(mGoogleApiClient);

startActivityForResult(signInIntent, RC_SIGN_IN);
```

La actividad de Google aparece sobre la pantalla de la app como se muestra en la imagen.

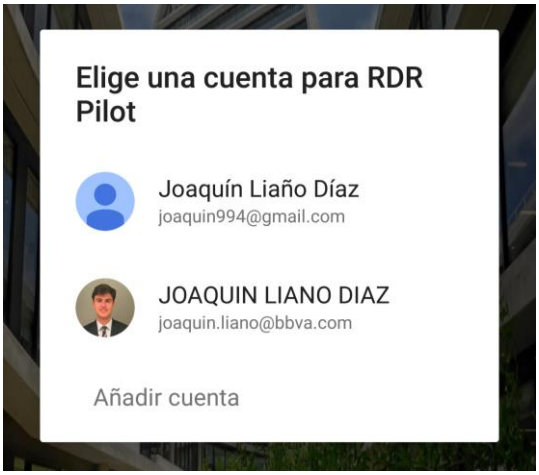


Figura 22 – Selección de cuenta de usuario para la autenticación.

Esta actividad pedirá el usuario y contraseña al usuario. Una vez introducidas y realizado el proceso de autenticación, se maneja el resultado del proceso mediante el método *@Override onActivityResult()*. Donde se introduce la siguiente instrucción.

```
GoogleSignInResult result = Auth.GoogleSignInApi.getSignInResultFromIntent(data);
```

En esta variable se almacena el resultado del intento de autenticación. Llamando al método isSuccess() de la variable, en caso de que el resultado sea *true*, la autenticación se ha llevado a cabo correctamente y podemos lanzar la actividad del menú principal. Antes de lanzar la nueva actividad, guardamos el *nick* de la cuenta, el nombre y la foto de perfil, a las cuales se accede mediante los métodos .getPhotoUrl(), getDisplayName() y getEmail(), para ser tratados en la siguiente actividad. Los almacenamos mediante el método putExtra().

En caso de que el método isSuccess() devuelva un *false*, se muestra el mensaje *Login failed*, y el usuario permanece en la misma pantalla, pudiendo realizar un nuevo intento de autenticación.

6.2.3.2 Menú principal de la aplicación

El diseño del menú principal de la aplicación consistirá en una pestaña que se desplegará desde la parte izquierda de la pantalla al pulsar el icono de arriba a la izquierda.

La pestaña estará dividida en dos partes, en la parte superior se mostrará la información referente a la cuenta de Google con la que se ha autenticado el usuario, en concreto se mostrarán la foto de su perfil, su nombre y su correo.

En la parte inferior de la pantalla se mostrará una lista con los servicios disponibles y un fragmento de ayuda. Pulsando sobre el nombre de uno de los servicios se accede al mismo. En la siguiente imagen se muestra el diseño de la página del menú.

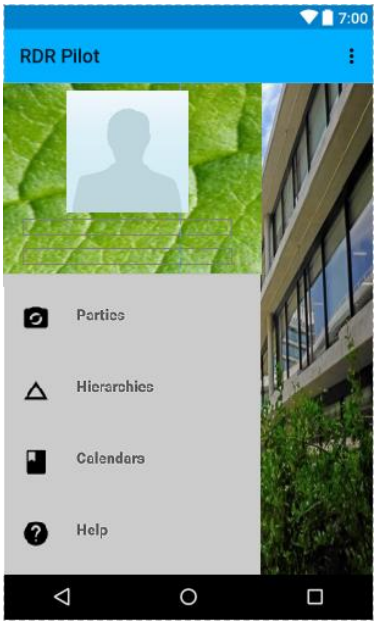


Figura 23 – Diseño de la pantalla de menú de servicios.

A continuación se explica la lógica de esta pantalla. En primer lugar se recuperan la foto, el nombre y el correo, que habían sido pasados como parámetros en la actividad anterior. Para esta parte de la vista se ha creado un recurso *layout* aparte del recurso principal de la actividad en el cual se cargarán estos tres datos. Mediante la clase *LayoutInflater* insertamos este fragmento del diseño en la pantalla.

Para rellenar la lista de servicios se ha utilizado una clase llamada *CustomAdapter* que hereda de la clase *BaseAdapter*. Desde la actividad principal se crea una lista java *List* que guarda objetos que contienen el nombre del servicio y el icono mostrado, y pasando la lista a la clase *CustomAdapter*, esta se encarga de colocar estos dos datos en la pantalla.

Los pasos anteriores son los que configuran la vista inicial, una vez finalizados, la aplicación queda a la espera de que el usuario pulse sobre uno de los servicios. Para lograr esto se hace uso del método *setOnItemClickListener()* que detecta cuando el usuario pulsa sobre un elemento de la lista y devuelve la posición del elemento pulsado. Como la lista es estática, este proceso no tiene mayor complicación, simplemente se le asigna el índice 0 al

servicio de *parties*, 1 a *hierarchy*, 2 a *calendars* y 3 al fragmento de ayuda. Se hace un simple *switch* con el índice y se inicia la actividad correspondiente, en este caso no es necesario pasar ningún parámetro a la nueva actividad.

6.2.3.3 Implantación del proceso de conexión con las APIs

Tal y como se ha explicado, la forma de hacer una petición a cualquiera de los tres servicios es mediante una petición HTTP. Como el proceso es muy similar para los tres casos se explica en primer lugar cómo se realiza la conexión de manera genérica y posteriormente se explica cómo se concatena la URL de cada servicio en particular.

Para realizar las peticiones a las APIs se han implementado una serie de clases cuya relación queda representada en el siguiente diagrama UML.

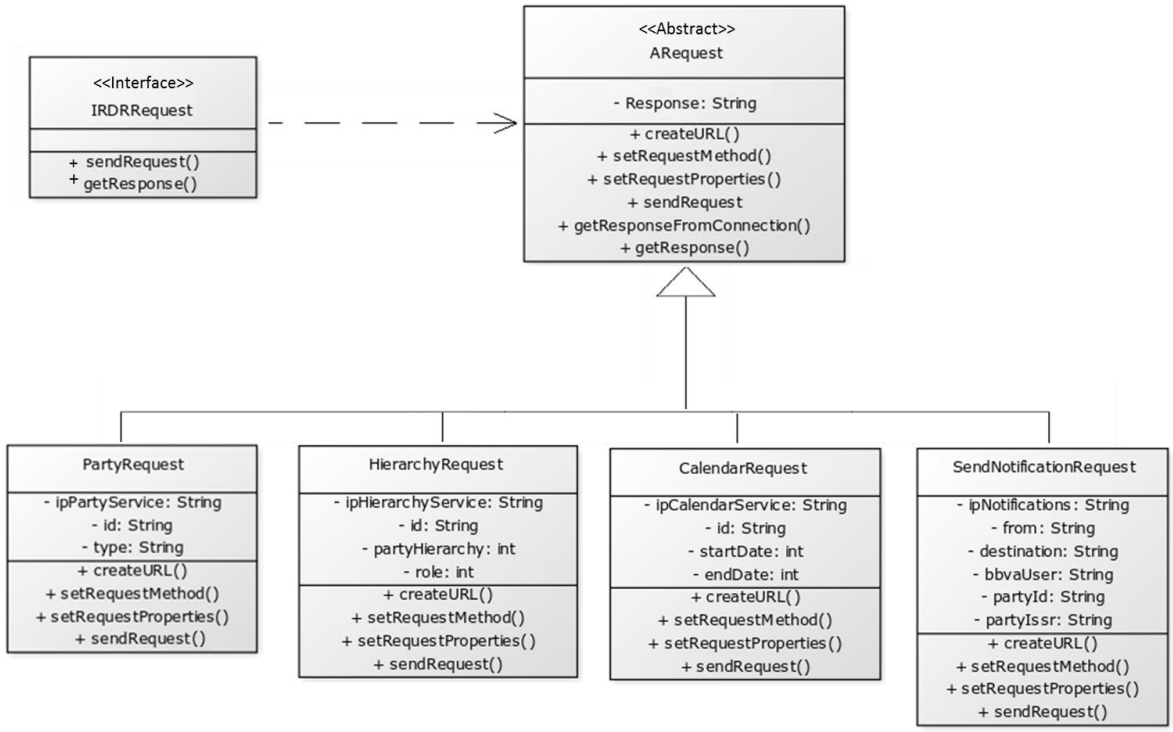


Figura 24 – Diagrama de clases UML del sistema encargado de realizar la conexión HTTP con la API.

En primer lugar se ha implantado un interfaz llamado IRDRRequest que implementa los métodos sendRequest() que lanza la petición y almacena la respuesta recibida, y el método getResponse(), con el que se recibe el objeto de la respuesta en formato String, desde el hilo principal de la aplicación.

Desde la aplicación, se puede hacer una instancia genérica del *interface*, y a continuación hacer la instanciación de la clase con el tipo de petición específico según el servicio.

La clase abstracta ARequest implementa la lógica de los dos métodos de la interfaz, la lógica de una petición genérica. El método sendRequest() en primer lugar crea la url correspondiente, como ésta varía de un servicio a otro, este método se delega a las clases hijas. Para el proceso de conexión se utilizan principalmente dos clases, las clases URL y HttpURLConnection. Una vez concatenada la URL, se almacena como instancia de la clase URL y se inicia la conexión con el servidor mediante el método openConnection(), de la propia clase, y que devuelve una instancia de la clase HttpURLConnection.

Mediante los métodos disponibles de esta clase configuramos las características de la conexión, en concreto, un *timeout* de 2 segundos, GET como método HTTP y *application/json* como tipo de respuesta.

A continuación se ha definido el método getResponseFromConnection(), que procesa la respuesta recibida, convirtiéndola en un String mediante las clases BufferedReader e InputStreamReader. La respuesta es de tipo JSON como se comentó y se almacena como variable local de la clase en formato String.

Para recuperar la respuesta desde el hilo de ejecución de la aplicación una vez que se ha ejecutado el método sendRequest(), se llama al método getResponse(), el cual simplemente devuelve el String almacenado en la variable *response*.

A continuación se describen las clases que heredan de la clase ARequest. Existe una por cada servicio, es decir, las clases PartyRequest, HierarchyRequest, CalendarRequest y SendNotificationRequest. Estas clases implementan el método que concatena la URL específica de servicio con los parámetros introducidos en cada caso. La URL se compone de la dirección IP del *host* donde se ha configurado la API, que se ha omitido en este documento, el puerto TCP en el que escucha el servidor, el nombre del recurso y los parámetros de consulta.

A continuación se incluye la URL del servicio de contrapartidas, el nombre del recurso es /parties y los parámetros de consulta son Id e Issr. Estos datos se encuentran en los archivos RAML generados.

```
http://***ip de la API de parties***:10655/parties?Id=***Id
introducido***&Issr=***Issr introducido***
```

El nombre del recurso de la API de jerarquía interna es /internalStructure y los parámetros ID, PartyHierarchy y Role.

```
http://***ip de la API de jerarquía***:10658/internalStructure?ID=***Id
introducido***&PartyHierarchy=***Tipo de jerarquía***&Role=***Rol  introducido***
```

El nombre del recurso de la API de calendarios es /calendars y los parámetros son ID, StartDate y EndDate.

```
http://***ip de la API de calendarios***:10672/calendars?ID=***Id
introducido***&StartDate=***fecha de inicio***&EndDate=***fecha de fin***
```

Por último, para enviar la notificación se debe acceder al recurso /notifications/send e incluir los parámetros que indiquen el *mail* de notificaciones del departamento, el *mail* del usuario al que se envía la notificación, el id del usuario que ha realizado la consulta y algunos de los datos de la contrapartida consultada.

```
http://***ip de la API de GN***:43001/notifications/send?from=***mail de
notificaciones***&to=***mail de destino***&bbva_user=***usuario que ha realizado
la petición***&id=***Id de la party***&issr=***Issr de la
party***&name=***name***&streetName=***Street name***&postalCode=***postal
code***&townName=***town
name***&country=***country***&countryOfResidence=***country of residence***
```

6.2.3.4 Implantación del servicio de consulta de contrapartidas

Cada uno de los servicios consta de dos pantallas, una para introducir los datos de la *request*, y otra que muestra los datos recibidos en la respuesta.

El diseño de la interfaz de introducción de los parámetros de consulta del servicio de contrapartidas es muy sencillo, incluye simplemente dos campos de texto en los cuales el usuario introducirá los dos parámetros de consulta (Id e *Issuer* de la contrpartida); y dos botones, uno para realizar la consulta una vez introducidos los parámetros y otro para limpiar los dos campos de texto en caso de que el usuario los introduzca erróneamente.

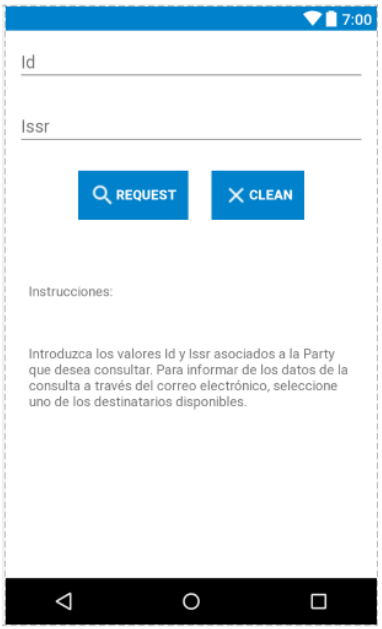


Figura 25 – Diseño de la pantalla de consulta del servicio de parties.

La lógica de esta actividad tampoco es complicada. Mediante el método requestButtonOnClick(), el hilo queda a la escucha que se pulse el botón *request*. Una vez invocado este método, se lee el texto de los dos campos de texto mediante la clase EditText (se comprueba que no sean campos vacíos, en cuyo caso el método se interrumpe y el programa queda nuevamente a la escucha) y con los dos valores se crea una instancia de la clase PartyRequest ya comentada, con la cual se gestionará la conexión con el servidor. La instancia se realiza mediante la siguiente línea de código.

```
IRDRRequest request = new PartyRequest(
                    partyId.getText().toString(),
                    partyType.getText().toString(),
                    this.getApplicationContext()
            );
```

Se invoca al método sendRequest() de la clase ARequest y se realiza la petición al servidor tal y como se ha comentado anteriormente.

En caso de que no se produzca ningún error en este proceso, se lanza la actividad PartyResultActivity, que tratará el JSON devuelto y lo mostrará la información en la pantalla. El objeto JSON con la respuesta se envía a la siguiente actividad como parámetro con el método putExtra() de la clase Intent.

- Actividad de resultado de la petición.

En esta actividad se trata el JSON recibido del servidor y se muestra al usuario. La información contenida en el JSON es una serie de etiquetas, el apartado 1 del Anexo C se encuentra la información detallada en el archivo RAML.

Como la información obtenida no incluye imágenes ni gráficos, el diseño de la pantalla se ha realizado organizando una serie de TextViews y LinearLayouts para mostrar la información como se ilustra en la siguiente imagen.

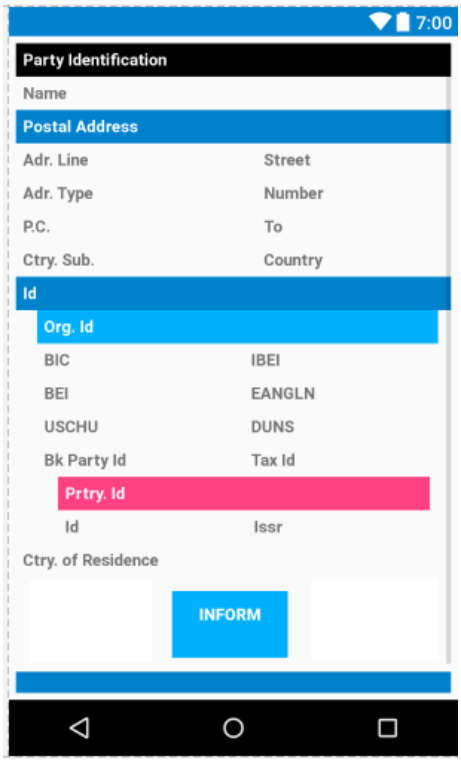


Figura 26 – Diseño de la pantalla de respuesta del servicio de parties.

En el diseño de la pantalla se pueden ver los nombres de los campos, al lado de cado uno se encuentra un campo de texto vacío que será completado con el parámetro recibido correspondiente.

Debajo de la información se incluye un botón mediante el que se utilizará el servicio de envío de notificaciones de la consulta realizada.

Como se ha comentado, la información se recibe en formato JSON, como en los tres servicios la forma de tratar este objeto es similar, se explica en primer lugar el proceso genérico y a continuación los detalles específicos de cada caso.

Las herramientas empleadas para este proceso han sido obtenidas de la librería Gson de Google, por lo que el primer paso consiste en añadir la siguiente dependencia en el archivo *build.gradle* de la aplicación.

```
compile 'com.google.code.gson:gson:2.7'
```

Una vez realizado este paso, se pueden importar las clases de esta librería que se van a utilizar, en concreto las clases *com.google.gson.Gson* y *com.google.gson.GsonBuilder*. Mediante estas clases podemos convertir un objeto JSON en una clase java manejable por la aplicación con tan solo las dos siguientes líneas de código.

```
Gson gson = new GsonBuilder().create();
MyPojo myPojo = gson.fromJson(respuesta, MyPojo.class);
```

En la segunda línea, el parámetro 'respuesta' es el objeto JSON recibido en la actividad anterior y almacenado como un *String*. La clase *MyPojo* consiste en una clase POJO (*Plain Old Java Object*) que actuará como *bean* del JSON. En cuanto a estas clases *bean* comentar que se han utilizado las anotaciones provistas por la librería Gson empleada, ya que permiten adaptar los nombres de los atributos del *bean* a los nombres de los parámetros del JSON en una sola línea, sin tener que cambiar los nombres en todos los métodos.

Realmente las clases *bean* no consisten en una sola clase, sino en una estructura de clases en forma de árbol para poder encapsular correctamente todos los campos del objeto JSON. La estructura de clases para el servicio de contrapartidas se puede ver en la Figura 14. Cada una de estas clases simplemente implementa los atributos que se aprecian en el diagrama, un constructor para estos atributos y sus métodos *getters* y *setters* correspondientes, además de las notaciones comentadas con los nombres de los parámetros. Las relaciones entre las clases se dan estableciendo las clases inferiores como atributos de su clase superior en la estructura.

Volviendo a la lógica de la actividad, una vez obtenido el POJO, utilizando los métodos *get* del mismo, se van obteniendo los parámetros (*Strings* en todos los casos) y rellenando los campos de texto correspondientes. En caso de que un parámetro se devuelva vacío (situación muy habitual ya que normalmente cada contrapartida posee unos datos u otros en función de sus características) el campo de texto quedará vacío.

Una vez establecidos todos los campos, finaliza el método onCreate() y se presenta la pantalla resultante al usuario. El hilo de la aplicación queda entonces a la escucha de que

se pulsen los botones de la pantalla. Al final de la pantalla se encuentra el botón de retorno, que termina esta actividad y vuelve a la pantalla de consulta

El otro botón es el de notificación de consulta, al pulsarse este botón, el método que se ejecuta crea una instancia de la clase SendNotificationRequest, a la cual se pasa información del usuario, correo de destino y la información de la contrapartida consultada. Esta clase, como se comentó, concatena la URL y a continuación se lleva a cabo la conexión. Este servicio no recibe información en la respuesta, así que en caso de que no se produzca un error mediante una excepción, se lanza un mensaje por pantalla indicando que la notificación se ha llevado a cabo correctamente.

6.2.3.5 Implantación del servicio de consulta de jerarquía interna

Para la pantalla de introducción de parámetros del servicio de jerarquía interna, el tipo de jerarquía y el rol de la contrapartida pedida, como son datos fijos de una lista de valores posibles, en lugar de dejar que el usuario introduzca los datos mediante texto, se ha habilitado una pestaña para cada valor, que al pulsarla despliega una lista con los valores posibles que se pueden escoger. El diseño de la pantalla se muestra a continuación.

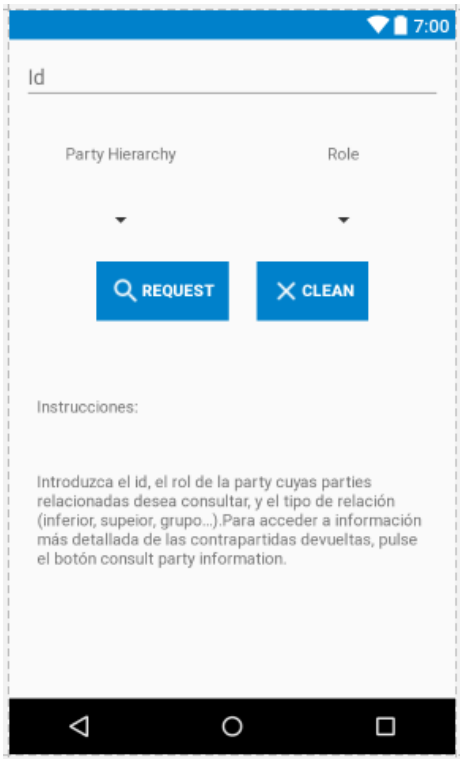


Figura 27 – Diseño de la pantalla de consulta del servicio de internal structure.

Para implementar los menús desplegables se ha utilizado el *widget Spinner* de Android. Los valores que se quieren mostrar en la lista se almacenan en el programa como un *array* de *Strings*, y mediante las clases *Spinner* y *ArrayAdapter* se rellena el menú con dicho *array*.

La lógica de esta actividad no presenta ninguna novedad respecto al caso anterior (salvo que hay que realizar un pequeño mapeo entre los valores de rol y tipo de jerarquía, que se presentan mediante su nombre y es necesario convertirlos a su código correspondiente).

- Pantalla de resultado del servicio.

La pantalla de respuesta del servicio de jerarquía presenta la dificultad de que las contrapartidas relacionadas pueden ser una, varias o ninguna, es decir, se recibe un *array* de objetos de este tipo. Para mostrar el *array* de datos recibido se ha utilizado el *widget ListView*. La parte superior de la pantalla muestra información de la contrapartida

introducida por lo que simplemente se rellenan las etiquetas de la misma manera que en la actividad anterior. Debajo de la etiqueta de *Related Parties* se mostrarán los datos de las contrapartidas relacionadas.

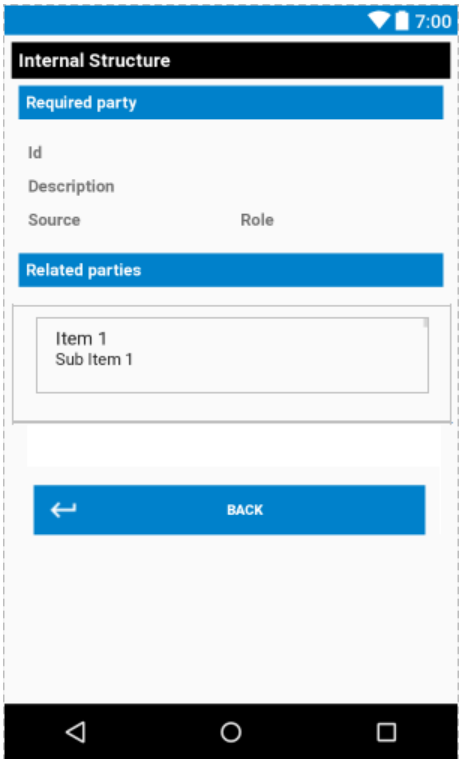


Figura 28 – Diseño de la pantalla de respuesta del servicio de internal structure.

A continuación se muestra el diseño de cada elemento de contrapartida relacionada con los cuatro campos recibidos. Debajo de los datos mostrados se incluye un botón que permite realizar una búsqueda de información más detallada de la contrapartida recibida haciendo uso del servicio de consulta de contrapartidas concatenando los valores de Id y rol.

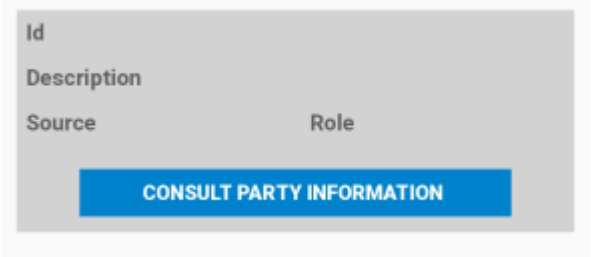


Figura 29 – Diseño del fragmento de información de contrapartida relacionada del servicio de internal structure.

En este caso, la diferencia respecto al servicio anterior al leer el objeto JSON consiste en que este contiene un *array* en uno de los atributos, en el de contrapartidas relacionadas. Para encapsular el objeto en una clase java se ha utilizado la clase *List* de java para manejar el *array*, el resto de la funcionalidad es idéntica.

Una vez leído el JSON de respuesta se rellenan los campos de texto de la contrapartida pedida utilizando los *getters* y *setters* del java *bean*. Mediante el *getter* del atributo de partidas relacionadas se obtiene una java *List*. Para adaptar los datos de este objeto a la pantalla, se ha implantado una clase llamada *RelatedPartyAdapter* que hereda de la clase *BaseAdapter*. Esta clase implementa el método getView() de *BaseAdapter*, llenando los campos del recurso de la contrapartida relacionada.

Por último, una vez desplegada la pantalla, el programa queda escuchando que se pulse el botón de consultar detalle de la contrapartida relacionada. En este caso, se toman los datos de id y rol del elemento, se crea una instancia de *PartyRequest* con estos dos datos, se lanza la petición y se lanza la actividad de resultado del servicio de contrapartidas.

6.2.3.6 Implementación del servicio de consulta de calendarios

Como se ha comentado, entre los beneficios que buscan sacar las APIs a los sistemas, uno de ellos consiste en transformar la información que estos proporcionan para presentarla de manera legible y visual para los usuarios, ofreciendo usabilidad. En el servicio de calendarios se ha tenido esta característica muy en cuenta, ya que lo que se recibía anteriormente era un fichero XML con todas las fechas no laborables puestas en una lista,

un formato para presentar la información poco intuitivo para el usuario. El diseño de este servicio ha buscado presentar esta información de manera visual, para lo cual se ha diseñado una pantalla con un calendario donde se mostrarán las fechas obtenidas.

En esta actividad dos de los datos son fechas y para introducirlos se han desplegado una serie de listas desplegables, de la misma manera que en el servicio anterior, con los valores posibles correspondientes al día, mes y año que se quiere introducir. El diseño es el siguiente.

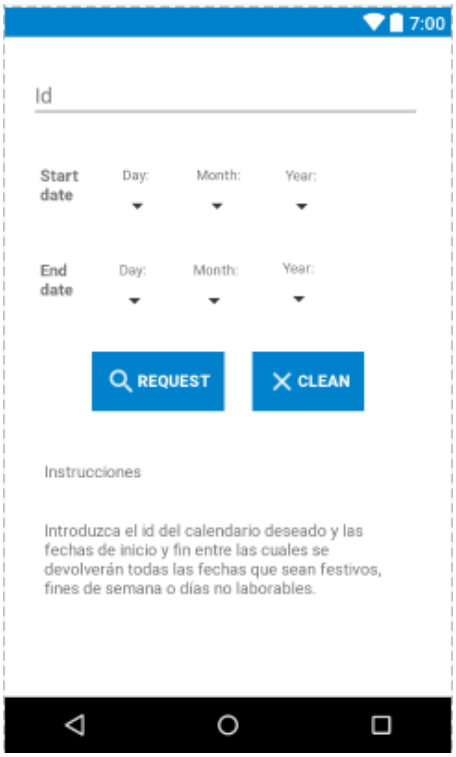


Figura 30 – Diseño de la pantalla de consulta del servicio de calendars.

En cuanto a la lógica de esta pantalla no hay ninguna novedad respecto a los dos servicios anteriores. Lo único reseñable consiste en que los valores de las fechas se deben pasar al servicio en formato YYYYMMDD, por lo que hay que hay que hacer una pequeña conversión al formato antes de concatenar la URL.

Una vez realizada la conexión, pasamos como parámetros a la actividad de resultado tanto el JSON devuelto como las fechas de inicio y fin de rango, en el formato comentado, para crear la vista.

- Pantalla de resultado del servicio.

El diseño de la pantalla de resultado del servicio de calendarios muestra en primer lugar el identificador y nombre del calendario pedido en un par de campos de texto.

El principal elemento de esta pantalla es la vista del calendario, en la cual se mostrarán los tres tipos de días no laborables (fines de semana, festivos y otros días no laborables) en tres colores diferentes (rojo, amarillo y marrón).

En primer lugar se trató de emplear el *widget* de calendarios ofrecido por Android, ya que ofrece un diseño sencillo y elegante, y tiene la ventaja de resultar muy sencillo su manejo, sin embargo, debido a la muy escasa funcionalidad que permite se acabó optando por otra solución.

Esta solución consiste en un fragmento ofrecido en la página oficial de Github y diseñado por Roomorama llamado Caldroid. [26] Caldroid es un fragmento para Android que despliega un calendario en la pantalla, aunque su implementación no es tan sencilla como la del *widget* ofrecido por Android, el objeto tiene mucha más funcionalidad disponible, como por ejemplo, poder cambiar el estilo de días individuales del calendario, lo cual resultaba imprescindible para el diseño.

El diseño además permite que se muestren como activos únicamente los días comprendidos entre el rango de fechas introducido por el usuario, que el usuario pueda navegar por los meses del año y que al seleccionar una fecha activa se muestre un mensaje con el valor de la fecha seleccionada.

Debajo del fragmento del calendario se mostrará un pequeño índice con los colores como se ha comentado, y por último, el botón de retorno a la actividad anterior presente en el resto de servicios. El diseño se muestra a continuación.

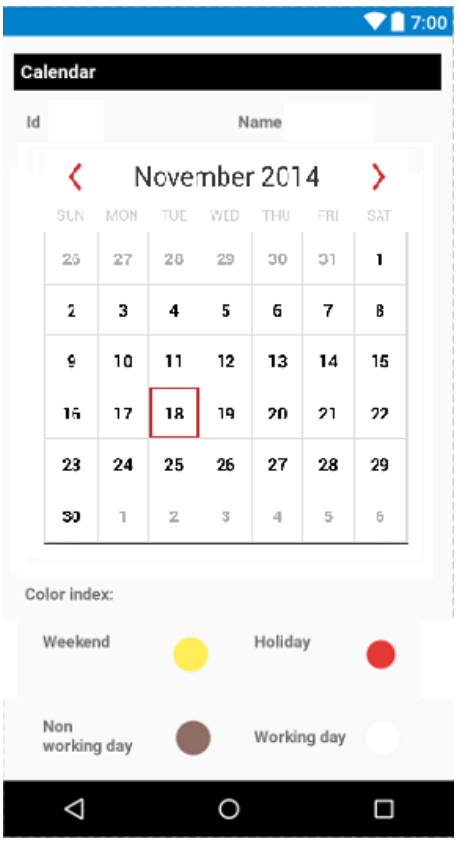


Figura 31 – Diseño de la pantalla de respuesta del servicio de calendars.

La manera de encapsular la información contenida en el JSON no presenta ninguna novedad respecto a los servicios anteriores por lo que en cuanto a la lógica de la actividad se comenta únicamente la manera de implantar el objeto Caldroid.

En primer lugar se agrega la dependencia en el fichero *gradle* correspondiente de la aplicación introduciendo la siguiente línea.

```
compile 'com.roomorama:caldroid:3.0.1'
```

Realizado esto se puede importar la librería de la cual usaremos los objetos *CaldroidFragment* y *CaldroidListener*.

El primer paso consiste en crear una instancia de la clase *CaldroidFragment* y configurar el diseño general de la vista, en concreto configuramos que el mes que aparezca al lanzar la pantalla coincida con la fecha de inicio de búsqueda introducida en la pantalla anterior por el usuario, habilitamos la opción de navegar entre meses, indicamos que se vean 6 semanas en una vista y configuramos el lunes como primer día de la semana. Para ello se utiliza la clase *Bundle* que almacena las configuraciones mencionadas. A continuación se incluyen estas líneas de código.

```
CaldroidFragment caldroidFragment = new CaldroidFragment();

Bundle args = new Bundle();
args.putInt(CaldroidFragment.MONTH, getMonthfromDate(startDateInt));
args.putInt(CaldroidFragment.YEAR, getYearfromDate(startDateInt));
args.putBoolean(CaldroidFragment.ENABLE_SWIPE, true);
args.putBoolean(CaldroidFragment.SIX_WEEKS_IN_CALENDAR, true);
args.putInt(CaldroidFragment.START_DAY_OF_WEEK, CaldroidFragment.MONDAY);

caldroidFragment.setArguments(args);
```

Una vez configurada la vista general del calendario, encapsulamos el JSON en las dos clases POJO descritas en la Figura 14, rellenamos los campos de texto de Id y nombre del calendario y obtenemos un *List* de la clase *CalendarDateData* con todas las fechas devueltas y su tipo de día no laborable. Este valor puede ser 1, 2 ó 3 en función de si es fin de semana, festivo u otro día no laborable, así que iteramos la lista 3 veces para encontrar todas las fechas de cada tipo, y se rellena esa fecha del calendario con su color correspondiente mediante el método setBackgroundDrawableForDate() de la clase *FragmentCaldroid*. Las fechas se introducen con instancias de la clase *Date*.

Una vez establecidos todos los colores, insertamos el calendario configurado en la pantalla mediante el siguiente código.

```
FragmentTransaction t = getSupportFragmentManager().beginTransaction();
t.replace(R.id.calendar1, caldroidFragment);
t.commit();
```

6.2.4 REALIZACIÓN DE TESTS MEDIANTE SERVICIOS *MOCK* IMPLEMENTADOS CON SOAPUI

Aunque la app definitiva está diseñada para lanzar las peticiones contra la API ubicada en los servidores de desarrollo de BBVA CIB, para poder realizar *tests* de conexión de la app de una manera más cómoda, ya que el realizar peticiones a los servidores está sujeto a restricciones debido a temas de seguridad, se ha utilizado el programa SoapUI para desplegar 4 servicios *mock* correspondientes a los de *parties*, *internal structure*, *calendars* y el del gestor de notificaciones para enviar automáticamente los correos de aviso de consulta realizada.

El programa permite desplegar servicios REST y SOAP, así que se ha seleccionado la opción de REST en todos los casos, al ser una API *restful*. Aunque el programa permite introducir *scripts* para dotar de cierto dinamismo a los servicios, en este caso solamente se pretendía comprobar que la app lanzaba la petición de manera correcta y procesaba bien el objeto JSON recibido en la respuesta, por ello se han definido los servicios como estáticos.

Todos los servicios se han configurado con ip *localhost*, por lo que se utilizará la ip del propio pc en el que se cargan los servicios para formar la url desde el móvil que ejecuta la app. Cada uno utilizará un puerto TCP diferente de manera que se puedan utilizar los 4 a la vez para poder probar la funcionalidad completa de la app, también se deberán entonces configurar los puertos en la concatenación de las url. Una vez hecho esto, se configura el código de estado HTTP como 200, es decir, petición correcta, ya que no se pretenden comprobar los errores, se selecciona el tipo de respuesta como *application/json*, y se adjunta un objeto JSON de ejemplo que se devolverá en cada petición hecha a ese recurso.

A continuación se muestra una imagen de la configuración del servicio *mock* de consulta de contrapartidas con el JSON utilizado en la prueba.

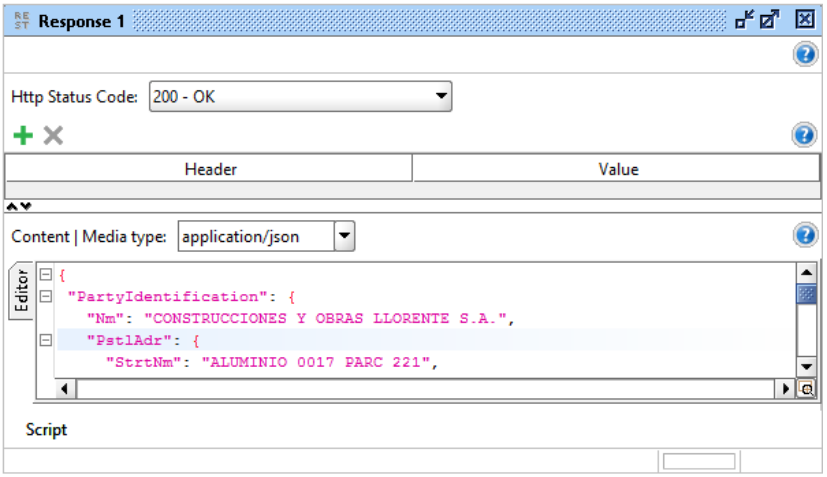


Figura 32 – Mock service del servicio de parties con SoapUI

Se incluye además una imagen del servicio corriendo en el puerto 8089, con algunas peticiones realizadas, como se puede observar en la ventana inferior de la pantalla.

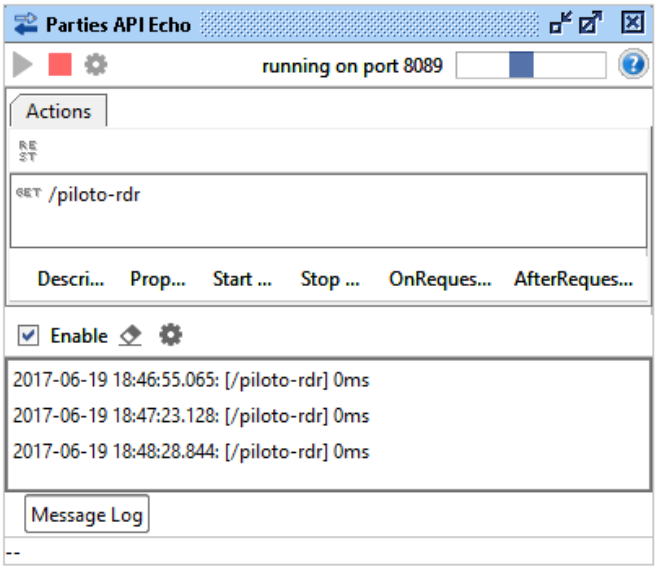


Figura 33 – Mock service del servicio de parties escuchando peticiones de la app.

7. ANÁLISIS DE RESULTADOS

En primer lugar se ha logrado generar los documentos reales de modelado de tres APIs, consulta de información de contrapartidas, consulta de jerarquía interna y consulta de calendarios, exponiendo unos sistemas como servicios RESTful presentando los datos en formato JSON, mientras que anteriormente la integración de sistemas con estos servicios era compleja debido a la necesidad de emplear un formato muy específico de Fix-XML, facilitando enormemente la integración con cualquier formato de tecnología.

Además, se ha logrado adaptar los datos del servicio de contrapartidas a un esquema en formato ISO 20022, logrando así la adaptación a los estándares que la Unión Europea pretende que sean adoptados mediante la nueva ley de pagos. En el caso de los servicios de jerarquía y calendarios ha terminado resultando imposible la adaptación a dicho estándar debido a la inexistencia de un esquema capaz de adaptar los datos de estos dos sistemas, queda, por tanto, como objetivo pendiente comentado en el apartado de trabajos futuros del capítulo siguiente el encontrar la manera de apartar estos datos al estándar.

Comentar por último que estos documentos han sido catalogados en una primera maqueta de portal API implantado en el departamento CIB de BBVA.

Como se ha explicado, el proceso de apificación de servicios es un proceso complejo y que sobrepasa ampliamente las expectativas de un proyecto de esta envergadura debido principalmente a la complejidad técnica de la tecnología interna (bus de integración, esquemas de datos, etc…). Por ello, una vez desarrollados y validados los archivos de documentación RAML (ver anexo C), los encargados de desarrollar el adaptador de canal que traduce el formato JSON-REST a la tecnología propia del bus (XML, FiX, JMS…) fue el propio equipo de integración del departamento CIB.

Una vez desarrolladas estas piezas e implementadas en los servidores de desarrollo internos, y desarrollada la aplicación explicada en el capítulo anterior, fue posible realizar una prueba completa de la funcionalidad del sistema.

Para ello se utilizó un móvil Samsung Galaxy S6 con versión de sistema operativo Android 6.0.1 para ejecutar la aplicación. Debido a las políticas de seguridad del banco, los servidores donde se encuentran instaladas las APIs sólo aceptan peticiones provenientes de la red corporativa, por lo que la prueba se ha debido llevar a cabo a través de la misma.

A continuación se muestran las pantallas con los resultados de la ejecución de la aplicación en el entorno real.

Para realizar la autenticación con el servicio de Google se ha utilizado mi cuenta de correo de BBVA, que utiliza el servicio de Google, por lo que la cuenta es válida. A continuación se muestra una captura de la aplicación una vez realizado el *login*.

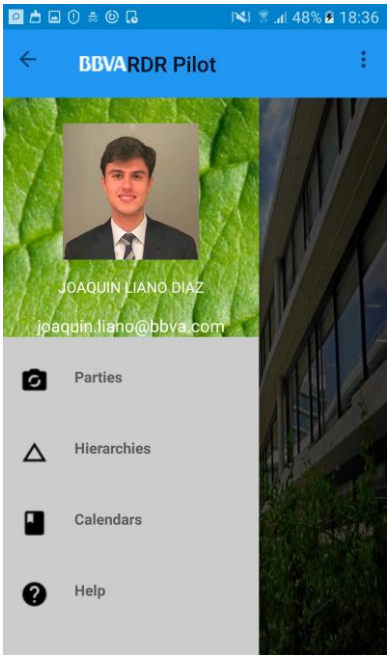


Figura 34 – Resultado del proceso de login de la aplicación.

Una vez realizado este proceso correctamente se procede a comprobar el funcionamiento de los tres servicios.

Escogemos en primer lugar el servicio de consulta de contrapartidas e introducimos los datos de la contrapartida conocida Id = '101' e *Issuer* = '45'. Realizamos la consulta y obtenemos el resultado mostrado a continuación.

Figura 35 – Resultado de una consulta al servicio de contrapartidas.

Los datos corresponden a una empresa de Valladolid de la cual obtenemos su nombre y dirección. Comprobamos asimismo el funcionamiento del servicio de notificaciones, para lo cual, una vez pulsado el botón de la parte inferior de la pantalla, se comprueba que le llega un correo electrónico al usuario configurado con la información de la consulta realizada.

A continuación comprobamos el servicio de consulta de jerarquía interna. Introducimos los datos de Id = 'PORT00000067' y rol = '*Calculation agent*', y seleccionamos la opción de jerarquía inferior. Obtenemos el resultado mostrado en la imagen.

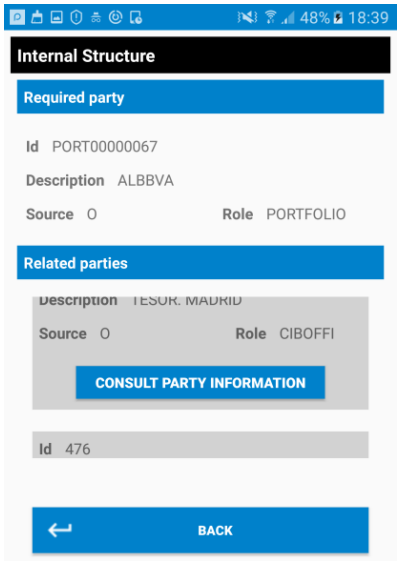


Figura 36 – Resultado de una consulta al servicio de internal structure.

Se han obtenido los datos de dos contrapartidas por debajo de la pedida según la estructura de datos de Ids = '475' y '476' pertenecientes a dos oficinas. Se ha comprobado que la navegabilidad entre los servicios de jerarquía y contrapartidas se llevan a cabo correctamente, para lo cual se pulsa el botón de consultar información sobre el elemento y se accede a una pantalla similar a la anterior mostrada del servicio de contrapartidas, con datos sobre la dirección de las oficinas.

Por último, para comprobar el funcionamiento del servicio de consulta de calendarios se ha utilizado el calendario de Madrid, introduciendo como rango de fechas del 5 al 20 de abril de 2017, para el cual se han devuelto las fechas correspondientes a los dos fines de semana comprendidos entre medias, y el Viernes Santo y Lunes de Pascua, días en los cuales era fiesta en la bolsa europea.

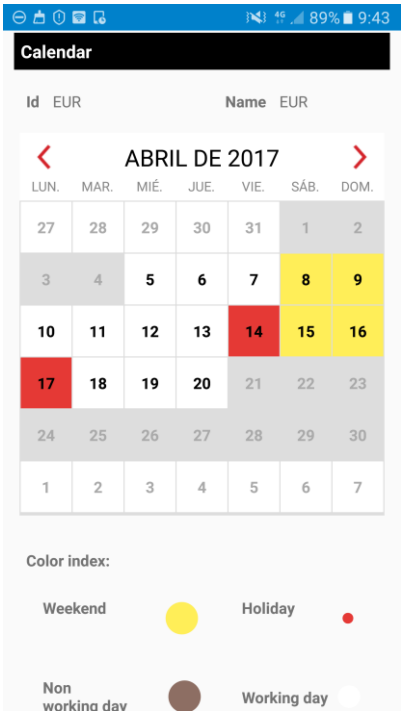


Figura 37 – Resultado de una consulta al servicio de calendars.

Con los resultados obtenidos, se ha conseguido realizar una prueba completa de apificación, cubriendo los elementos de la cadena de valor de las APIs correspondientes al diseño de la propia API y de una aplicación cliente con la que se ha pretendido obtener cierta funcionalidad añadida, como la rápida navegabilidad entre los servicios de jerarquía y contrapartidas, y el diseño visual del servicio de calendarios.

8. CONCLUSIONES Y TRABAJOS FUTUROS

En este apartado se comentarán las conclusiones extraídas a nivel teórico del estudio realizado, así como las extraídas de la prueba de apificación y consumo de los servicios realizado, y cuáles son las principales líneas de trabajo futuro a raíz del trabajo realizado.

8.1 CONCLUSIONES

En el aspecto teórico se ha pretendido en primer lugar analizar los principales requerimientos, habiendo estado este proyecto centrado en el mundo de la banca, por los que el desarrollo de APIs está cobrando una importancia tan elevada en los últimos años, y a continuación dar una visión global acerca de cómo este tipo de sistemas dan una solución efectiva a estas necesidades.

A nivel teórico, en cuanto a la implantación de sistemas de apificación, se ha pretendido recalcar la importancia de la adopción de un equipo de gobierno API que realice las funciones explicadas, como la adaptación hacia estándares promovidos por las instituciones (caso del estándar ISO 20022 por parte de la EBA), la gestión efectiva del ciclo de vida de las APIs o el análisis de los posibles consumidores y desarrolladores de apps y de qué manera hacerles llegar mejor la información sobre las APIs. Se recalca asimismo la importancia de la adopción de una solución de API *manager* potente, pues tal y como se ha comentado, gran parte del valor que aportan las APIs radica en la correcta gestión de las mismas, proceso para el cual resulta imprescindible la adopción de dicho sistema por parte de las entidades. Este sistema, como se ha comentado, es vital no solo para soportar eficiente y escalablemente las llamadas a los recursos apificados, sino también para que los beneficios mencionados de la apificación como pueden ser la agilización y flexibilización de los procesos, así como la reducción del *time to market*, se vean llevados a cabo, así como la posibilidad de monetizar el uso de los recursos, y para definir las políticas de seguridad correctas. El API *manager* también ejerce un rol

fundamental en el proceso de garantizar que no falle ningún elemento de la cadena de valor, lo cual es imprescindible para obtener el beneficio esperado, para lo cual se debe diseñar un API *Portal* que resulte eficiente tanto para que los desarrolladores de APIs puedan desarrollar las mismas eficientemente, como para que los desarrolladores de apps clientes reciban la documentación necesaria para su uso.

Se ha hecho especial hincapié en el tema de la securización de los servicios, un tema de una importancia cada vez más elevada, especialmente si atendemos a los últimos ciberataques realizados a escala global y que han conseguido poner en jaque durante cierto tiempo a decenas de empresas e instituciones de diferentes países, y que en el sector financiero cobra una relevancia elevada debido al manejo de datos bancarios de usuarios, por poner un ejemplo. [27] Este análisis ha puesto el foco principalmente en el proceso de autenticación de los clientes, analizando las principales soluciones actuales, como el estándar OAuth2 y los sistemas de utilización de *tokens*.

Por último, se ha tratado de dar una visión de las principales líneas de diseño de APIs en el aspecto puramente tecnológico. Principalmente la adopción del modelo REST y algunas buenas prácticas al respecto, como el versionado, la paginación, etc…, así como la adopción del formato JSON de los datos, frente a otros como XML, debido a la flexibilidad que presenta y la correcta documentación de APIs mediante lenguajes de modelado como RAML y *Swagger*.

En el aspecto práctico se ha llevado a cabo el proceso de apificación completo de tres servicios de *backend* de BBVA. Concretamente, se han cubierto los roles de desarrollador de la propia API y de aplicaciones clientes, comprobando así la viabilidad del sistema al haber integrado los recursos *core* del banco con una aplicación Android en un periodo de tiempo tan reducido.

Se puede concluir asimismo que la apificación de los servicios es una empresa muy costosa de realizar, por la cantidad de recursos que resulta necesario invertir, pero muy beneficiosa a largo plazo.

Lo costoso es la adaptación a la metodología, la implantación de un API portal eficiente a nivel público, de departamentos, entre empresas, etc… La implantación de los procesos de gestión, el API *manager*. Sin embargo, una vez realizada esta inversión correctamente se puede observar que los procesos se agilizan enormemente ya que, como se ha demostrado en este proyecto, se ha logrado cubrir casi toda la cadena de valor de la API para ofrecer un producto más o menos funcional (a falta de seguridad, adaptación al estándar, monitorización, etc…) en un tiempo muy reducido (4 meses de proyecto) y con un presupuesto mínimo (12 900 € aproximadamente el trabajo realizado dentro del proyecto).

Por otra parte, con PSD2, cualquier usuario podrá iniciar pagos y obtener información sobre cuentas habiendo obtenido únicamente la autorización del usuario propietario de la cuenta. Viendo la rapidez con que están apareciendo nuevos dispositivos, nuevas tecnologías, etc, y la preferencia cada vez mayor de los usuarios por las aplicaciones móviles, ligeras, rápidas y sencillas de utilizar, se concluye que la apificación de los servicios necesarios para llevar a cabo esta funcionalidad, resulta un proceso adecuado para ofrecer la mejor funcionalidad posible, con las garantías de seguridad requeridas en procesos de estas características.

8.2 TRABAJOS FUTUROS

Este proyecto ha tenido carácter de prueba de concepto para estudiar la viabilidad de soluciones de estas características, por ello, el apartado de trabajos futuros tiene una gran importancia aquí.

En primer lugar, se debe comentar que actualmente, en el departamento CIB de BBVA no hay un auténtico API *manager* implantado, sino que las diferentes funcionalidades del mismo (*Gateway, portal* y *manager*) se llevan a cabo por diferentes equipos, lo cual dificulta enormemente la gestión del ciclo de vida. Una vez adoptado dicho elemento, se podrá realizar una gestión efectiva de las API desarrolladas. Comentar por tanto la necesidad por parte del departamento de implementar esta solución.

8.2.1 MEJORAS RESPECTO A LAS API DESARROLLADAS

Tal y como se ha comentado en los apartados anteriores, mientras que el objetivo inicial consistía en adoptar el estándar ISO 20022 para el modelado de los datos de las tres APIs, finalmente sólo ha resultado posible la adopción en la API de contrapartidas. Una de las principales líneas de trabajo, por tanto, debería ser la de encontrar un esquema mediante el cual adaptar los dos servicios restantes a este estándar.

Para el servicio de contrapartidas resultaría de gran utilidad poder acceder a la información de la contrapartida introduciendo campos más sencillos como el nombre de la persona o de la entidad en cuestión, en lugar de tener que introducir el código identificador interno de la partida.

Por otra parte, sólo se han desarrollado servicios de consulta de información (método HTTP GET), un próximo paso podría ser desarrollar servicio para introducir nuevos elementos (POST), *update* (POST), etc…

En el servicio de consulta de calendarios se ha descartado en esta primera versión del servicio API el campo de divisa asociada al calendario, que podría tener utilidad práctica, por tanto, como trabajo futuro cabría incluir este campo en el objeto JSON devuelto en la respuesta a la consulta.

8.2.2 SEGURIDAD

En la app desarrollada cualquier usuario que posea una cuenta de Google válida puede acceder a la aplicación, ya que la API está orientada al consumo privado o de *partners*, aunque realmente lo principal es configurar las políticas de seguridad en la propia API y no en la app que la utiliza, como solución provisional se podrían configurar *scopes* en el proceso de autenticación con el servicio de Google, que permitan, por ejemplo, utilizar únicamente los servicios si el usuario se ha validado con una cuenta de Google de BBVA.

Este sistema de validación con Google sin embargo, se considera un sistema provisional, ya que está orientado a pequeñas aplicaciones que manejan datos de poco valor. Los

sistemas de BBVA sin embargo, por la importancia de los datos que maneja, debe tener un sistema mucho más robusto que el que puede garantizar la protección mencionada. El enfoque que debe tomar el sistema, por tanto, debe ser el de la implantación de estándares como OAuth, sistemas de protección como Armadillo o la adopción de HTTPS en las conexiones REST.

Además sólo se puede conectar a las APIs a través de la red corporativa debido a las políticas de seguridad internas, sin haber sido posible realizar la conexión ni siquiera a través de una VPN. Al tener estas APIs una orientación hacia el consumo de usuarios externos (API para *partners*), estas APIs deberán ser desplegadas en servidores a los que se pueda acceder desde el exterior.

8.2.3 DISEÑO DE LA APP

Debido a la rapidez con que se ha tenido que desarrollar la aplicación, además de funcionalidad de securización, autenticación... no se ha desarrollado una correcta gestión de los flujos, y en caso de que se produzca un fallo en la conexión, la gestión implantada es mínima, cubre pocos casos, y por tanto, en caso de que se dé un error sin tratamiento correcto, la app entera se detendría, derivando en una experiencia de usuario muy pobre. Es necesario, para evitar esto, tratar los flujos de llamada a la API en un hilo paralelo al hilo principal con el objetivo de que en caso de error, se pueda retomar el hilo principal sin que se detenga la app.

9. BIBLIOGRAFÍA

[1] Daniel Jacobson, Greg Brail & Dan Woods "APIs, a strategy guide", O`Reilly, 2012

[2] "The Power of the API Economy. Stimulate Innovation, Increase Productivity, Develop New Channels, and Reach New Markets" – IBM

[3] "Framework de Integración. Conceptos" – CIB BBVA Site

[4] "Android developers" https://developer.android.com/studio/index.html

[5] SoapUI - https://www.soapui.org/

[6] IBM - Implement ISO 20022 payment initiation messages in a payment processing solution https://www.ibm.com/developerworks/data/library/techarticle/dm-1307isopayment/

[7] "Órdenes en formato ISO 20022 para emisión de transferencias y cheques en euros", CECA, Noviembre 2015

[8] "PSD2, a Business Model Perspective: Financial APIs will Foster business model innovations". *BBVA Research.*

[9] "PSD2 y Open API, una oportunidad para la banca". www.atmira.com

[10] BBVA API market. https://www.bbvaapimarket.com/home

[11] Fintonic Web Site, https://www.fintonic.com/

[12] Trustly Web Site. https://trustly.com/es/

[13] *How to Build an API Program That Doesn't Suck - ApiGee*

[14] Simple Web Site. https://www.simple.com/

[15] Corporate & Investment Banking CTO – BBVA Site

[16] API Management: ¿Qué es y para qué sirve? – Paradigma. https://www.paradigmadigital.com/dev/api-management-que-es-y-para-que-sirve/

[17] Difference between XSS and CSRF https://security.stackexchange.com/questions/138987/difference-between-xss-and-csrf

[18] BBVA CIB Guía de diseño – BBVA Site

[19] REST, el principio HATEOAS y Spring HATEOAS. https://www.adictosaltrabajo.com/tutoriales/spring-hateoas/

[20] BBVA CIB Gobierno: Comparativa de lenguajes de diseño de APIs – BBVA Site

[21] What is FIX? – FIX Trading Community. http://www.fixtradingcommunity.org/pg/main/what-is-fix

[22] FPL:FIXML Syntax. http://fixwiki.org/fixwiki/FPL:FIXML_Syntax

[23] The road to standards Nirvana, Martin Sexton

[24] tsin.002.001.01. schema https://www.iso20022.org/standardsrepository/public/wqt/Description/mx/tsin.002.001.01

[25] Google Sign-In. https://developers.google.com/identity/

[26] Roomorama, Caldroid. A better calendar for Android. https://github.com/roomorama/Caldroid .

[27] Un potente ciberataque afecta a grandes empresas de todo el mundo. http://internacional.elpais.com/internacional/2017/06/27/actualidad/1498568187_011218.html

ANEXO A: MANUAL DE USUARIO

Este manual de usuario tiene como objetivo explicar los pasos que debe seguir un usuario para la utilización de la aplicación.

Una vez iniciada la aplicación, el primer paso consiste en autenticarse. Para poder acceder a la funcionalidad de la aplicación el usuario debe tener una cuenta de Google y autenticarse con ella.

Para ello, en la pantalla inicial se debe pulsar el botón de la parte superior de la pantalla con la etiqueta "Iniciar sesión con Google". Una vez pulsado se pedirá al usuario que introduzca su cuenta de correo y contraseña.

Una vez realizada correctamente la autenticación contra el servicio de *log-in* de *google*, se accederá a la pantalla principal de la aplicación. Pulsando sobre el icono de menú de la parte superior izquierda se accede al menú mostrado en la siguiente imagen.

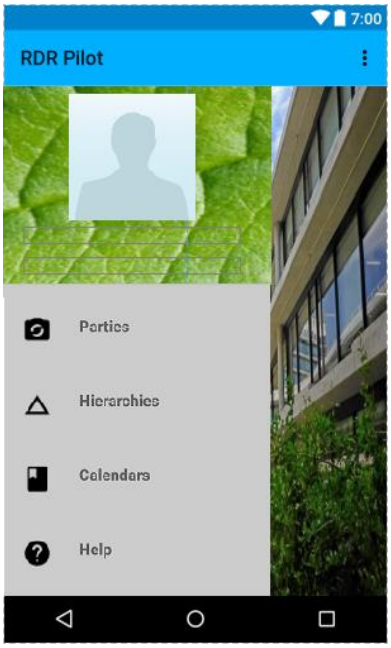


Figura 23 – Diseño de la pantalla de menú de servicios.

En la parte superior del menú se muestran la imagen, en caso de tenerla, nombre y correo de la cuenta empleada en el proceso de *log in*. En la parte inferior se listan los servicios disponibles y un icono de ayuda.

Es necesario mencionar, antes de explicar el funcionamiento de los servicios, que para su funcionamiento, es necesario conectar el dispositivo que utiliza la aplicación a la red WiFi corporativa de BBVA, ya que los servicios solamente son accesibles de la misma, aunque el proceso de autenticación si se puede realizar desde cualquier red con conexión a internet ya que se utiliza un servicio de *google*.

Una vez autenticados y conectados a la red corporativa, podemos acceder a cualquiera de los tres servicios disponibles pulsando sobre su nombre.

A.1. SERVICIO DE CONSULTA DE INFORMACIÓN DE CONTRAPARTIDAS

Pulsando sobre *Parties* accedemos a la pantalla mostrada en la siguiente imagen.

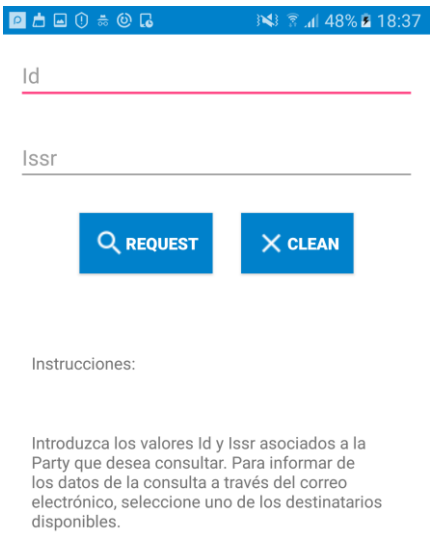


Figura 25 – Diseño de la pantalla de consulta del servicio de parties.

Este servicio muestra la información asociada a una contrapartida relacionada con el grupo BBVA. Para consultar la información disponible sobre una de ellas se deberán conocer dos datos, el código identificador de la misma según los datos almacenados en la entidad, y el código identificador de la *branch* de la misma.

Conocidos estos dos datos se introducirán en los dos campos mostrados en la pantalla anterior, el primero en el campo Id y el segundo en el campo Issr. Pulsando el botón clean se borra todo texto introducido en los dos campos de texto. Una vez introducidos los datos, pulsando sobre el botón *request*, en caso de que la petición se realice sin fallos, se accederá a la siguiente pantalla.

Figura 26 – Diseño de la pantalla de respuesta del servicio de consulta de contrapartidas.

Ya que dependiendo del tipo de contrapartida, se obtendrán unos datos u otros, en la pantalla se mostrarán todos los datos disponibles, y aquellos de los que no se disponga, sus campos quedarán vacíos.

Pulsando el botón de informar de la parte inferior, se envía un correo a un usuario fijo del banco informando acerca de la consulta efectuada y el usuario que la ha efectuado.

Haciendo *scroll* en la pantalla hacia abajo, se muestra un botón de retorno, pulsándolo se accede de nuevo a la pantalla de consulta de manera que se puede efectuar una nueva búsqueda.

En caso de introducir un identificador de una contrapartida inexistente en la base de datos la pantalla se mostrará la pantalla anterior con todos los campos vacíos.

Por último, en caso de que se produjese algún fallo en la conexión en el momento de efectuar la consulta, se mostrará la siguiente ventana.

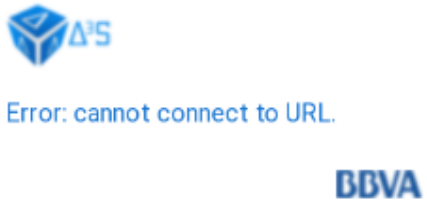


Figura 38 – Mensaje de error de conexión .

A.2. SERVICIO DE CONSULTA DE JERARQUÍA INTERNA

Si accedemos desde el menú principal al servicio de *internal structure* se mostrará la siguiente pantalla de consulta.

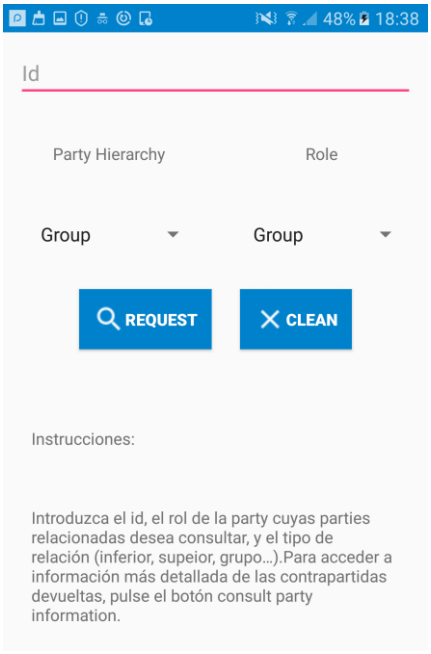


Figura 27 – Diseño de la pantalla de consulta del servicio de internal structure.

Este servicio permite consultar las contrapartidas asociadas a una conocida, según la estructura interna de BBVA, por ejemplo, conocida una contrapartida, se podrá consultar el grupo o entidad a la que pertenece, o las entidades superiores o inferiores según la estructura jerárquica de entidades que maneja el banco. Se deberán conocer el código identificador de la contrapartida según los datos de la entidad, y el tipo de contrapartida o rol, que podrá ser uno de los siguientes: *Group, Matrix, Legal entity, Enterprise, Branch, Office y Calculation agent.*

El código identificador se introduce como texto en el campo marcado como Id.

Para escoger el tipo de jerarquía, pulsando sobre la pestaña situada debajo de la etiqueta *Party Hierarchy*, se desplegará una lista con los valores que se muestran en la imagen.

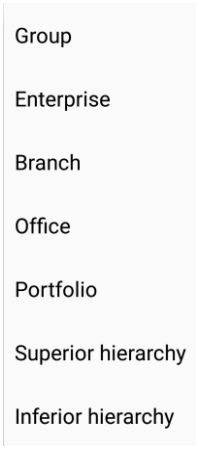


Figura 39 – Detalle de la pantalla de consulta del servicio de internal structure.

El tipo de la contrapartida de la cual se buscarán los valores asociados se seleccionará de una lista similar situada en la pestaña bajo la etiqueta *Role,* y que tendrá los valores ya comentados.

Una vez configurados los tres datos, pulsando el botón de *request*, en caso de petición efectuada correctamente, se desplegará la siguiente pantalla.

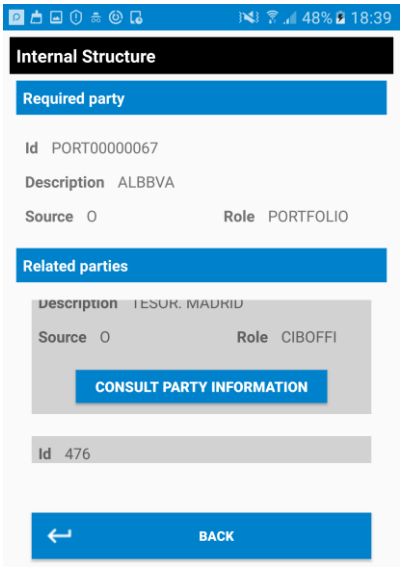


Figura 28 – Diseño de la pantalla de respuesta del servicio de internal structure.

La pantalla de resultado de consulta estará dividida en dos partes. Debajo de la etiqueta de *Required Party* se mostrará la información referente a la contrapartida consultada, en concreto, cuatro campos, el código identificador, un pequeño texto descriptivo, el rol o tipo de contrapartida y la fuente de la que se ha extraído la información.

Debajo de la etiqueta *Related Parties* se mostrará la información de las contrapartidas relacionadas con la solicitada. En caso de no existir ninguna no se mostrará nada en esa parte de la pantalla. De cada una de las contrapartidas relacionadas se mostrarán los mismos cuatro campos que los mencionados para la contrapartida consultada. En caso de existir más de una contrapartida relacionada se deberá hacer *scroll* en esa zona de la pantalla para visualizar la información referente a las demás.

Debajo de la información de las contrapartidas relacionadas, se sitúa el botón con etiqueta *consult party information*, este botón permite realizar una consulta más detallada de la información de la contrapartida relacionada empleando el servicio de consulta de información de contrapartidas, pulsando este botón se mostrará la pantalla de respuesta del servicio anterior.

Por último, pulsando el botón con etiqueta *back* se vuelve a la pantalla anterior, pudiendo efectuar una nueva consulta.

A.3. SERVICIO DE CONSULTA DE CALENDARIOS

Accediendo desde el menú al servicio de consulta de calendarios se accede a la siguiente pantalla.

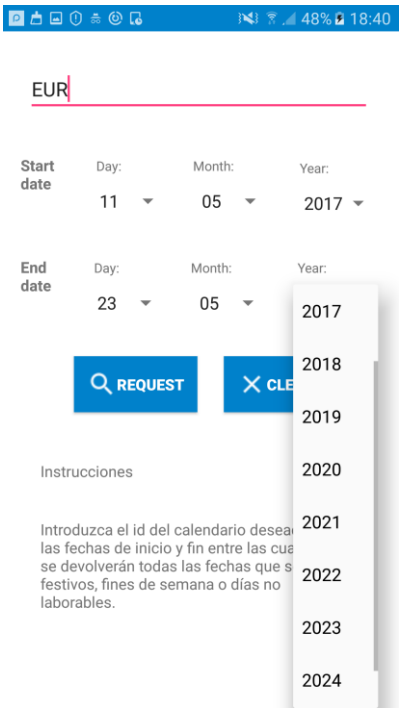


Figura 30 – Diseño de la pantalla de consulta del servicio de calendars.

Este servicio permite consultar todos los días no laborables (fines de semana, festivos u otros días no laborables) en un calendario concreto, entre dos fechas dadas.

Se debe conocer el código identificador del calendario que se desea consultar, el cual se introducirá en el campo de texto marcado como Id. Para configurar las fechas de inicio y

fin del rango se seleccionarán los respectivos día, mes y año pulsando las pestañas correspondientes. Una vez configurados los datos, pulsando el botón *request* se accede a la siguiente pantalla.

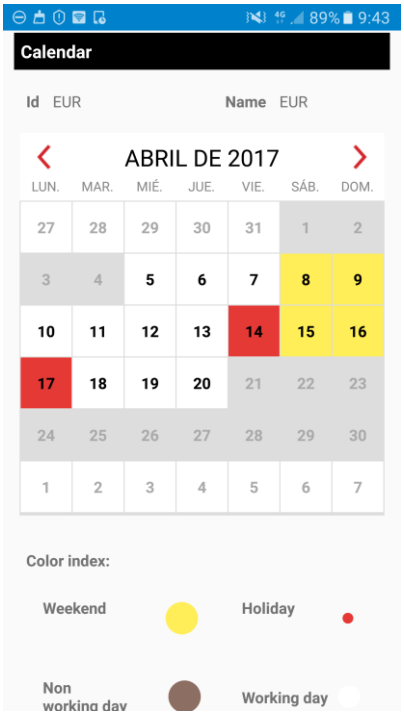


Figura 31 – Diseño de la pantalla de respuesta del servicio de calendars.

Debajo de la etiqueta de *calendar* se mostrarán dos etiquetas con el código identificador del calendario y el nombre o descripción del mismo.

A continuación se muestra la vista de un calendario por meses, inicialmente, el calendario aparecerá situado en el mes correspondiente a la fecha de inicio introducida por el usuario. Para desplazarse entre distintos meses se pulsarán las flechas rojas situadas a los lados del texto con el mes y año mostrados. Los días que quedan fuera del rango de fechas introducido aparecen en gris, los días laborables comprendidos en el rango de fechas aparecen con el fondo blanco, los fines de semana en amarillo, los festivos en rojo, y otros días no laborables en marrón. Debajo del calendario se muestra un índice con los colores

mencionados y su significado. Haciendo *scroll* en la pantalla hacia abajo aparece un botón con etiqueta *back* con el cual se accederá a la pantalla anterior de consulta y se podrá realizar una nueva búsqueda.

ANEXO B: GUÍA DE INSTALACIÓN

En esta guía se explican los requisitos y los pasos a seguir para poder utilizar la aplicación móvil desarrollada.

En primer lugar se deberá disponer de un dispositivo móvil con sistema operativo Android, de versión 4.0.3 (ICE_CREAM_SANDWICH_MR1) o superior, aunque la aplicación ha sido compilada para la versión 7.1.2 (Nougat), y con conexión de red.

Al no tratarse de una aplicación pública, no se dispone de instalador en la App Store de Android, por lo que se deberá obtener el archivo apk instalador correspondiente, un archivo llamado rdrPilot.apk.

Una vez obtenido el instalador, se deberá acceder al directorio del dispositivo en que ha sido almacenado y ejecutarlo.

En la mayoría de dispositivos Android la instalación se bloqueará ya que el apk no está firmado por Google. Para permitir la instalación del apk se deberá acceder al menú de ajustes del dispositivo. Seleccionar la opción de Pantalla Bloqueo y Seguridad, y buscar la opción Fuentes desconocidas. Para poder instalar el apk se debe habilitar esta opción durante el proceso de instalación, una vez instalada la aplicación se puede reactivar la opción. En algunos dispositivos, incluso, se da la opción de habilitar la opción únicamente para la siguiente instalación. Una vez habilitada la opción, al ejecutar el apk el sistema preguntará al usuario si desea instalar la aplicación, se pulsará Instalar.

Una vez instalada, aparecerá un icono como el mostrado en la siguiente imagen, en el menú de aplicaciones del dispositivo, para iniciar la aplicación simplemente se tendrá que pulsar sobre él.

Figura 40 – Icono de la aplicación RDR Pilot.

ANEXO C: ARCHIVOS RAML DE LOS SERVICIOS APIFICADOS

C.1. DOCUMENTO RAML DEL SERVICIO DE CONSULTA DE INFORMACIÓN DE CONTRAPARTIDAS

```
#%RAML 1.0
title: Parties
version: V03R00F00
baseUri: http://api.samplehost.com
types:
  PtyIdType:
    type: object
    properties:
      Nm:
        required: false
        type: string
        description: Party name
        maxLength: 70
        minLength: 1
      PstlAdr:
        required: false
        type: PstlAdrType
        description: Mailing address of the entity
      Id:
        required: false
        type: IdType
        description: Identification of the party
      CtryOfRes:
        required: false
        type: string
        description: Country code of country of residence
        pattern: "[A-Z]{2,2}"
  PstlAdrType:
    type: object
    properties:
      AdrTp:
        required: false
        type: string
        description: Specifies the type of address. Values can only be ADDR if
address is the complete postal address, PBOX if address is a postal office (PO)
box, HOME if address is the home address, BIZZ if is the business address, MLTO
```

```
if is the address to which mail is sent, DLVY if is the address to which delivery
is to take place.
        enum:
          - ADDR
          - PBOX
          - HOME
          - BIZZ
          - MLTO
          - DLVY
      AdrLine:
        required: false
        type: string
        description: Address in plain text
        maxLength: 70
        minLength: 1
      StrtNm:
        required: false
        type: string
        description: Name of the street
        maxLength: 70
        minLength: 1
      BldgNb:
        required: false
        type: string
        description: Building number
        maxLength: 16
        minLength: 1
      PstCd:
        required: false
        type: string
        description: Post code
        maxLength: 16
        minLength: 1
      TwnNm:
        required: false
        type: string
        description: Town name
        maxLength: 35
        minLength: 1
      CtrySubDvsn:
        required: false
        type: string
        description: Country subdivision
        maxLength: 35
        minLength: 1
      Ctry:
        required: false
        type: string
        description: Code to identify a country, a dependency, or another area of
particular geopolitical interest, on the basis of country names obtained from the
United Nations (ISO 3166, Alpha-2 code).
        pattern: "[A-Z]{2,2}"
  IdType:
```

```
    type: object
    properties:
      OrgId:
        required: false
        type: OrgIdType
        description: Organisation identification
      PrvtId:
        required: true
        type: PrvtIdType
        description: Private identification, identification of a person, eg.
passport
  OrgIdType:
    type: object
    properties:
      BIC:
        required: false
        type: string
        description: Business Identifier Code of the legal entity. Code allocated
to a financial institution by the ISO 9362
        pattern: "[A-Z]{6,6}[A-Z2-9][A-NP-Z0-9]([A-Z0-9]{3,3}){0,1}"
      IBEI:
        required: false
        type: string
        description: International Business Entity Identifier to uniquely
identify business entities playing a role in the lifecycle of and events related
to a financial instrument.
        pattern: "[A-Z]{2,2}[B-DF-HJ-NP-TV-XZ0-9]{7,7}[0-9]{1,1}"
      BEI:
        required: false
        type: string
        description: Business Entity Identifier.
        pattern: "[A-Z]{6,6}[A-Z2-9][A-NP-Z0-9]([A-Z0-9]{3,3}){0,1}"
      EANGLN:
        required: false
        type: string
        description: Global Location Number. A non-significant reference number
used to identify legal entities, functional entities or physical entities
according to the European Association for Numbering (EAN) numbering scheme rules.
        pattern: "[0-9]{13,13}"
      USCHU:
        required: false
        type: string
        description: (United States) Clearing House Interbank Payments System
(CHIPS) Universal Identification (UID).
        pattern: "CH[0-9]{6,6}"
      DUNS:
        required: false
        type: string
        description: Data Universal Numbering System. A unique identification
number provided by Dun & Bradstreet to identify an organization.
        pattern: "[0-9]{9,9}"
      BkPtyId:
        required: false
```

```
        type: string
        description: Bank Party Identification, assignment to identify a
relationship as defined between the bank and its client
        maxLength: 35
        minLength: 1
      TaxIdNb:
        required: false
        type: string
        description: Tax Identification Number, number assigned by a tax
authority to an entity
        maxLength: 35
        minLength: 1
      PrtryId:
        required: false
        type: PrtryIdType
        description: Proprietary Identification
  PrtryIdType:
    type: object
    properties:
      Id:
        required: false
        type: string
        description: Identification of the entity
        maxLength: 35
        minLength: 1
      Issr:
        required: false
        type: string
        description: Issuer, identification of the issuer of the reference
document type
        maxLength: 35
        minLength: 1
  PrvtIdType:
    type: object
    description: There will be selected only one property from values DrvrsLicNb,
CstmrNb, SclSctyNb, AlnRegnNb, PsptNb, TaxIdNb, IdntyCardNb, MplyrIdNb,
DtAndPlcOfBirth and OthrId.
    properties:
      DrvrsLicNb:
        required: false
        type: string
        description: Driver license number
        maxLength: 35
        minLength: 1
      CstmrNb:
        required: false
        type: string
        description: Customer number
        maxLength: 35
        minLength: 1
      SclSctyNb:
        required: false
        type: string
```

```
        description: Social security number
        maxLength: 35
        minLength: 1
      AlnRegnNb:
        required: false
        type: string
        description: Alien registration number (number assigned to identify
foreign nationals)
        maxLength: 35
        minLength: 1
      PsptNb:
        required: false
        type: string
        description: Passport number
        maxLength: 35
        minLength: 1
      TaxIdNb:
        required: false
        type: string
        description: Tax identification number
        maxLength: 35
        minLength: 1
      IdntyCardNb:
        required: false
        type: string
        description: Identity card number
        maxLength: 35
        minLength: 1
      MplyrIdNb:
        required: false
        type: string
        description: Employer identification number
        maxLength: 35
        minLength: 1
      DtAndPlcOfBirth:
        required: false
        type: DtAndPlcOfBirthType
        description: Date And Place Of Birth
      OthrId:
        required: false
        type: OtherIdType
        description: Identifier issued to a person for which no specific
identifier has been defined
      Issr:
        required: false
        type: string
        description: Issuer, identification of the issuer of the reference
document type
        maxLength: 35
        minLength: 1
  DtAndPlcOfBirthType:
    type: object
    properties:
```

```
      BirthDt:
        required: false
        type: string
        description: Birthday date
        maxLength: 35
        minLength: 1
      PrvcOfBirth:
        required: false
        type: string
        description: Province of birth
        maxLength: 35
        minLength: 1
      CityOfBirth:
        required: false
        type: string
        description: City of birth
        maxLength: 35
        minLength: 1
      CtryOfBirth:
        required: false
        type: string
        description: Country code of the country of birth
        pattern: "[A-Z]{2,2}"
  OtherIdType:
    type: object
    properties:
      Id:
        required: false
        type: string
        description: Identifier
        maxLength: 35
        minLength: 1
      IdTp:
        required: false
        type: string
        maxLength: 35
        minLength: 1


/parties:
  displayName: Trading Parties
    description:Parties resource with all the methods related to this entity
  get:
    description: Gets Party related information
    queryParameters:
      Id:
        description: Party ID
        type: string
        required: true
        maxLength: 35
        minLength: 1
        example: "1"
      Issr:
```

```
      description: Branch related to the party
      type: string
      required: true
      maxLength: 35
      minLength: 1
  responses:
    200:
      body:
        application/json:
          type: PtyIdType
          description: OK
          example: |
            {
            "Nm": "Sica Corporation",
            "Id": {
              "PrvtId": {
                "OthrId": {
                  "Id": "0000003L",
                  "IdTp": "CBI Identifier"
                  }
                }
              }
            }

    206:
      body:
        application/json:
          type: PtyIdType
          description: Partial Content
          example: |
            {
            "Nm": "Sica Corporation",
            "Id": {
              "PrvtId": {
                "OthrId": {
                  "Id": "0000003L",
                  "IdTp": "CBI Identifier"
                  }
                }
              }
            }
```

C.2. DOCUMENTO RAML DEL SERVICIO DE CONSULTA DE JERARQUÍA INTERNA

```
#%RAML 1.0
title: InternalStructure
version: v1
baseUri: http://api.samplehost.com
types:
  PartyDetailType:
    type: object
    properties:
      ID:
        required: false
        type: string
        description: Value of the required party.
      Source:
        required: false
        type: string
        description: |
          BBVA's additional Party ID Source.
          O - BBVA RDR System
          P - BBVA eFX Users System
        enum:
          - O
          - P
      Description:
        required: false
        type: string
        description: Description of the required party.
      Role:
        required: false
        type: string
        description: Type of the required party.
      RelatedParties:
        required: false
        type: RelatedPartyType[]
        description: Information of the related party.

  RelatedPartyType:
    type: object
    properties:
      ID:
        required: false
        type: string
        description: Value of the related party.
      Description:
        required: false
        type: string
        description: Description of the related party.
      Source:
```

```
      required: false
      type: string
      description: |
        BBVA's additional Party ID Source.
        O - BBVA RDR System
        P - BBVA eFX Users System
      enum:
        - O
        - P
    Role:
      required: false
      type: string
      description: Type of the related party.

PartyHierarchyType:
  type: integer
  description: |
    Type of hierarchy required.
    0 - Group
    1 - Enterp
    2 - Branch
    3 - Office
    4 - Portfolio
    5 - Superior hierarchy
    6 - Inferior hierarchy
  enum:
    - 0
    - 1
    - 2
    - 3
    - 4
    - 5
    - 6

RoleType:
  type: integer
  description: |
    Role of the required party.
    1002 - Group
    1003 - MATRIX
    1004 - LEGALENT
    1005 - ENTERPRISE
    1006 - BRANCH
    1007 - Office
    1008 - Calculation Agent
  enum:
    - 1002
    - 1003
    - 1004
    - 1005
    - 1006
    - 1007
    - 1008
```

```
/internalStructure:
  displayName: Internal Structure
  get:
    description: Returns the information of the parties of the internal structure
of BBVA.
    queryParameters:
      ID:
        description: Value of the required party.
        type: string
        required: true
        example: Mexico
      PartyHierarchy:
        description: Type of hierarchy required.
        type: PartyHierarchyType
        required: true
        example: 5
      Role:
        description: Role of the required party.
        type: RoleType
        required: true
        example: 1003

    responses:
      200:
        body:
          application/json:
            type: PartyDetailType
            description:
            example: |
              {
                "ID" : "PORT00000067",
                "Source" : "O",
                "Description" : "ALBBVA",
                "Role" : "PORTFOLIO",
                "RelatedParties" : [{
                  "ID" : "455",
                  "Source" : "O",
                  "Description" : "TESOR. MADRID",
                  "Role" : "CIBOFFI"
                }]
              }
```

C.3. DOCUMENTO RAML DEL SERVICIO DE CONSULTA DE CALENDARIOS

```
#%RAML 1.0
title: Calendars
version: v1
baseUri: http://api.samplehost.com
types:
  CalendarDataType:
    type: object
    properties:
      ID:
        description: Unique identifier of the calendar.
        type: string
        required: true

      Name:
        description: Name of the calendar.
        type: string
        required: true

      CalendarDateData:
        description: Array of the required dates.
        type: CalendarDateDataType[]
        required: false

  CalendarDateDataType:
    type: object
    properties:
      DateValue:
        description: The actual value of the date calendar in YYYYMMDD format.
        type: integer
        required: true

      DateType:
        description: |
          The type of calendar day, either a business or a holiday day.
          0 - Business Day
          1 - Holiday
          2 - Weekend
          3 - Non Working Day
        type: integer
        enum:
          - 0
          - 1
          - 2
          - 3
        required: true
```

```
/calendar:
  displayName: Calendars
  get:
    description: Returns all non business days within a specific date range.
    queryParameters:
      ID:
        description: Unique identifier of the calendar.
        type: string
        required: true
        example: EUR

      StartDate:
        description: Start date of the calendar to request a specific date range
in YYYYMMDD format.
        type: integer
        required: true
        example: 20170220

      EndDate:
        description: End date of the calendar to request a specific date range in
YYYYMMDD format.
        type: integer
        required: true
        example: 20170302

    responses:
      200:
        body:
          application/json:
            type: CalendarDataType
            example: |
              {
                "ID" : "EUR",
                "Name" : "EUR",
                "CalendarDateData" : [{
                    "DateValue" : 20170225,
                    "DateType" : 2
                },{
                  "DateValue" : 20170226,
                  "DateType" : 2
                }]

              }
```

C.4. FRAGMENTO DE LOS ARCHIVOS DE TRATAMIENTO DE ERRORES

A continuación se incluyen los dos fragmentos de los archivos RAML referentes al tratamiento de los errores. Al ser idénticos en los tres servicios se han omitido y se incluyen a continuación.

El primer fragmento corresponde a la descripción del cuerpo de los mensajes de error devueltos por la aplicación en formato JSON. Todos los errores tendrán 4 campos: el primero con el código de error HTTP (errores 400 y 500), la descripción corresponderá a la descripción del tipo de error (*forbidden, not found…*), otro campo indicando el nivel del error y por último se incluirá un campo con la traza completa del error generado.

Este fragmento se incluye en los tres casos a continuación de la descripción de la estructura de la respuesta del servicio.

```
Error:
  type: object
  properties:
    code:
      required: true
      type: string
      description: Error code
    description:
      required: true
      type: string
      description: Error description
    level:
      required: true
      type: string
      description: Error level
    stacktrace:
      required: true
      type: string
      description: Error stacktrace
```

El segundo fragmento indica las posibles respuestas asociadas a errores generados en la invocación al servicio, coincide con los tipos de errores HTTP que pueden generarse en los servicios.

El fragmento se incluye en los tres casos a continuación de la descripción de la respuesta con código 200 (respuesta correcta).

```
400:
  body:
    application/json:
      type: Error
      description: Bad Request
401:
  body:
    application/json:
      type: Error []
      description: Unauthorized
403:
  body:
    application/json:
      type: Error []
      description: Forbbiden
404:
  body:
    application/json:
      type: Error []
      description: Not Found
500:
  body:
    application/json:
      type: Error []
      description: Internal Server Error
503:
  body:
    application/json:
      type: Error []
      description: Service Unavailable
504:
  body:
    application/json:
      type: Error []
      description: Gateway Timeout
```