Grado en Ingeniería Informática

Trabajo Final de Grado

# Diseño e implementación de una aplicación web para la gestión de remesas bancarias

*Autor:*
Yoshiya Magaña Martinez

*Supervisor:*
Pablo Domínguez Cuevas
*Tutor académico:*
María José Aramburu Cabo

Fecha de lectura: 14 de Septiembre de 2017
Curso académico 2016/2017

## Resumen

Este documento contiene la especificación, planificación, diseño e implementación del proyecto de fin de grado realizado por el alumno Yoshiya Magaña Martínez en la empresa PDCnet Spain S.L. durante una estancia de 300 horas en las oficinas de la empresa en el Grao de Castellón.

El proyecto realizado consiste en una aplicación para la creación de los ficheros que se deben enviar a un banco para realizar el cobro de facturas. La normativa que regula el formato de estos ficheros cambió recientemente a nivel europeo, por lo que las empresas deben implementar nuevos programas o modificar los existentes para adaptarse a la normativa. En este caso, la empresa PDCnet quiere ofrecer a sus clientes un nuevo programa para realizar tal función. Esto ha implicado realizar todo el diseño y la implementación de la aplicación.

La aplicación se ha desarrollado según los requisitos del cliente en un plazo ligeramente superior al estimado inicialmente. El resultado ha sido satisfactorio para la empresa, por lo que en poco tiempo podrá ofrecerlo a sus clientes.

## Palabras clave

Facturación, módulo de remesas, ASP.net

## Keywords

Billing, direct debits module, ASP.net

# Tabla de contenidos

# Índice de figuras

# Índice de tablas

# Capítulo 1. Introducción

## 1.1 Contexto y motivación del proyecto

Este proyecto ha sido desarrollado por el estudiante Yoshiya Magaña Martinez para la empresa PDCnet Spain, ubicada en El Grao de Castellón, como trabajo de fin de grado con el objetivo de terminar los estudios universitarios de Grado de Ingeniería Informática de la Universitat Jaume I de Castelló, con especialización en el itinerario de Ingeniería del Software.

PDCnet Spain S.L. nació para cubrir las necesidades de las empresas que se dedican a dar servicios de prevención de riesgos laborales, así como de formación privada y subvencionada. Su plataforma de producción reside en los *data centers* de Microsoft en Europa y EEUU y se basa en la tecnología Microsoft Azure. En 2012 renovaron la marca PDCnet.es, y bajo el concepto "computación en la nube", nació Nubenet. Actualmente, el software que produce se ha diversificado y ha evolucionado tanto, gracias a la colaboración de sus clientes, que ya ha permitido a las empresas que lo han solicitado superar la norma ISO 9001.

PDCnet se desarrolla en el ámbito de gestión de personal (recursos humanos) de las empresas, permitiendo realizar un seguimiento y control de los empleados en cuanto a los controles de prevención de riesgos laborales, vigilancia de la salud, evaluación de riesgos y formación. También facilita la coordinación de actividades empresariales entre la contrata y subcontrata y el control de accesos. Además, ofrece un CRM para la completa gestión comercial, con control de avisos, reclamaciones, llamadas, gestión de proyectos, etc., así como herramientas del área administrativa para la realización de presupuestos, contratos, facturas e impagados.

PDCnet quiere utilizar siempre la tecnología más reciente, por lo que desarrolla sus aplicaciones basadas en web con tecnología HTML5/CSS3 con un diseño adaptable, lo que permite ejecutar sus aplicaciones desde cualquier dispositivo de forma muy sencilla [1].

En su afán de ofrecer a sus clientes nuevas herramientas, con este proyecto PDCnet desea facilitar a sus clientes la tarea de generar remesas, es decir, el documento que una entidad bancaria necesita para realizar el cobro de facturas a clientes. Esta labor es principalmente del departamento administrativo que realiza la contabilidad de las empresas. La nueva aplicación es necesaria para que los clientes no tengan que recurrir a una aplicación de terceros para realizar la conversión entre el formato de texto plano y el XML exigido por los bancos y, además, ahorren tiempo en la creación de las remesas.

Los usuarios de la aplicación son los mismos que actualmente utilizan sus aplicaciones *netprevención, netformación y netcrm*.

## 1.2 Objetivos del proyecto

Este proyecto se enmarca en el contexto de la transición de los requisitos de los bancos en cuanto al fichero que deben presentar las empresas para generar remesas de facturas y poder efectuar su cobro. Hasta febrero de 2016 se aceptaban ficheros de texto plano conocidos como "AEB Cuaderno Norma 19" para realizar las domiciliaciones. A partir de entonces, y con el objetivo de crear la Zona Única Europea de Pagos, se estableció un formato común para este fichero a nivel europeo. En lugar de utilizar un fichero de texto plano se utiliza un fichero XML (eXtensible Markup Language) que sigue un estándar claramente definido al que todas las entidades bancarias deben adherirse.

El objetivo principal de la aplicación es gestionar remesas bancarias, tanto la creación como la edición y modificación de las mismas, para conocer qué facturas se han enviado a una entidad bancaria para su cobro, con todos los datos relacionados, y cuáles están pendientes de cobro. Este objetivo general se compone a su vez de otros objetivos que están directamente relacionados:

1. Crear un fichero válido para el adeudo directo SEPA, que cumpla la normativa vigente según el estándar XML, con los campos y la información obligatoria para que los bancos puedan producir el cobro de los documentos (facturas) incluidos sin problemas.

2. Facilitar el proceso de gestión de las remesas permitiendo al usuario acceder al sistema desde cualquier dispositivo mediante una interfaz web *responsive* para que se adapte al tamaño de dispositivo sin tener que estar ampliando o reduciendo los contenidos y visualizarlos correctamente.

El resultado esperado es una aplicación web que se ejecute en un servidor a la que los usuarios tienen acceso mediante una interfaz web adaptable, y que permita a los mismos crear remesas pudiendo filtrar fácilmente las facturas con el fin de conocer cuáles están ya cobradas y cuáles no, así como editar y borrar las remesas en caso de tener que modificar las facturas que se incluyen o cancelar por completo la remesa.

## 1.3 Estructura de la memoria

Esta memoria consta de varios apartados que se organizan según diferentes temas relacionados con el desarrollo del proyecto. En el capítulo 2 se realiza una breve descripción del proyecto y las tecnologías utilizadas, así como su arquitectura. El capítulo 3 trata sobre la planificación del proyecto, la metodología de trabajo utilizada, la estimación de los recursos y costes del proyecto y el seguimiento del desarrollo con respecto a la planificación inicial. El capítulo 4 detalla el análisis y el diseño de la aplicación y su arquitectura. El capítulo 5 destaca los puntos importantes de la implementación y la validación del sistema y las técnicas de programación utilizadas. Por último, el capítulo 6 relata las conclusiones obtenidas del desarrollo, terminando con la bibliografía que cita las fuentes utilizadas para la redacción de esta memoria. El último apartado contiene los anexos a la memoria con el diseño de la especificación del XML y el manual de usuario de la aplicación.

# Capítulo 2. Descripción del proyecto

El proyecto se ha desarrollado partiendo de una lista de requisitos iniciales facilitada por la empresa y que indicaba los resultados generales que se debían obtener. No existía ninguna implementación inicial, es decir, se debía programar todo desde cero.

## 2.1 Requisitos

Tras investigar y realizar más preguntas a la empresa, se obtuvo la siguiente lista de requisitos más detallada:

1. La interfaz del lado cliente debe ser *responsive*, conocido en castellano como diseño web adaptable y que se define como "*una filosofía de diseño y desarrollo cuyo objetivo es adaptar la apariencia de las páginas web al dispositivo que se esté utilizando para visitarlas*" [2]. Para esto se debe utilizar el *framework* Bootstrap (el más conocido para desarrollar páginas web responsive) combinando HTML5, CSS3, Javascript, jQuery y *plugins* de jQuery para una navegación más cómoda.

2. Se debe poder controlar el acceso a la aplicación mediante autenticación de usuarios. Para ello se debe disponer de un panel de control de usuarios. Los usuarios se deben poder clasificar según distintos roles, que tienen diferentes permisos: sólo lectura (perfil "invitado"), lectura y escritura (perfil "usuario"), o lectura, escritura y borrado (perfil "administrador"). Además, los usuarios administradores deben poder gestionar y cambiar los permisos de los demás usuarios. Para el registro de usuarios se debe ofrecer la posibilidad de utilizar el protocolo de autenticación OAuth, que permite a los usuarios registrarse utilizando los datos de sus cuentas en plataformas externas tales como redes sociales. Esto permite al usuario no tener que introducir sus datos de nuevo en el formulario de registro pudiendo utilizar los mismos que en otras páginas. Y aún más importante, le permite no tener que recordar una nueva contraseña, dado que una vez realizado el registro, su cuenta queda ligada a la cuenta externa utilizada y después basta con iniciar sesión con la cuenta externa para identificar la cuenta del usuario e iniciar sesión en la aplicación automáticamente.

3. La aplicación debe almacenar los datos de las remesas generadas en una base de datos que se debe crear. Al igual que la aplicación, esta base de datos se debe alojar en Azure. Azure es una colección de servicios Microsoft integrados en la nube, que los desarrolladores pueden utilizar para crear, implementar y administrar aplicaciones a través de una red global de centros de datos, pagando una cuota mensual flexible según los recursos consumidos. Sobre esta base de datos se deben realizar, mediante la interfaz web, las operaciones CRUD (crear, leer, actualizar y borrar) de las remesas y entidades relacionadas.

4. En la ventana de creación de una remesa se deben poder elegir las facturas que se desean incluir, y una vez seleccionadas, se deben enviar en formato JSON a una web API. Una Web API es un marco que facilita la creación de servicios HTTP disponibles para una amplia variedad de dispositivos clientes, entre los que se incluyen

exploradores y dispositivos móviles. Se caracteriza por seguir el estilo de arquitectura de software REST, que se basa en un protocolo de transferencia cliente-servidor sin estado y utilizando un conjunto de operaciones bien definidas que permite obtener datos o indicar la ejecución sobre datos en cualquier formato.

5. La Web API debe recibir los datos en JSON y devolver un fichero XML válido según la normativa europea SEPA. Este fichero se debe validar correctamente con un validador SEPA. Aunque no se indica ninguno en concreto se debe buscar y utilizar uno *online* para las pruebas. Aunque el XML schema se presenta en dos variantes, en la primera versión de la aplicación sólo se debe implementar la primera variante, nombrada SEPA Direct Debits Core, que se define a continuación:

    - ❏ SEPA direct debits SDD (XML basado en UNIFI ISO 20022 con esquema PAIN.008. SDD Esquema básico - core). Se trata de un instrumento que permite realizar cobros en euros, adeudando la cuenta del deudor ya sea consumidor, empresa o autónomo. Este es el esquema básico que se debe implementar inicialmente.

## 2.2 Alcance

Con el alcance del proyecto indicamos qué sistemas externos se pueden ver involucrados o afectados por el desarrollo o pueden ser necesarios para el uso de la aplicación, así como qué departamentos utilizan el sistema. Se pueden diferenciar distintos tipos de alcance:

- ❏ Con respecto al alcance funcional, el sistema debe permitir a los usuarios registrarse para poder utilizarlo. Una vez registrado, un usuario debe poder crear una remesa eligiendo las facturas que quiere incluir en la remesa, para después poder descargar el fichero XML que enviará al banco para el cobro de las facturas seleccionadas. Para poder crear una remesa se necesitan ciertos datos, como la cuenta donde se deben realizar los ingresos. En empresas medianas o grandes es normal tener varias cuentas en las que reciben los pagos, con lo cual, también se deben poder mantener cuentas bancarias. Dichas cuentas bancarias están ligadas a una entidad bancaria, cuyos datos deben aparecer en el fichero XML que se envía al banco, por lo tanto, también es preciso poder mantener información sobre las entidades bancarias. Resumiendo, la aplicación debe permitir la gestión de usuarios, entidades bancarias, cuentas bancarias, y remesas, pudiendo obtener de las remesas un fichero XML válido para realizar el cobro de las facturas incluidas.

- ❏ Con respecto al alcance organizativo, la aplicación está ideada únicamente para ser utilizada por el departamento de administración y contabilidad, que es el encargado de emitir las facturas, crear las remesas según su vencimiento y marcarlas como pagadas cuando el cobro se ha efectuado, entre otras cosas.

- ❏ En cuanto al alcance informático, la aplicación se desea alojar utilizando los servicios en la nube de Microsoft Azure, con lo cual se debe diseñar para poder residir en varios servidores distintos simultáneamente. Se debe conectar con dos bases de datos distintas, la de la propia aplicación y una base de datos de una aplicación de

facturación externa de la cual obtendrá las facturas para incluir en las remesas. No se debe conectar con ningún otro sistema informático.

## 2.3 Restricciones

El proyecto presenta una serie de restricciones que se deben tener en cuenta antes y durante su desarrollo con el fin obtener un resultado satisfactorio. Estas restricciones son principalmente temporales.

- Las restricciones temporales se basan en que el proyecto debe estar finalizado en el periodo de prácticas acordado con la empresa que comprende 300 horas de desarrollo en las oficinas de la empresa. No pudiendo extenderse, puesto que el contrato termina y no hay cobertura laboral legal.

- En cuanto a las restricciones económicas, no existen puesto que el estudiante desarrolla el proyecto de forma gratuita para la empresa y tanto el hardware como el software que se utiliza en el entorno de desarrollo es propiedad de la empresa. Para el IDE (Integrated Development Environment) se utiliza una versión gratuita de Visual Studio, así no se incurren en gastos de licencias.

- Sobre las restricciones humanas, el proyecto se debe realizar exclusivamente por un desarrollador, siendo el estudiante en prácticas, bajo la supervisión de un encargado/a en la empresa y un tutor/a en la Universidad Jaume I.

- En cuanto a las restricciones tecnológicas y materiales, no se requiere nada especial con lo cual, lo facilitado por la empresa es suficiente, no siendo necesario adquirir nuevos equipos ni dispositivos.

## 2.4 Arquitectura tecnológica

El proyecto se ha desarrollado partiendo del *framework* de aplicaciones web ASP.NET MVC. Este es un *framework* de aplicaciones web que implementa el patrón modelo-vista-controlador (MVC). Basado en ASP.NET, permite a los desarrolladores de software construir una aplicación web como una composición de tres funciones: modelo, vista y controlador [3]. Este patrón de arquitectura de software separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son: el modelo, la vista y el controlador (ver figura 1). Es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario. Se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento [4].


*Figura 1. Componentes del patrón Modelo-Vista-Controlador y sus roles [5]*

La estructura del proyecto se compone de varios sub-proyectos siguiendo los patrones de diseño MVC, Repositorio, Servicio, Unidad de Trabajo, Inversión de Control e Inyección de Dependencias (ver figura 2), que se explican con más detalle en el capítulo 4 de análisis y diseño del sistema. Para esto último se utiliza el *framework* Unity de Microsoft para aplicaciones ASP.NET, que se explica en el capítulo 5 de detalles de implementación.

En el proceso de creación de una remesa los datos de las facturas a incluir se envían en JSON a una web API que devuelve el fichero XML correspondiente. Dicha web API también es un sub-proyecto independiente.

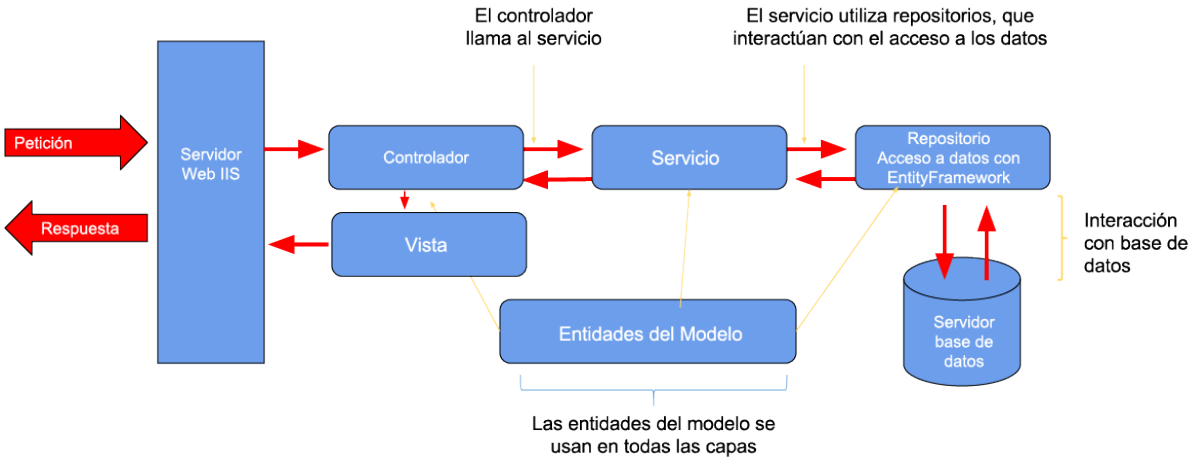



*Figura 2. Vista general de los patrones utilizados en la estructura de la aplicación*

El lado servidor de la aplicación se ha codificado en C# utilizando el ORM Entity Framework, un conjunto de tecnologías de ADO.NET que permiten el desarrollo de aplicaciones de software orientadas a datos utilizando un mapeo objeto-relacional para convertir los datos entre el sistema de tipos utilizando una base de datos como motor de persistencia.

Para la codificación del lado cliente se ha empleado HTML5, CSS3 y Javascript con librerías como jQuery y sus *plugins*, utilizando como base el *framework responsive* Bootstrap 3 para crear un diseño elegante e intuitivo que se adapta al tamaño del dispositivo del usuario. El código fuente de las vistas no es HTML puro puesto que se utiliza una sintaxis propia de ASP.net denominada Razor, que se explica en el capítulo 5 de detalles de implementación.

El proyecto está enlazado con dos bases de datos distintas (ver figura 3), una ya existente de una aplicación externa que contiene los datos de las facturas emitidas y la información de los clientes a la que esta aplicación tiene acceso de sólo lectura, y otra nueva que se ha creado para la propia aplicación, con el fin de gestionar los usuarios y remesas y todos sus datos relacionados. Ambas bases de datos de SQL Server están alojadas en la nube distribuida de Microsoft Azure. Para leer de los datos existentes de la base de datos externa se utiliza la técnica Database-First de Entity Framework que crea las clases del modelo orientado a objetos partiendo de una base de datos ya creada previamente. Para crear la base de datos nueva se ha utilizado la técnica Code-First, que funciona justo al revés. Es decir, primero se definen las clases del modelo y al ejecutar la aplicación por primera vez se crea la base de datos y las tablas para dichas entidades automáticamente, pudiendo detectar cambios en estas clases y aplicarlos al diseño físico de la base de datos sin intervención del programador. La conversión de tipos se ha personalizado con un mapeo específico utilizando la técnica Fluent API de Entity Framework. Para las entidades que no se ha definido un mapeo específico, se aplican los valores de conversión por defecto que establece Entity Framework. Los detalles de cada técnica y las diferencias entre cada una se explican también en el capítulo 5 de implementación.

Para la construcción de consultas sobre las base de datos y el filtrado de entidades a mostrar en las vistas se ha utilizado LINQ (Language-Integrated Query), que es un componente de la plataforma Microsoft .NET que agrega capacidades de consulta a datos de manera nativa a los lenguajes .NET. Este componente extiende el lenguaje a través de las llamadas “expresiones de consulta”, que son parecidas a las sentencias SQL y pueden ser usadas para extraer y procesar convenientemente datos de *arrays*, clases enumerables, documentos XML, bases de datos relacionales y fuentes de terceros.

El proyecto se ha desarrollado de forma local, utilizando el IDE de Microsoft Visual Studio 2017 y un servidor IIS Express (Internet Information Server) instalado en el equipo asignado al estudiante. En el momento en que se considere oportuno, la aplicación se cargará también a la nube de Azure donde se lanzará a producción, con lo cual tanto la aplicación como las bases de datos residirán en Azure.

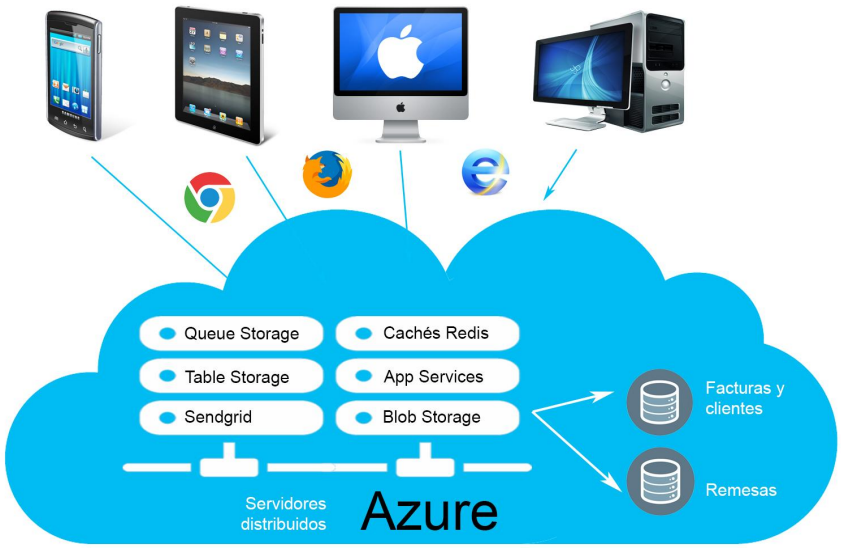



*Figura 3. Arquitectura de la aplicación en la nube de Azure*

## 2.5 Situación previa y desafíos

Según se ha mencionado anteriormente, hasta el momento la empresa no disponía de ninguna herramienta propia para crear remesas y obtener el fichero XML válido. Con lo cual, el proyecto presentaba el inconveniente de tener que crear una solución partiendo de la nada, teniendo que buscar toda la información tanto contextual para entender qué era una remesa y cómo se usa, como técnica para que cumpliera los requisitos del XML Schema.

Por lo tanto, el desafío ha sido especificar los requisitos y desarrollar la aplicación de forma que los cumpliera, y sobre todo, que fuera fácil de utilizar e intuitiva para los usuarios, dentro del plazo de tiempo de la estancia en prácticas. El mayor reto tecnológico ha sido integrar todas las fuentes de datos y patrones de diseño mencionados en el apartado 2.4 Arquitectura Tecnológica, dado que comprender cada uno por separado puede parecer sencillo pero integrar todos ellos en un mismo proyecto resulta más complicado por la gran variedad de formatos que se debían transformar y unificar en un solo esquema.

# Capítulo 3. Planificación del proyecto

Este capítulo trata sobre la metodología de trabajo elegida para la planificación y desarrollo del proyecto, su justificación y sus ventajas, así como los resultados y desviaciones con respecto de la planificación inicial y las lecciones aprendidas.

## 3.1 Metodología de trabajo

La metodología de trabajo elegida para el desarrollo del proyecto ha sido Scrum, puesto que es la forma de trabajar de la empresa PDCnet y también ha sido la metodología utilizada para el desarrollo de proyectos de las asignaturas de los últimos cursos del Grado.

Scrum es el nombre con el que se denomina a la metodología de trabajo ágil caracterizada por:

- Adoptar una estrategia de desarrollo incremental, en lugar de la planificación y ejecución completa del producto en fases distintas como se realiza tradicionalmente.

- Basar la calidad del resultado más en el conocimiento de las personas en equipos auto-organizados, que en la calidad de los procesos empleados. Gestión regular de las expectativas del cliente y basada en resultados tangibles, en lugar de realizar el desarrollo completo y luego realizar la revisión con el cliente.

- Solapamiento de las diferentes fases del desarrollo, en lugar de realizar una tras otra en un ciclo secuencial o en cascada.

En la metodología Scrum se definen varios roles dentro del equipo según la función que tiene cada uno:

- Product Owner: es el cliente, y se asegura de que el equipo Scrum trabaja de forma adecuada desde la perspectiva del negocio y entiende los requisitos del sistema realizando un seguimiento del desarrollo y priorizando las tareas. Escribe historias de usuario, las prioriza, y las coloca en el Product Backlog, que se define como el conjunto de funcionalidades que debe ofrecer el producto una vez terminado. En este caso el Product Owner es la empresa PDCnet, puesto que es quien define y prioriza los requisitos que debe cumplir la aplicación y realiza un seguimiento del desarrollo.

- Scrum Master: su trabajo primario es eliminar los obstáculos que impiden que el equipo alcance los objetivos definidos en cada fase del desarrollo, actuando como una protección entre el equipo y cualquier influencia que le distraiga y haciendo que las reglas se cumplan.

- Equipo Scrum: lo forma el conjunto de empleados que se encarga de llevar a cabo el proyecto. Normalmente se compone de personas con habilidades transversales que son capaces de realizar distintas tareas como pueden ser diseño, desarrollo, pruebas

o documentación, en lugar de tener equipos claramente diferenciados que sólo se dedican a una tarea, como es habitual en la forma de trabajar en equipos tradicional. En este caso tanto el Scrum Master como el equipo Scrum es el propio estudiante que desarrolla el proyecto, puesto que es el encargado de asegurar que el trabajo se desarrolla según las metas definidas y de llevar a cabo el propio proyecto en todos sus aspectos.

Vistas las ventajas de esta metodología ágil, que prioriza la comunicación con el cliente y la mejora continua frente a la extensa documentación y análisis y el desarrollo por fases, se ha decidido que es la mejor forma de trabajar para llevar a cabo el proyecto con éxito.

A lo largo del desarrollo se ha podido apreciar que esta forma de trabajar ha sido muy útil, puesto que al revisar el estado del proyecto cada poco tiempo con el cliente (la empresa) se han ido detectando errores o mejoras que había que introducir a medida que las funcionalidades se iban implementando. Esto ha permitido ir terminando dichas funcionalidades al ser verificadas por el cliente a medida que se desarrollaba la aplicación, en lugar de desarrollarla completamente y luego realizar la verificación, cosa que hubiera provocado tener que modificar mucho código para ajustarlo a los cambios que se han requerido durante el desarrollo. Cuanto antes se detecta un error o algo que no se ha implementado correctamente por un malentendido con el cliente o durante el análisis de requisitos, antes se puede corregir. Esta es una de las mayores ventajas de las metodologías ágiles.

## 3.2 Planificación

Inicialmente el proyecto se ha planificado siguiendo una propuesta de la asignatura para el desarrollo de esta clase de proyectos basado en la experiencia previa. Esta planificación se divide en varias fases. Estas fases se pueden desglosar en el listado de tareas que aparece en la tabla 1 con su correspondiente estimación temporal, teniendo en cuenta que deben emplearse 300 horas en la empresa con el horario a continuación:

Fecha de comienzo: 20/02/2017. Fecha de fin estimada: 12/05/2017
Horario: Lunes, Martes, Viernes: 9:00 - 14:00. Miércoles, Jueves: 9:00 - 14:00, 15:00 - 19:00

Queda excluida la semana de Magdalena (fiestas locales de Castellón de la Plana) y todos los festivos.

| **Nº** | **Tarea** | **Horas** | **Depend.** | **Inicio** | **Fin** |
|---|---|---|---|---|---|
| **1** | **Desarrollar propuesta técnica** | **50 h** | | **20/02/17** | **06/03/17** |
| **1.1** | **Inicio** | **9 h** | | **20/02/17** | **22/02/17** |
| 1.1.1 | Definir proyecto con tutor y supervisor | 1 | | 20/02/17 | 20/02/17 |
| 1.1.2 | Definir metodología de trabajo | 5 | 1.1.1 | 21/02/17 | 21/02/17 |

| | | | | | |
|---|---|---|---|---|---|
| 1.1.3 | Definir formato y estándares de trabajo | 3 | 1.1.1 | 22/02/17 | 22/02/17 |
| **1.2** | **Documentar y planificar proyecto** | **21 h** | | **22/02/17** | **27/02/17** |
| 1.2.1 | Revisar contexto y buscar información | 16 | 1.1 | 22/02/17 | 24/02/17 |
| 1.2.2 | Identificar alcance y objetivos | 5 | 1.1 | 24/02/17 | 27/02/17 |
| **1.3** | **Planificar proyecto** | **20 h** | | **27/02/17** | **06/03/17** |
| 1.3.1 | Definir historias de usuario, tareas y estimaciones | 8 | 1.2 | 27/02/17 | 28/02/17 |
| 1.3.2 | Documentar propuesta técnica | 12 | 1.2 | 01/03/17 | 02/03/17 |
| 1.3.3 | Entregar propuesta técnica | 0 | 1.3.2 | 06/03/17 | 06/03/17 |
| **2** | **Desarrollo técnico del proyecto** | **262 h** | | **06/03/17** | **08/03/17** |
| **2.1** | **Definir requisitos del proyecto** | **16 h** | | **06/03/17** | **08/03/17** |
| 2.1.1 | Crear diagrama de casos de uso | 8 | 1.2 | 06/03/17 | 07/03/17 |
| 2.1.2 | Definir y documentar requisitos de datos | 8 | | 08/03/17 | 08/03/17 |
| **2.2** | **Análisis** | **53 h** | | **08/03/17** | **27/03/17** |
| 2.2.1 | Crear diagrama de clases | 8 | 2.1 | 08/03/17 | 09/03/17 |
| 2.2.2 | Documentar clases | 16 | 2.2.1 | 09/03/17 | 14/03/17 |
| 2.2.3 | Diagrama de actividades | 24 | 2.1.1 | 14/03/17 | 17/03/17 |
| 2.2.4 | Validar análisis | 5 | 2.2.2 | 17/03/17 | 27/03/17 |
| **2.3** | **Diseño** | **18 h** | | | |
| 2.3.1 | Identificar y clasificar usuarios | 2 | 2.1.1 | 27/03/17 | 28/03/17 |
| 2.3.2 | Diseñar interfaces gráficas | 8 | | 28/03/17 | 29/03/17 |
| 2.3.3 | Refinar diagrama de clases de diseño | 5 | 2.2.1 | 29/03/17 | 29/03/17 |
| 2.3.4 | Validar diseño | 3 | 2.3.3 | 30/03/17 | 30/03/17 |
| **2.4** | **Desarrollar proyecto** | **171 h** | | **30/03/17** | **09/05/17** |
| 2.4.1 | Programación | 155 | | 30/03/17 | 04/05/17 |
| 2.4.2 | Pruebas | 16 | | 05/05/17 | 09/05/17 |
| **2.5** | **Puesta en marcha** | **4 h** | | **10/05/17** | **12/05/17** |
| 2.5.1 | Implantación | 3 | 2.4 | 10/05/17 | 10/05/17 |
| 2.5.2 | Formación | 1 | 2.5.1 | 10/05/17 | 10/05/17 |
| 2.5.3 | Entrega final | 0 | 2.5.1 | 12/05/17 | 12/05/17 |

| **3** | **Documentación y presentación del TFG** | **138 h** | | **15/05/17** | **12/06/17** |
|---|---|---|---|---|---|
| 3.1 | Redacción de informes quincenales | 10 | | 06/03/17 | 12/05/17 |
| 3.2 | Redacción de la memoria técnica | 100 | | 15/05/17 | 01/06/17 |
| 3.3 | Entrega de la memoria técnica | 0 | 3.2 | 02/06/17 | 02/06/17 |
| 3.4 | Preparación de la presentación oral | 27 | 3.3 | 02/06/17 | 08/06/17 |
| 3.5 | Presentación oral | 1 | 3.4 | 26/06/17 | 26/06/17 |

*Tabla 1. Planificación inicial del proyecto*

**Diagrama de Gantt**

El objetivo de un diagrama de Gantt es exponer de manera gráfica la planificación de las tareas en el transcurso del tiempo y los recursos asignados a cada tarea. Muestra de forma visual si las tareas se solapan en el tiempo o se realizan secuencialmente.

En la figura 4 se muestra el diagrama de Gantt de la planificación de tareas al inicio del proyecto, teniendo en cuenta el horario mencionado anteriormente y los días festivos. También se puede observar la asignación de los recursos humanos a cada tarea, siendo en este proyecto exclusivamente el estudiante en prácticas.

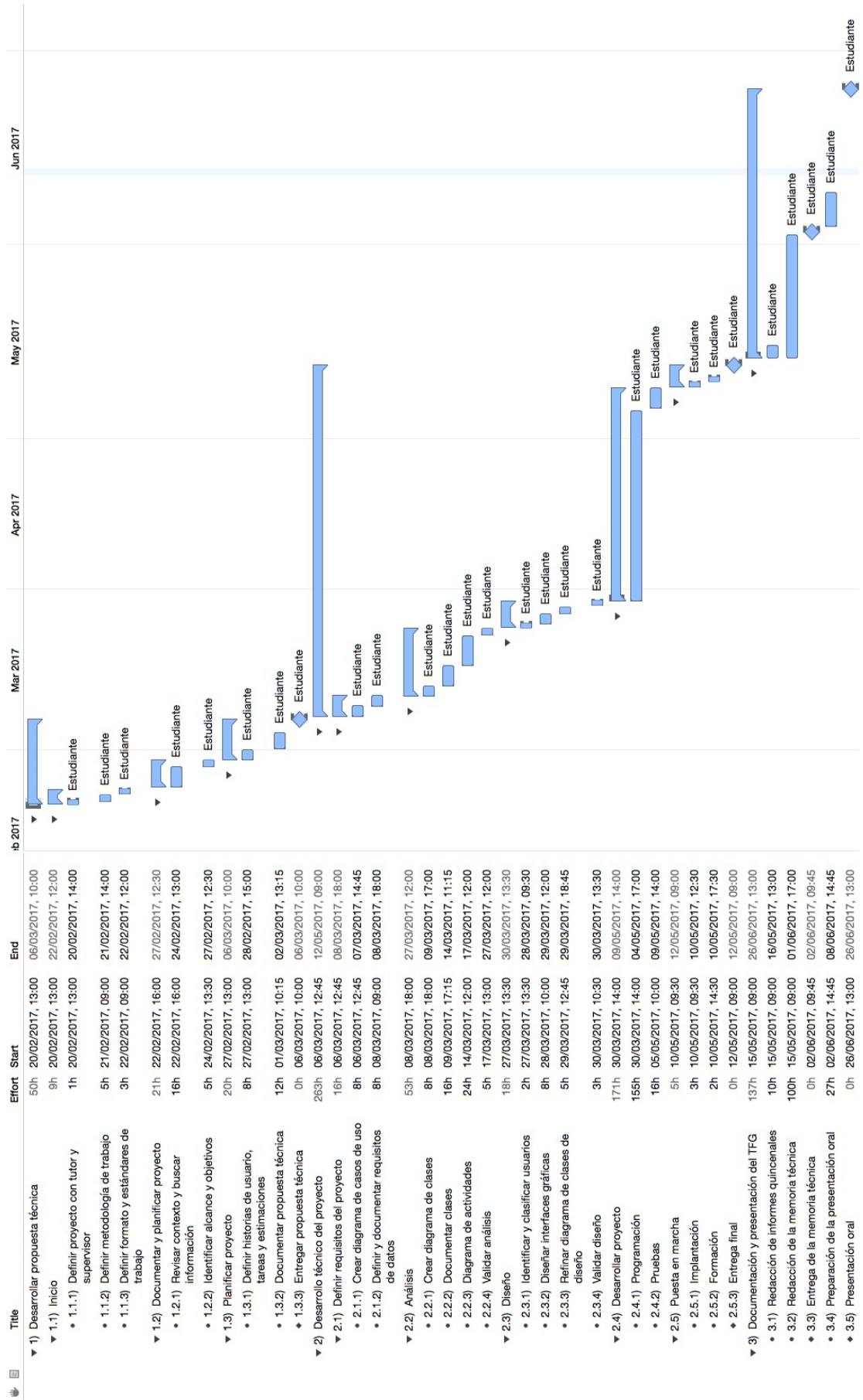



*Figura 4. Diagrama de planificación inicial de tareas (Gantt)*

**Historias de usuario**

Siguiendo la metodología Scrum, el trabajo a desarrollar se ha desglosado en historias de usuario según los requisitos de la aplicación. Una historia de usuario es una representación de un requisito escrito en una o dos frases utilizando un lenguaje común. La estimación del desarrollo de cada historia de usuario se puntúa, normalmente siguiendo una serie, en este caso la serie de Fibonacci. A su vez, las historias de usuario se desglosan en tareas que se estiman en horas.

Las historias de usuario se organizan siguiendo algún criterio, bien por dificultad, prioridad para el cliente, etc. y se agrupan en iteraciones, en el mundo de las metodologías ágiles más bien conocido como *sprints*. Un *sprint* es un bloque temporal corto (de dos semanas a un mes) en el que se implementan unas funcionalidades acordadas al inicio que se estiman se pueden finalizar dentro de la duración del mismo. El objetivo de un *sprint* es tener un producto entregable funcional al final del mismo. Es decir, en un *sprint* se deben incorporar nuevas funcionalidad a una aplicación de forma que al terminarlo el producto esté un poco más desarrollado pero ya sea funcional. Si se incluyen tareas que luego no hay tiempo de implementar se deben trasladar al siguiente *sprint*. Es importante que el plazo del *sprint* no se modifique.

La figura 6 muestra la tabla con las historias de usuario y su desglose en tareas con sus correspondientes estimaciones, así como el *sprint* al que se han asignado. En este desglose de tareas queda excluido el tiempo dedicado a aprender el lenguaje y a investigar sobre el contexto en el que se desarrolla el proyecto al inicio del mismo, que también requiere bastante tiempo cuando todo es desconocido, así como a escribir la memoria del proyecto.

El desarrollo del proyecto se ha dividido en 3 iteraciones de 4 semanas cada una. Al final de cada iteración se ha ido revisando el trabajo completado, el trabajo pendiente y la desviación con respecto a la planificación inicial (comentada en el apartado 3.4).

La estimación de horas es relativa dado que es la primera vez que se realiza un proyecto de este tipo con un lenguaje nuevo y un IDE nuevo y pueden surgir problemas imprevistos. En la tabla 2 a continuación se aprecia en qué iteración se incluyeron las tareas inicialmente y, debido a la desviación, en qué *sprint* se acabaron realizando.

| Historia de usuario / tarea | Est. | It. inicial | It. final |
|---|---|---|---|
| **Como Administrador tengo que poder gestionar bancos y cuentas bancarias con el fin de elegir a cuál se ingresarán los cobros** | 3 | 1 | 1 |
| Implementación del modelo de bancos y cuentas bancarias | 3h | 1 | 1 |
| Implementación del repositorio de bancos | 3h | 1 | 1 |
| Validación de datos de las cuentas bancarias | 5h | 1 | 1 |
| Crear capa de servicio para controlador de bancos | 4h | 1 | 1 |
| Controlador de bancos | 3h | 1 | 1 |

| | | | |
|---|---|---|---|
| Prototipos de las vistas | 3h | 1 | 1 |
| Vistas de bancos | 7h | 1 | 1 |
| Pruebas validación de crear, editar y eliminar bancos | 6h | 1 | 1 |
| Vistas de cuentas bancarias | 5h | 1 | 1 |
| Controlador de cuentas bancarias | 3h | 1 | 1 |
| Pruebas validación de crear, editar y eliminar cuentas | 6h | 1 | 1 |
| **Como usuario tengo que poder registrarme en la aplicación con el fin de gestionar remesas** | 5 | 1 | |
| Diseño de modelo de usuarios | 3h | 1 | 1 |
| Diseño de roles y permisos de usuario | 3h | 1 | 1 |
| Implementar patrón repositorio genérico | 2h | 1 | 1 |
| Diseño de prototipos de las vistas (registro, inicio de sesión | 2h | 1 | 1 |
| Implementar vista de registro de usuario | 5h | 1 | 1 |
| Implementar vista de inicio de sesión de usuario | 3h | 1 | 1 |
| Controlador de detalles de usuario | 4h | 1 | 1 |
| Vista de detalles de usuario | 5h | 1 | 1 |
| Implementar registro de usuarios manual (por administrador) | 4h | 1 | 1 |
| Controlador de registro de usuarios OAuth | 4h | 1 | 1 |
| Implementar OAuth 2.0 con Google | 3h | 1 | 1 |
| Implementar OAuth 2.0 con Facebook | 3h | 1 | 1 |
| Implementar OAuth con Twitter | 3h | 1 | 1 |
| Implementar OAuth con LinkedIn | 3h | 1 | 1 |
| Implementar OAuth con Microsoft | 3h | 1 | 1 |
| Ajustar visibilidad de menús según perfil | 2h | 1 | 1 |
| Crear capa de servicio para controlador de usuarios | 3h | 1 | 1 |
| Diseñar e implementar página de inicio | 5h | 1 | 3 |
| Pruebas validación de inicio de sesión de usuario | 8h | 1 | 3 |
| Pruebas validación de registro de usuario | 8h | 1 | 3 |
| Pruebas validación de edición de datos de usuario por sí mismo | 6h | 1 | 3 |

| | | | |
|---|---|---|---|
| **Como administrador tengo que poder gestionar usuarios con el fin de controlar el acceso a la aplicación** | 3 | 1 | |
| Diseño de prototipos de las vistas (gestión de usuarios por administrador) | 4h | 1 | 1 |
| Consultas de gestión de usuarios | 8h | 1 | 1 |
| Controlador de listado de usuarios para administrador | 8h | 1 | 1 |
| Vista de listado de usuarios para administrador | 8h | 1 | 1 |
| Test de edición de datos de usuario por administrador | 3h | 1 | 3 |
| Test cambio de rol de usuario por administrador | 3h | 1 | 3 |
| Activar y desactivar usuarios por AJAX | 2h | 1 | 1 |
| **Como usuario tengo que poder elegir las facturas que quiero incluir en una remesa con el fin de prepararla para el banco** | 8 | 2 | 2 |
| Análisis: crear diagrama de clases y documentar requisitos de datos | 8h | 2 | 2 |
| Diseño de modelo de remesas | 4h | 2 | 2 |
| Implementación de modelo remesas | 8h | 2 | 2 |
| Diseño de prototipo de bloque de selección de facturas | 4h | 2 | 2 |
| Controlador de consulta de facturas | 8h | 2 | 2 |
| Vista de selección de facturas para remesa | 8h | 2 | 2 |
| Filtros de selección de facturas | 8h | 2 | 2 |
| Controlador de creación de remesa | 8h | 2 | 2 |
| Vista de creación de remesa | 16h | 2 | 2 |
| Conversión de facturas seleccionadas a JSON | 4h | 2 | 2 |
| **Como usuario tengo que poder obtener el XML de una serie de facturas con el fin de enviarlo al banco para su cobro** | 5 | 2 | 2 |
| Web API: definición de estructura de API | 5h | 2 | 2 |
| Web API: definición de estructura XML | 8h | 2 | 2 |
| Web API: conversor JSON a XML | 12h | 3 | 2 |
| **Como usuario tengo que poder gestionar las remesas creadas** | 3 | 3 | |
| Diseño de prototipos de las vistas (listado, edición y borrado de remesa) | 2h | 3 | 2 |
| Controlador de listado de remesas | 8h | 3 | 2 |
| Vista de listado de remesas | 4h | 3 | 2 |

| | | | |
|---|---|---|---|
| Vista y controlador de edición de remesa | 7h | 3 | 2 |
| Vista y controlador de borrado de remesa | 3h | 3 | 2 |
| Test validación edición de remesa | 6h | 3 | 3 |
| Test validación borrado de remesa | 4h | 3 | 3 |

*Tabla 2. Desglose de historias de usuario y tareas*

Como se puede observar, la estimación inicial no fue exacta y hubo tareas que finalmente se trasladaron a otros *sprints* según las circunstancias y la prioridad de las tareas restantes, algunas a *sprints* posteriores y otras a *sprints* anteriores.

En cuanto a los entregables (conocidos como *deliverables* en inglés) durante el desarrollo del proyecto, la planificación prevé poder ofrecer una versión de pruebas al finalizar cada iteración con las nuevas funcionalidades introducidas. Aunque no se ha logrado cumplirlo exactamente, sí que se ha conseguido obtener una versión prácticamente definitiva con las principales funcionalidades previstas para cada iteración. Principalmente ha faltado la implementación de las pruebas unitarias dentro del plazo previsto para cada iteración, cosa que se ha realizado al final con el único fin de tener una aplicación funcional lo antes posible debido a algunas circunstancias explicadas más adelante.

**Repositorio y seguimiento del desarrollo**

Con el fin de realizar un seguimiento de las historias de usuario y sus tareas a medida que se desarrolla la aplicación, se precisaba utilizar una herramienta de gestión especializada en proyectos de software puesto que ofrecen funcionalidades adicionales que suelen ser muy útiles. En concreto lo que se buscaba es tener un repositorio, es decir, un espacio centralizado donde poder almacenar toda la información de la planificación y el seguimiento del desarrollo de la aplicación así como el código fuente. Hay muchos repositorios muy conocidos para la gestión de proyectos de software basados en metodologías ágiles y la gestión de cambios en el código cuando se trabaja en equipos con control de versiones de los ficheros.

El repositorio más conocido es GIT. Este software es muy sencillo de utilizar y muy flexible y fiable para realizar un control de versiones de código en grandes equipos sin problemas, permitiendo unificar cambios y publicar nuevas versiones de un producto de manera rápida, incluyendo el registro de cambios, autores y toda la información relacionada, fichero por fichero. Sin embargo, carece de un sistema de seguimiento de historias de usuario y tareas completadas profesional, de forma que se puedan asociar dichas tareas a cambios en el código del software. Con GIT cada operación *commit* (nuevo bloque de cambios) se entiende como una tarea finalizada. Normalmente para realizar dicho seguimiento se utiliza otro software de planificación de tareas como puede ser JIRA.

Aquí es donde entra en escena Team Foundation Server, o con su nuevo nombre, Visual Studio Team Services (VSTS). Esta herramienta de Microsoft incluye todas las funcionalidades en un sólo sistema y se integra perfectamente en Visual Studio, que como hemos mencionado anteriormente es el IDE que utilizaremos para desarrollar el proyecto. VSTS permite realizar un control de todos los cambios en los ficheros subidos al repositorio, pudiendo deshacer cambios en caso de introducir algún error, igual que con GIT, pero

además permite realizar toda la planificación del proyecto con metodologías ágiles desde el panel de control de la herramienta en su interfaz web. Esto incluye las historias de usuario, las tareas de cada historia con su estimación inicial, su coste real, la planificación de iteraciones, etc. El seguimiento de todos los aspectos del desarrollo desde un único sitio y su integración con Visual Studio son las ventajas por las que se ha elegido utilizar esta herramienta para este proyecto. Además de ser la herramienta que ya utiliza PDCnet para el desarrollo y mantenimiento de las otras aplicaciones que ofrecen.

Otra ventaja muy interesante de VSTS es que permite asociar un conjunto de cambios subido al repositorio (conocido como "*push*") con una tarea de la planificación. Así, con un sólo clic desde el panel de control se pueden ver los ficheros que contienen código asociado a cada tarea, lo que permite identificar errores y solucionarlos de forma más rápida y eficaz.

La figura 5 muestra una captura de pantalla de parte del *backlog* (listado de tareas a realizar) en la interfaz web de Visual Studio Team Services.

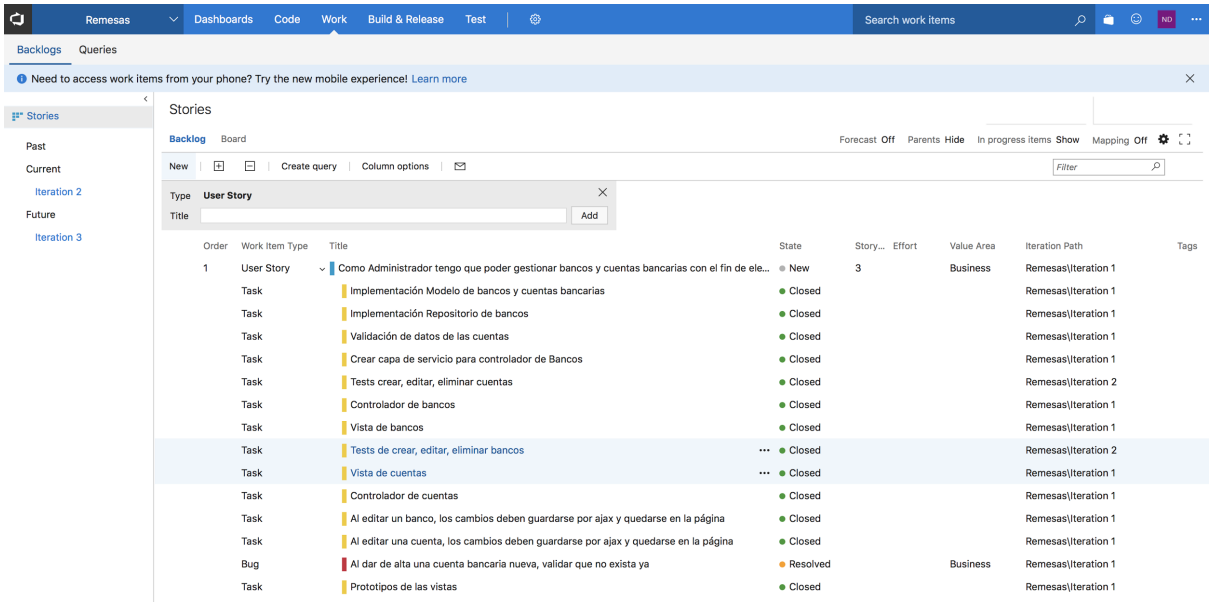



*Figura 5. Visual Studio Team Services como herramienta de planificación*

## 3.3 Estimación de recursos y costes del proyecto

Los recursos necesarios para el desarrollo de un proyecto de este tipo pueden clasificarse como personas, componentes software reutilizables y herramientas de hardware/software necesarios para realizar el proyecto.

- **Recursos humanos**. Este proyecto se ha planificado para ser desarrollado únicamente por un estudiante durante su estancia en prácticas de 300 horas. Además, lo realiza sin recibir una compensación económica por lo que los recursos humanos son siempre los mismos a lo largo de todo el desarrollo. Una vez lanzada la aplicación es necesario realizar el mantenimiento de la misma cuando surjan errores, haya que aplicar cambios de seguridad en el código, o mejoras para evitar nuevas formas de ataques o intrusiones. No obstante, la estimación de los recursos humanos

necesarios para el mantenimiento no supera las 4h mensuales y será responsabilidad del personal de la propia empresa para la que se ha desarrollado el proyecto.

- **Herramientas de hardware y software**. Debemos diferenciar entre las herramientas necesarias durante la fase de desarrollo y las necesarias cuando la aplicación ya se encuentra en producción.

    - Durante la fase de desarrollo se requiere como hardware un equipo informático donde el estudiante pueda desarrollar la aplicación, con un coste aproximado de 1.200€ contando con los periféricos imprescindibles.

    - En cuanto al software, es necesario disponer de Visual Studio 2015/2017, pero con la edición Community para estudiantes y desarrolladores *open-source* es suficiente, por lo que no supone ningún desembolso económico. Para optimizar el código también se desea utilizar el *plugin* para Visual Studio ReSharper (de Jetbrains, https://www.jetbrains.com/resharper/) que facilita mucho el mantenimiento y la organización del código, así como el desarrollo y verificación de pruebas con estadísticas detalladas de cobertura de código y mucho más que no ofrece el propio IDE. Igual que con Visual Studio, la licencia gratuita para estudiantes es suficiente. Para la ejecución de la aplicación durante el desarrollo en el equipo local se debe instalar también el Servidor IIS Express, que también es gratuito.

    - Una vez publicada la aplicación en Azure, los servicios de alojamiento de la aplicación y las bases de datos son los únicos recursos necesarios. Estos recursos requieren el pago de una cuota mensual, que varía en función de su uso. Si la aplicación tiene más volumen de peticiones de lo habitual, el servidor se escala y libera más recursos hasta que baja el volumen. Una de las ventajas de la computación en la nube en general y de Azure es la gran escalabilidad que tienen debido a las enormes granjas de servidores e infraestructuras prácticamente ilimitadas de las que disponen.

- **Componentes de software reutilizables**. Como se ha mencionado anteriormente, el desarrollo se parte de inicio, con lo cual no se utiliza ninguna implementación previa de ningún componente.

**Cálculo de los costes como si fuera un proyecto profesional**

Aunque el proyecto se realiza por el estudiante sin recibir una compensación económica de la empresa, se puede estimar el coste del desarrollo de la aplicación teniendo en cuenta las horas invertidas y el precio medio de un programador junior en la zona. El presupuesto se divide en dos partes: el coste del desarrollo y el coste del alojamiento y mantenimiento.

Sabiendo que la duración de la estancia en prácticas es de 300 horas y cobrando un precio medio de 20€ la hora como programador junior en Castellón, el coste del desarrollo serían 300 x 20 = 6.000€ + IVA (21%).

En cuanto al mantenimiento, según lo estimado anteriormente se deben dedicar 4 horas mensuales a la corrección de errores o mejoras de seguridad, con un coste de 80€.

El coste de los servicios de Azure varía en función de la carga que reciba la aplicación. Cada servicio está localizado en una región y en función de esto puede tener un nivel de precios distinto, aunque suele ser similar en todas las zonas. El nivel va de Básico a Premium y dentro de cada nivel hay "subniveles" (A,B…) que se diferencian básicamente en el SLA (*Service Level Agreement*) y el hardware que lo soporta.

Lo bueno de la nube es que se pagan solamente los recursos que se usan. Por tanto, durante el desarrollo se suele utilizar el nivel básico y luego en producción se va pasando de un nivel/subnivel a otro en función de los usuarios conectados. También se pueden configurar límites según rangos de horas, como por ejemplo limitar los servicios a sus mínimos durante la noche y dejar que se ajuste automáticamente ( o programar los aumentos también según las horas) por el día.

Para ejecutar la aplicación en Azure se necesitan 3 servicios: *App Service* par alojar la aplicación, *SQL Database* para alojar las aplicaciones y *Service Bus* para la implementación de una cola de balanceo de carga (explicada en el capítulo 4 de diseño del proyecto). Se estima que para una aplicación de este calibre que al principio tendrá pocos usuarios conectamos simultáneamente, es suficiente contratar el nivel básico, que conlleva los siguientes costes, obtenidos de la calculadora de Azure en https://azure.microsoft.com/es-es/pricing/calculator/:

- ❏ App Service: 47,06€ / mes. 1 núcleo, 1.75GB de RAM y 10GB de almacenamiento.
- ❏ SQL Database: 4,20€ / mes. Hasta 5 bases de datos de 2GB cada una.
- ❏ Service Bus: 0,04€/mes. I millón de mensajes con un límite de 100 conexiones asincrónicas.

En total suma 51,30€ mensuales. Con lo cual el mantenimiento tiene un coste de 131,30€ + IVA (21%) mensuales. En el caso de tener que añadir funcionalidades nuevas se presupuestarán aparte.

### 3.4 Seguimiento del proyecto

A lo largo del desarrollo del proyecto, normalmente al menos una vez a la semana, se ha realizado una revisión del estado de la implementación con el *product owner*, en este caso el supervisor en la empresa PDCnet. En dicha revisión se comprobaba conjuntamente los cambios introducidos con respecto a los requisitos de la aplicación y se verificaba que fueran correctos, anotando las modificaciones que hubiera que realizar, en su caso.

Al final de cada *sprint* se ha realizado la revisión del *sprint*, comparando la estimación inicial de las tareas incluidas en el *sprint* y el resultado obtenido. Como se puede apreciar en la figura 6 que muestra el *burndown* (representación gráfica del trabajo por hacer en un proyecto dentro de un periodo de tiempo) de la primera iteración, la estimación no ha sido del todo correcta, requiriendo un reajuste de la planificación.

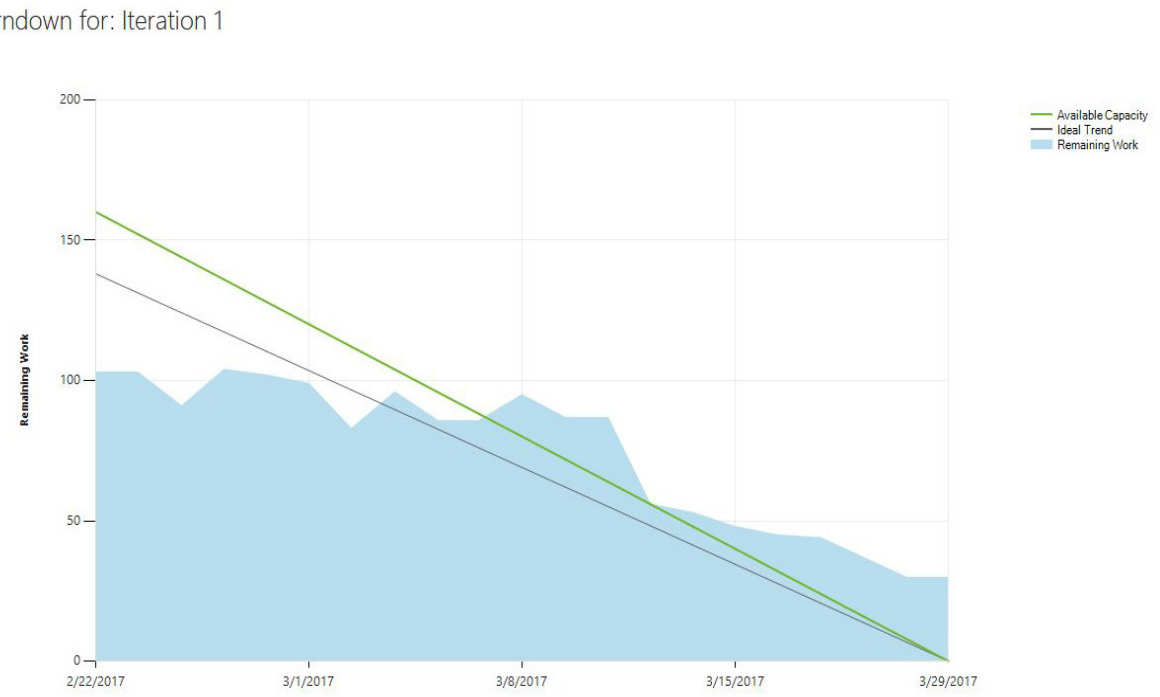



*Figura 6. Burndown del sprint 1*

Este retraso se debió principalmente al hecho de que en las revisiones semanales durante la primera iteración el supervisor de la empresa exigió modificar el código ya implementado en varias ocasiones para introducir nuevos patrones de diseño (explicados en el capítulo 5 de Implementación y Validación). Cuando ya parecía que una funcionalidad estaba implementada según los requisitos iniciales, se exigía reestructurar y modificar el código para introducir un nuevo patrón de diseño o cambios en la interacción del usuario con las interfaces del sistema. En alguna ocasión esto significaba también reescribir las pruebas unitarias de validación de dicho código, con el consecuente retraso de las demás tareas. Por este motivo, la validación se dejó para la última fase del desarrollo, para no perder tiempo realizando cambios tanto en la aplicación como en la validación sin saber si la empresa exigiría más cambios de arquitectura de la aplicación. No obstante, los cambios introducidos siempre daban buenos resultados, puesto que la aplicación quedaba más limpia, estructurada y organizada.

La lección aprendida en la primera iteración fue añadir siempre más tiempo al estimado a las tareas para prever este tipo de cambios al realizar las revisiones con el cliente. Y esto es lo que se hizo, aumentar el tiempo de las tareas de la iteración 2, pero no se contó con otros factores. En el segundo *sprint* ya no hubo más modificaciones de código por la introducción de nuevas tecnologías, simplemente había que ir completando las tareas restantes. Además, había bastante código que se podía reutilizar para crear nuevos apartados (sobre todo las vistas), por lo que la estimación de horas de la mayoría de las tareas de la segunda iteración resultó ser excesiva. Incluso se pudo adelantar tareas de la tercera iteración y realizarlas en la segunda. Finalmente hubo algunas tareas que se asignaron al *sprint* 2 que no se pudieron completar y se acabaron en el *sprint* 3. La figura 7 muestra el *burndown* de la iteración 2.

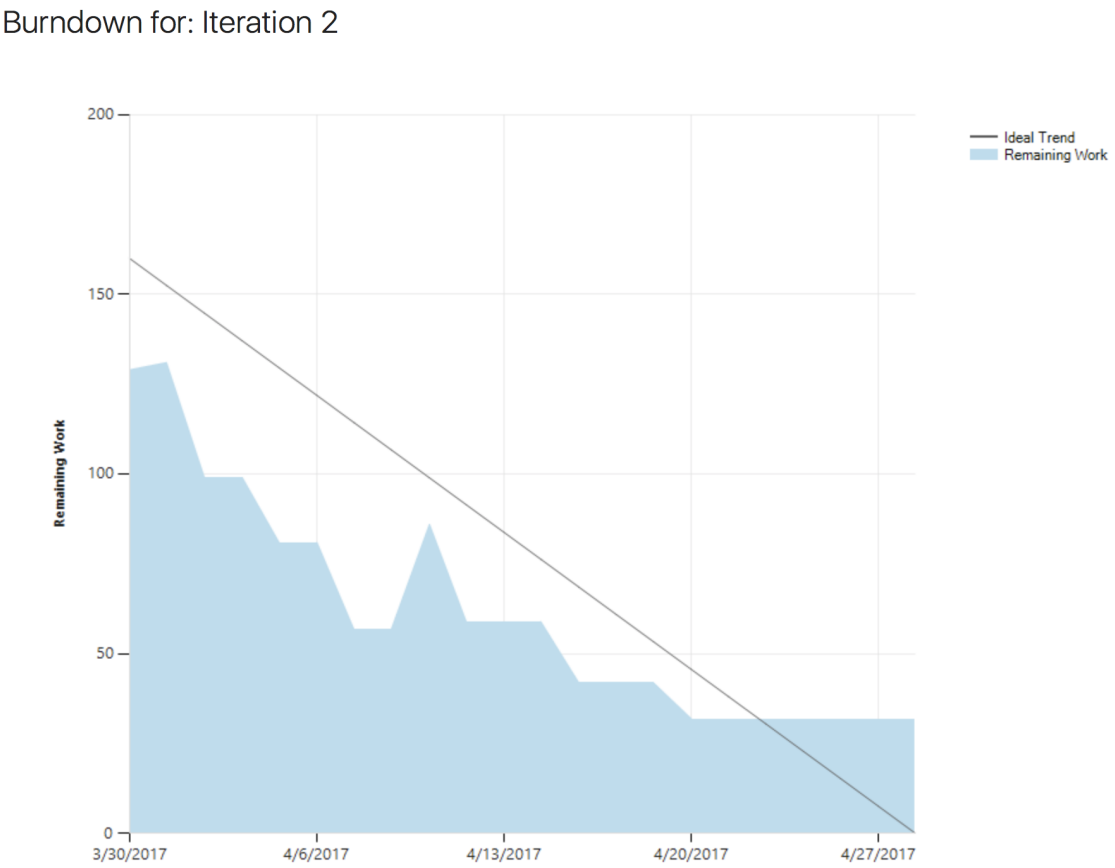
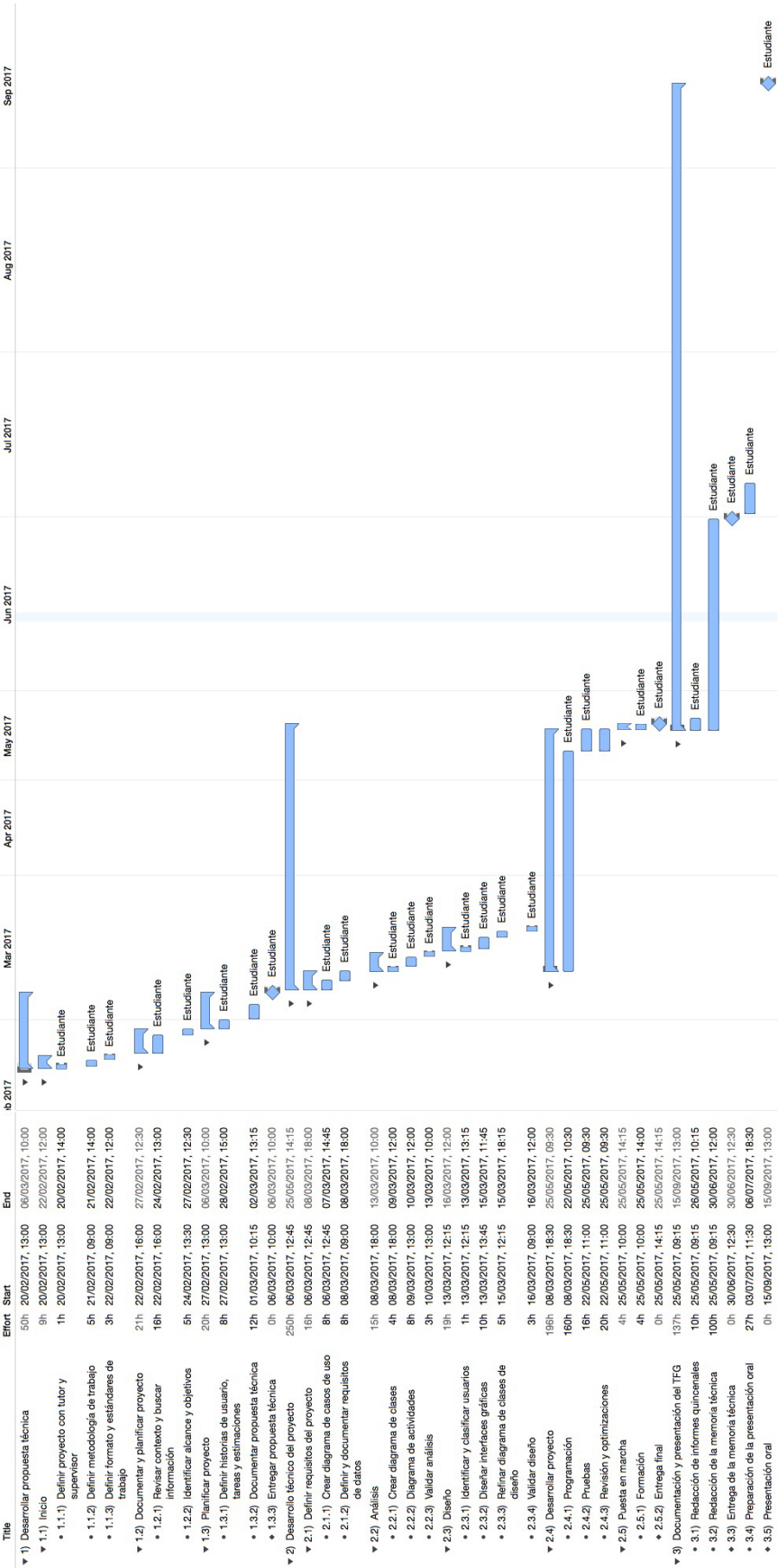



*Figura 7. Burndown del sprint 2*

Finalmente, en el sprint 3 se completaron los requisitos de la aplicación.

Por último, aunque la planificación inicial sólo incluía la implementación del esquema XML Core, al final del *sprint* 3 se ha implementado también el esquema XML B2B, a petición de la empresa.

Teniendo todos estos puntos en cuenta al finalizar la aplicación se ha actualizado el diagrama de Gantt inicial con los resultados del tiempo invertido en cada tarea. Cabe destacar que durante el desarrollo también se decidió que ciertas tareas de la planificación inicial no eran precisas durante la implementación (en concreto la especificación de las clases) y se podían realizar al redactar esta memoria, por lo que dichas tareas se han omitido del diagrama de Gantt final (ver figura 8), así como la puesta en marcha final dado que no se ha llegado a realizar. Por otro lado, ciertas tareas se desarrollaron concurrentemente (análisis e implementación). Finalmente, se decidió realizar la presentación del proyecto en la convocatoria de septiembre por falta de tiempo del estudiante para terminar la memoria antes, puesto que comenzó a trabajar a jornada completa al terminar las prácticas.




| Title | Effort | Start | End |
|---|---|---|---|
| ▼ 1) Desarrollar propuesta técnica | 50h | 20/02/2017, 13:00 | 06/03/2017, 10:00 |
| ▼ 1.1) Inicio | 9h | 20/02/2017, 13:00 | 22/02/2017, 12:00 |
| • 1.1.1) Definir proyecto con tutor y supervisor | 1h | 20/02/2017, 13:00 | 20/02/2017, 14:00 |
| • 1.1.2) Definir metodología de trabajo | 5h | 21/02/2017, 09:00 | 21/02/2017, 14:00 |
| • 1.1.3) Definir formato y estándares de trabajo | 3h | 22/02/2017, 09:00 | 22/02/2017, 12:00 |
| ▼ 1.2) Documentar y planificar proyecto | 21h | 22/02/2017, 16:00 | 27/02/2017, 12:30 |
| • 1.2.1) Revisar contexto y buscar información | 16h | 22/02/2017, 16:00 | 24/02/2017, 13:00 |
| • 1.2.2) Identificar alcance y objetivos | 5h | 24/02/2017, 13:30 | 27/02/2017, 12:30 |
| ▼ 1.3) Planificar proyecto | 20h | 27/02/2017, 13:00 | 06/03/2017, 10:00 |
| • 1.3.1) Definir historias de usuario, tareas y estimaciones | 8h | 27/02/2017, 13:00 | 28/02/2017, 15:00 |
| • 1.3.2) Documentar propuesta técnica | 12h | 01/03/2017, 10:15 | 02/03/2017, 13:15 |
| • 1.3.3) Entregar propuesta técnica | 0h | 06/03/2017, 10:00 | 06/03/2017, 10:00 |
| ▼ 2) Desarrollo técnico del proyecto | 250h | 06/03/2017, 12:45 | 25/05/2017, 14:15 |
| ▼ 2.1) Definir requisitos del proyecto | 16h | 06/03/2017, 12:45 | 08/03/2017, 18:00 |
| • 2.1.1) Crear diagrama de casos de uso | 8h | 06/03/2017, 12:45 | 07/03/2017, 14:45 |
| • 2.1.2) Definir y documentar requisitos de datos | 8h | 08/03/2017, 09:00 | 08/03/2017, 18:00 |
| ▼ 2.2) Análisis | 15h | 08/03/2017, 18:00 | 13/03/2017, 10:00 |
| • 2.2.1) Crear diagrama de clases | 4h | 08/03/2017, 18:00 | 09/03/2017, 12:00 |
| • 2.2.2) Diagrama de actividades | 8h | 09/03/2017, 13:00 | 10/03/2017, 12:00 |
| • 2.2.3) Validar análisis | 3h | 10/03/2017, 13:00 | 13/03/2017, 10:00 |
| ▼ 2.3) Diseño | 19h | 13/03/2017, 12:15 | 16/03/2017, 12:00 |
| • 2.3.1) Identificar y clasificar usuarios | 1h | 13/03/2017, 12:15 | 13/03/2017, 13:15 |
| • 2.3.2) Diseñar interfaces gráficas | 10h | 13/03/2017, 13:45 | 15/03/2017, 11:45 |
| • 2.3.3) Refinar diagrama de clases de diseño | 5h | 15/03/2017, 12:15 | 15/03/2017, 18:15 |
| • 2.3.4) Validar diseño | 3h | 16/03/2017, 09:00 | 16/03/2017, 12:00 |
| ▼ 2.4) Desarrollar proyecto | 196h | 08/03/2017, 18:30 | 25/05/2017, 09:30 |
| • 2.4.1) Programación | 160h | 08/03/2017, 18:30 | 22/05/2017, 10:30 |
| • 2.4.2) Pruebas | 16h | 22/05/2017, 11:00 | 25/05/2017, 09:30 |
| • 2.4.3) Revisión y optimizaciones | 20h | 22/05/2017, 11:00 | 25/05/2017, 09:30 |
| ▼ 2.5) Puesta en marcha | 4h | 25/05/2017, 10:00 | 25/05/2017, 14:15 |
| • 2.5.1) Formación | 4h | 25/05/2017, 10:00 | 25/05/2017, 14:00 |
| • 2.5.2) Entrega final | 0h | 25/05/2017, 14:15 | 25/05/2017, 14:15 |
| ▼ 3) Documentación y presentación del TFG | 137h | 25/05/2017, 09:15 | 15/09/2017, 13:00 |
| • 3.1) Redacción de informes quincenales | 10h | 25/05/2017, 09:15 | 26/05/2017, 10:15 |
| • 3.2) Redacción de la memoria técnica | 100h | 25/05/2017, 09:15 | 30/06/2017, 12:00 |
| • 3.3) Entrega de la memoria técnica | 0h | 30/06/2017, 12:30 | 30/06/2017, 12:30 |
| • 3.4) Preparación de la presentación oral | 27h | 03/07/2017, 11:30 | 06/07/2017, 18:30 |
| • 3.5) Presentación oral | 0h | 15/09/2017, 13:00 | 15/09/2017, 13:00 |

*Figura 8. Diagrama de Gantt con los plazos reales*

# Capítulo 4. Análisis y diseño del sistema

Este capítulo trata sobre el estudio de los requisitos en cuanto a funcionalidades del sistema y el diseño de la implementación de los mismos en sus distintas capas. Las técnicas de análisis que se han utilizado son las mismas que se han estudiado en la titulación, que son los diagramas de casos de uso, diagramas de actividades, requisitos de datos y tecnológicos, diagramas de clases y capas de la aplicación y diagramas de navegación para la presentación.

## 4.1 Análisis del sistema

**Diagrama de casos de uso**

Un diagrama de casos de uso representa las funcionalidades que debe tener un sistema de software asociadas a los actores que se ven afectados por ellas, bien porque deben poder aprovecharse de ellas o porque se ven envueltos en alguna fase de su ejecución. Estos actores pueden ser humanos, entidades externas o equipos como servidores u otros productos de software de los que depende la funcionalidad. En la figura 9 se muestra el diagrama de casos de uso de este proyecto.

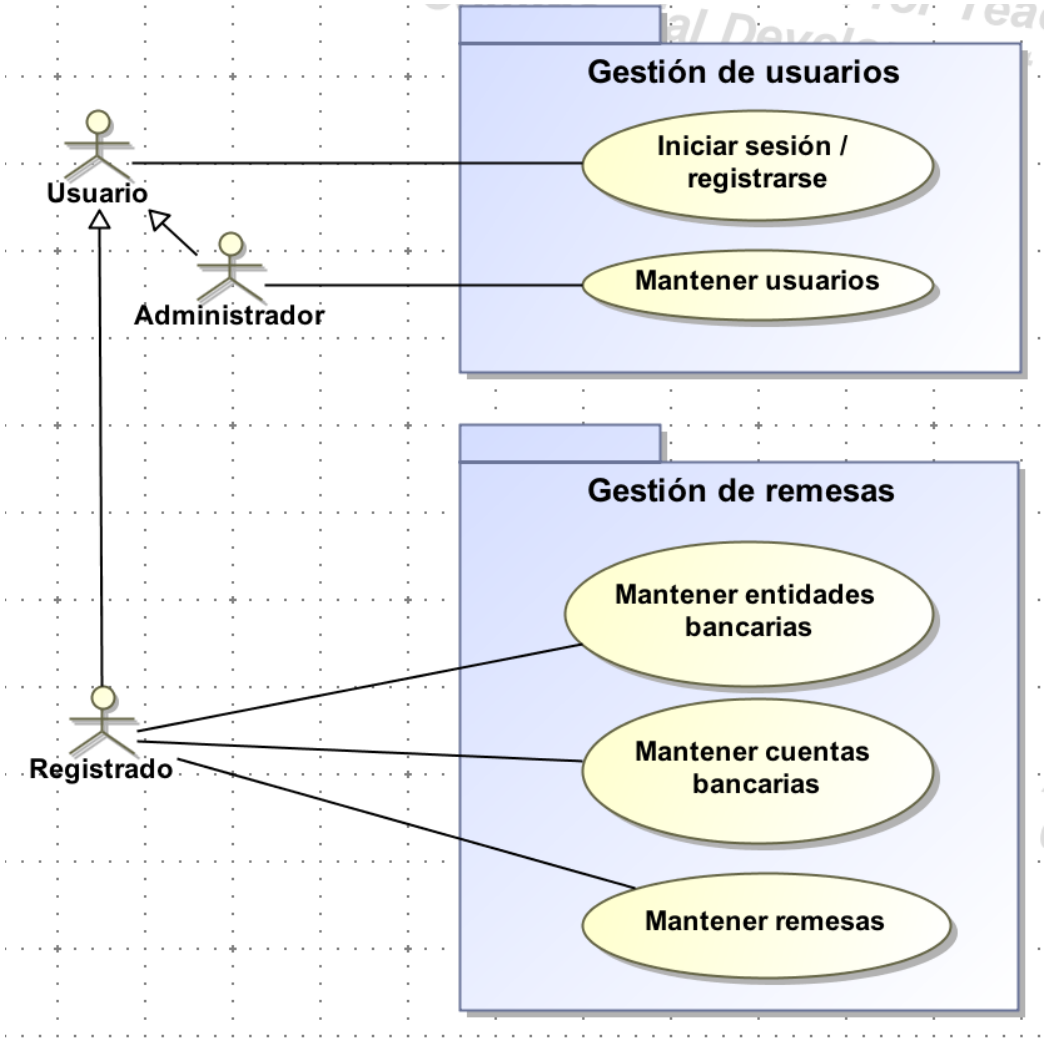



*Figura 9. Diagrama de casos de uso*

**Especificación de casos de uso**

La especificación de un caso de uso describe con detalle la funcionalidad esperada que engloba dicho caso de uso, así como las acciones y los pasos detallados que un usuario deberá realizar para utilizar tal funcionalidad. El detalle en la especificación de un caso de uso facilita su posterior implementación y evita posibles malentendidos con el cliente en la fase de especificación de requisitos.

A continuación se presenta la especificación de cada caso de uso del proyecto.

| **Especificación de caso de uso** | |
|---|---|
| **Identificador** | CU01 |
| **Nombre** | Iniciar sesión / registrarse |
| **Versión** | 1.0 |
| **Autores** | Yoshiya Magaña Martinez |
| **Fuentes** | PDCnet S.L. |
| **Descripción** | El sistema permitirá a un usuario registrarse en la plataforma web introduciendo sus datos personales manualmente o eligiendo utilizar su cuenta en una plataforma externa como Facebook, Twitter o LinkedIn. Una vez registrado también podrá asociar cuentas externas a su cuenta para iniciar sesión con dichas cuentas más rápidamente. También podrá modificar sus datos personales. |
| **Alcance** | Desde la introducción de sus datos en el sistema hasta poder iniciar sesión con éxito. |
| **Nivel** | Tarea principal. |
| **Actor principal** | Usuario |
| **Actores secundarios** | N/A. |
| **Relaciones** | N/A. |
| **Precondición** | N/A. |
| **Condición final con éxito** | **Iniciar sesión:** el usuario accederá a la aplicación y podrá realizar las operaciones asignadas a su rol de usuario.<br>Modificar usuario: los datos del usuario se actualizarán correctamente.<br>**Registro:** El sistema almacenará los datos del usuario para que pueda iniciar sesión. .<br>**Modificar perfil:** el sistema guardará los cambios introducidos en el perfil de usuario. |
| **Condición final con fracaso** | **Iniciar sesión:** el usuario no podrá acceder a la aplicación.<br>**Registro:** el usuario no se creará en el sistema.<br>**Modificar perfil:** los cambios introducidos no se guardarán. |
| ***Trigger*** | Una persona quiere darse de alta o acceder a la aplicación con sus credenciales para utilizar el sistema. |

| Secuencia normal | Acción |
|---|---|
| **1** | INICIAR SESIÓN |
| **1.1** | El usuario introducirá sus claves de acceso a la aplicación e iniciará sesión. |
| **1.2** | El usuario elegirá iniciar sesión con una cuenta externa de las plataformas disponible y, tras iniciar sesión en la plataforma seleccionada, accederá a la aplicación. |
| **2** | REGISTRARSE |
| **2.1** | El usuario introducirá sus datos personales de forma manual. |
| **2.2** | El usuario elegirá registrarse utilizando una cuenta externa y tras iniciar sesión en la plataforma externa confirmará los datos de registro en la aplicación. |
| **3** | El sistema recibirá los datos introducidos y los guardará para que el usuario pueda iniciar sesión. |
| **4** | MODIFICAR PERFIL |
| **5** | El sistema solicitará las credenciales al usuario. |
| **6** | El usuario iniciará sesión y accederá a los datos de su cuenta. |
| **7** | El usuario modificará sus datos. |
| **8** | El sistema guardará los cambios efectuados. |
| **Excepción <Paso 1.1>** | **Excepción** |
| **1** | Las credenciales introducidas no son válidas. |
| **Excepciones <Paso 3>** | **Excepción** |
| **1** | El sistema reconoce que ya hay un usuario registrado con los mismos datos y devuelve un error. |
| **Frecuencia esperada** | 3/mes |
| **Importancia** | Muy importante. |
| **Prioridad** | Corto plazo. |
| **Comentarios** | En el apartado de modificación de datos del perfil se incluye asociar o desasociar cuentas externas que tenga el usuario en otras plataformas. |

*Tabla 3. Especificación de caso de uso "Iniciar sesión / registrarse"*

| Especificación de caso de uso | |
|---|---|
| **Identificador** | CU02 |
| **Nombre** | Mantener usuarios |
| **Versión** | 1.0 |
| **Autores** | Yoshiya Magaña Martinez |

| | |
|---|---|
| **Fuentes** | PDCnet S.L. |
| **Descripción** | El sistema permitirá añadir, modificar, bloquear/desbloquear el acceso y eliminar usuarios manualmente por el administrador de la aplicación. |
| **Alcance** | Desde el alta de un usuario en el sistema hasta que se elimine por alguna razón. |
| **Nivel** | Tarea principal. |
| **Actor principal** | Administrador |
| **Actores secundarios** | N/A. |
| **Relaciones** | N/A. |
| **Precondición** | N/A. |
| **Condición final con éxito** | **Añadir usuario:** el usuario se añadirá al sistema permitiéndole iniciar sesión y utilizar la aplicación con el rol elegido por el administrador.<br>**Modificar usuario**: los datos del usuario se actualizarán correctamente.<br>**Bloquear usuario:** el usuario cambiará a estado "inactivo" y no tendrá acceso de inicio de sesión a la aplicación, pero sus datos permanecerán en el sistema.<br>**Desbloquear usuario:** el usuario cambiará a estado "activo" y tendrá nuevamente acceso al sistema.<br>**Eliminar usuario:** el usuario se eliminará del sistema con todos sus datos. |
| **Condición final con fracaso** | **Añadir usuario:** el usuario no se creará en el sistema.<br>**Modificar usuario:** el usuario no se actualizará en el sistema.<br>**Bloquear usuario:** el estado del usuario no cambiará.<br>**Desbloquear usuario:** el estado del usuario no cambiará.<br>**Eliminar usuario:** el usuario no se eliminará del sistema. |
| ***Trigger*** | El administrador solicita añadir, modificar, bloquear/desbloquear o eliminar un usuario desde el panel de control de la aplicación. |
| **Secuencia normal** | **Acción** |
| **1** | El sistema solicitará las credenciales de usuario administrador |
| **2** | El administrador introduce sus credenciales. |
| **3** | AÑADIR USUARIO |
| **3.1** | El administrador introducirá los datos del nuevo usuario. |
| **3.2** | El sistema almacenará los datos del usuario. |
| **4** | MODIFICAR USUARIO |
| **4.1** | El sistema listará todos los usuarios registrados. |
| **4.2** | El administrador elegirá el usuario que quiere modificar. |
| **4.3** | El administrador introducirá los nuevos datos del usuario. |
| **4.4** | El sistema almacenará los datos del usuario. |
| **5** | BLOQUEAR USUARIO |
| **5.1** | El sistema listará todos los usuarios registrados. |

| | |
|---|---|
| **5.2** | El administrador desactivará el usuario desde la lista de usuarios. |
| **5.3** | El sistema cambiará el estado del usuario bloqueando su acceso. |
| **6** | DESBLOQUEAR USUARIO |
| **6.1** | El sistema listará todos los usuarios registrados. |
| **6.2** | El administrador activará el usuario desde la lista de usuarios. |
| **6.3** | El sistema cambiará el estado del usuario restaurando su acceso. |
| **7** | BORRAR USUARIO |
| **7.1** | El sistema mostrará todos los usuarios almacenados. |
| **7.2** | El administrador indicará el usuario que desea eliminar. |
| **7.3.1** | El administrador confirmará la eliminación del usuario y el sistema borrará todos sus datos. |
| **7.3.2** | El administrador no confirmará la eliminación del usuario y todo se quedará igual, sus datos no se borrarán. |
| **Excepción <Paso 2>** | **Excepción** |
| **1** | Las credenciales introducidas no son válidas. |
| **Excepciones <Paso 3.2 y 4.4>** | **Excepción** |
| **1** | El administrador intenta añadir un usuario que ya está registrado en el sistema. |
| **Frecuencia esperada** | 3/mes |
| **Importancia** | Muy importante. |
| **Prioridad** | Corto plazo. |
| **Comentarios** | El alta y modificación de usuarios también lo podrán hacer los propios usuarios registrándose y modificando su perfil (solamente el suyo). Las acciones destacadas de este caso de uso son el bloqueo/desbloqueo y eliminación de usuarios para que el administrador del sistema pueda controlar el acceso a la aplicación. |

*Tabla 4. Especificación de caso de uso "Mantener usuarios"*

| **Especificación de caso de uso** | |
|---|---|
| **Identificador** | CU03 |
| **Nombre** | Mantener entidades bancarias |
| **Versión** | 1.0 |
| **Autores** | Yoshiya Magaña Martinez |
| **Fuentes** | PDCnet S.L. |
| **Descripción** | El sistema permitirá añadir, modificar, y eliminar entidades bancarias por usuarios registrados en la aplicación. |

| | |
|---|---|
| **Alcance** | Desde el alta de una entidad bancaria en el sistema hasta que se elimine por alguna razón. |
| **Nivel** | Tarea principal. |
| **Actor principal** | Usuario registrado |
| **Actores secundarios** | Usuario administrador |
| **Relaciones** | CU04: Mantener cuentas bancarias |
| **Precondición** | **Añadir banco:** el usuario deberá tener perfil de "registrado" para realizar la acción.<br>**Modificar banco:** el usuario deberá tener perfil de "registrado" para realizar la acción.<br>**Eliminar banco:** el usuario deberá tener perfil de "administrador" para realizar la acción. |
| **Condición final con éxito** | **Añadir banco:** el banco se añadirá al sistema.<br>**Modificar banco:** los datos del banco se actualizarán correctamente.<br>**Eliminar banco:** el banco se eliminará del sistema. |
| **Condición final con fracaso** | **Añadir banco:** el banco no se creará en el sistema.<br>**Modificar banco:** el banco no se actualizará en el sistema.<br>**Eliminar banco:** el banco no se eliminará del sistema. |
| ***Trigger*** | El usuario administrativo solicita añadir, modificar o eliminar una entidad bancaria desde el panel de control de la aplicación. |
| **Secuencia normal** | **Acción** |
| **1** | El sistema solicitará las credenciales del usuario. |
| **2** | El usuario introduce sus credenciales para iniciar sesión. |
| **3** | AÑADIR BANCO |
| **3.1** | El usuario introducirá los datos del nuevo banco. |
| **3.2** | El sistema almacenará los datos del banco. |
| **4** | MODIFICAR BANCO |
| **4.1** | El sistema listará todos los bancos almacenados. |
| **4.2** | El usuario elegirá el banco que desea modificar. |
| **4.3** | El usuario introducirá los nuevos datos de la entidad bancaria. |
| **4.4** | El sistema almacenará los datos del banco. |
| **5** | BORRAR BANCO |
| **5.1** | El sistema mostrará todos los bancos almacenados. |
| **5.2** | El usuario indicará el banco que desea eliminar. |
| **5.3.1** | El usuario confirmará la eliminación del banco y el sistema borrará todos sus datos. |
| **5.3.2** | El usuario no confirmará la eliminación del banco y todo se quedará igual, los datos del banco no se borrarán. |

| **Excepción <Paso 2>** | **Excepción** |
|---|---|
| **1** | Las credenciales introducidas no son válidas. |
| **Excepciones <Paso 3.1, 4.2 y 5.2>** | **Excepción** |
| **1** | El usuario no dispone del rol necesario para realizar la acción solicitada. |
| **Excepción <Paso 5.3.1>** | **Excepción** |
| **1** | El banco tiene cuentas bancarias asociadas. Primero se deberán eliminar dichas cuentas para poder eliminar el banco posteriormente. |
| **Frecuencia esperada** | Baja. Solamente al comenzar a utilizar la aplicación. |
| **Importancia** | Importante. |
| **Prioridad** | Corto plazo. |
| **Comentarios** | N/A |

*Tabla 5. Especificación de caso de uso "Mantener entidades bancarias"*

| **Especificación de caso de uso** | |
|---|---|
| **Identificador** | CU04 |
| **Nombre** | Mantener cuentas bancarias |
| **Versión** | 1.0 |
| **Autores** | Yoshiya Magaña Martinez |
| **Fuentes** | PDCnet S.L. |
| **Descripción** | El sistema permitirá añadir, modificar, y eliminar cuentas bancarias por usuarios registrados en la aplicación. |
| **Alcance** | Desde el alta de una cuenta bancaria en el sistema hasta que se elimine por alguna razón. |
| **Nivel** | Tarea principal. |
| **Actor principal** | Usuario registrado |
| **Actores secundarios** | Usuario administrador |
| **Relaciones** | CU03: Mantener entidades bancarias |
| **Precondición** | **Añadir cuenta:** el usuario deberá tener perfil de "registrado" para realizar la acción. La entidad bancaria a la que pertenece la cuenta que queremos dar de alta deberá estar ya creada previamente.<br>**Modificar cuenta:** el usuario deberá tener perfil de "registrado" para realizar la acción. Si queremos modificar la entidad bancaria de la cuenta, la nueva entidad deberá estar creada previamente.<br>**Eliminar cuenta:** el usuario deberá tener perfil de "administrador" para realizar la acción. |

| | |
|---|---|
| **Condición final con éxito** | **Añadir cuenta:** la cuenta bancaria se añadirá al sistema.<br>**Modificar cuenta:** los datos de la cuenta bancaria se actualizarán correctamente.<br>**Eliminar cuenta:** la cuenta bancaria se eliminará del sistema. |
| **Condición final con fracaso** | **Añadir cuenta:** la cuenta bancaria no se creará en el sistema.<br>**Modificar cuenta:** la cuenta bancaria no se actualizará en el sistema.<br>**Eliminar cuenta:** la cuenta bancaria no se eliminará del sistema. |
| ***Trigger*** | El usuario administrativo solicita añadir, modificar o eliminar una cuenta bancaria desde el panel de control de la aplicación. |
| **Secuencia normal** | **Acción** |
| **1** | El sistema solicitará las credenciales del usuario. |
| **2** | El usuario introduce sus credenciales para iniciar sesión. |
| **3** | AÑADIR CUENTA BANCARIA |
| **3.1** | El usuario introducirá los datos de la nueva cuenta bancaria. |
| **3.2** | El sistema almacenará los datos de la cuenta bancaria. |
| **4** | MODIFICAR CUENTA BANCARIA |
| **4.1** | El sistema listará todas las cuentas bancarias almacenadas. |
| **4.2** | El usuario elegirá la cuenta que desea modificar. |
| **4.3** | El usuario introducirá los nuevos datos de la cuenta bancaria. |
| **4.4** | El sistema almacenará los datos de la cuenta. |
| **5** | BORRAR CUENTA BANCARIA |
| **5.1** | El sistema mostrará todas las cuentas almacenadas. |
| **5.2** | El usuario indicará la cuenta que desea eliminar. |
| **5.3.1** | El usuario confirmará la eliminación de la cuenta y el sistema borrará todos sus datos. |
| **5.3.2** | El usuario no confirmará la eliminación de la cuenta y todo se quedará igual, los datos de la cuenta no se borrarán. |
| **Excepción <Paso 2>** | **Excepción** |
| **1** | Las credenciales introducidas no son válidas. |
| **Excepciones <Paso 3.1, 4.2 y 5.2>** | **Excepción** |
| **1** | El usuario no dispone del rol necesario para realizar la acción solicitada. |
| **Excepciones <Paso 3.2 y 4.4>** | **Excepción** |
| **1** | El usuario intenta introducir una cuenta bancaria que ya existe. |
| **Frecuencia esperada** | 2/mes |
| **Importancia** | Importante. |

| | |
|---|---|
| **Prioridad** | Medio plazo. |
| **Comentarios** | N/A |

*Tabla 6. Especificación de caso de uso "Mantener cuentas bancarias*

| **Especificación de caso de uso** | |
|---|---|
| **Identificador** | CU05 |
| **Nombre** | Mantener remesas |
| **Versión** | 1.0 |
| **Autores** | Yoshiya Magaña Martinez |
| **Fuentes** | PDCnet S.L. |
| **Descripción** | El sistema permitirá crear, modificar, descargar el XML de una remesa y eliminar remesas por usuarios registrados en la aplicación. |
| **Alcance** | Desde la creación de una remesa en el sistema hasta su eliminación por cualquier razón. |
| **Nivel** | Tarea principal. |
| **Actor principal** | Usuario registrado |
| **Actores secundarios** | Usuario administrador |
| **Relaciones** | CU04: Mantener cuentas bancarias |
| **Precondición** | **Añadir remesa:** el usuario deberá tener perfil de "registrado" para realizar la acción. La cuenta bancaria en la que queremos recibir el cobro de la remesa deberá estar ya creada previamente.<br>**Modificar remesa:** el usuario deberá tener perfil de "registrado" para realizar la acción. Si queremos modificar la cuenta bancaria de la cuenta en la que recibir el cobro, la nueva cuenta deberá estar creada previamente.<br>**Eliminar remesa:** el usuario deberá tener perfil de "administrador" para realizar la acción.<br>**Descargar XML:** la remesa para la que queremos descargar el XML deberá estar ya creada en el sistema. |
| **Condición final con éxito** | **Añadir remesa:** la remesa se guardará en el sistema.<br>**Modificar remesa:** los datos de la remesa se actualizarán correctamente.<br>**Eliminar remesa:** la remesa se eliminará del sistema.<br>**Descargar XML:** el sistema devolverá un fichero XML válido con los datos de la remesa seleccionada. |
| **Condición final con fracaso** | **Añadir remesa:** la remesa no se creará en el sistema.<br>**Modificar remesa:** la remesa no se actualizará en el sistema.<br>**Eliminar remesa:** la remesa no se eliminará del sistema.<br>**Descargar XML:** el fichero no se descargará. |
| ***Trigger*** | El usuario administrativo solicita crear, modificar, descargar el XML o eliminar una remesa desde el panel de control de la aplicación. |

| Secuencia normal | Acción |
|---|---|
| 1 | El sistema solicitará las credenciales del usuario. |
| 2 | El usuario introduce sus credenciales para iniciar sesión. |
| 3 | CREAR REMESA |
| 3.1 | El usuario introducirá los datos necesarios para la creación de la remesa. |
| 3.2 | El sistema almacenará los datos de la remesa. |
| 4 | MODIFICAR REMESA |
| 4.1 | El sistema listará todas las remesas creadas previamente. |
| 4.2 | El usuario elegirá la remesa que desea modificar. |
| 4.3 | El usuario modificará los datos que desee de la remesa. |
| 4.4 | El sistema guardará los cambios aplicados a la remesa. |
| 5 | BORRAR REMESA |
| 5.1 | El sistema mostrará el listado de remesas creadas anteriormente. |
| 5.2 | El usuario seleccionará la remesa que desea eliminar. |
| 5.3.1 | El usuario confirmará la eliminación de la remesa y el sistema borrará todos sus datos. |
| 5.3.2 | El usuario no confirmará la eliminación de la remesa y todo se quedará igual, los datos de la remesa no se borrarán. |
| 6 | DESCARGAR XML DE REMESA |
| 6.1 | El sistema mostrará todas las cuentas almacenadas. |
| 6.2 | El usuario seleccionará la remesa para la que quiere descargar el XML. |
| 6.3 | El fichero se descargará eligiendo el usuario dónde quiere almacenarlo en su equipo. |
| **Excepción <Paso 2>** | **Excepción** |
| 1 | Las credenciales introducidas no son válidas. |
| **Excepciones <Paso 3.1, 4.2 y 5.2>** | **Excepción** |
| 1 | El usuario no dispone del rol necesario para realizar la acción solicitada. |
| **Frecuencia esperada** | 5/mes |
| **Importancia** | Importante. |
| **Prioridad** | Medio plazo. Primero se deben implementar los otros casos de uso. |
| **Comentarios** | N/A |

*Tabla 7. Especificación de caso de uso "Mantener remesas"*

**Flujo de datos de creación de remesa**

Para comprender mejor cómo se relacionan los casos de uso entre sí y en qué orden se deben emplear con el fin de crear una remesa, se muestra a continuación la figura 10. Esta figura indica los pasos a seguir por un usuario de la aplicación desde el punto de vista de los casos de uso representando el flujo de los datos.

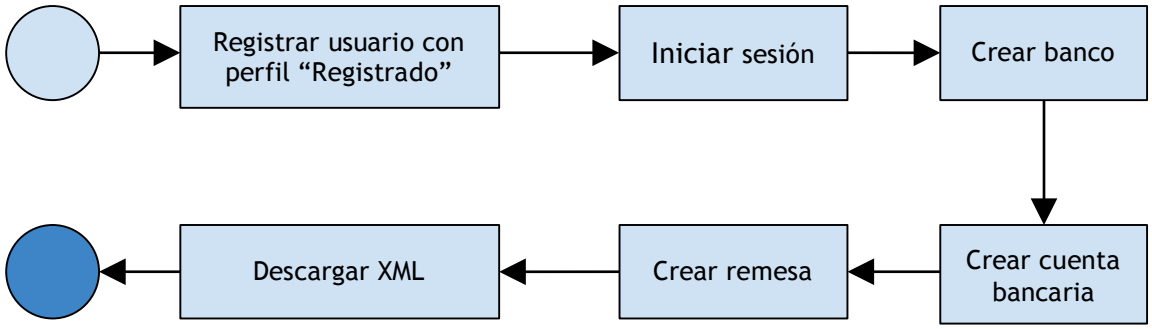



*Figura 10. Flujo de datos de creación de remesa partiendo de cero*

**Diagramas de actividades**

Un diagrama de actividades muestra un proceso de software como un flujo de trabajo a través de una serie de acciones. Las personas, los componentes del software o los equipos pueden realizar estas acciones. Es una representación sencilla y visual de los pasos que se realizan en un caso de uso. A continuación se muestran varios diagramas de actividades de algunos casos de uso del proyecto. No se incluyen todos puesto que son similares.

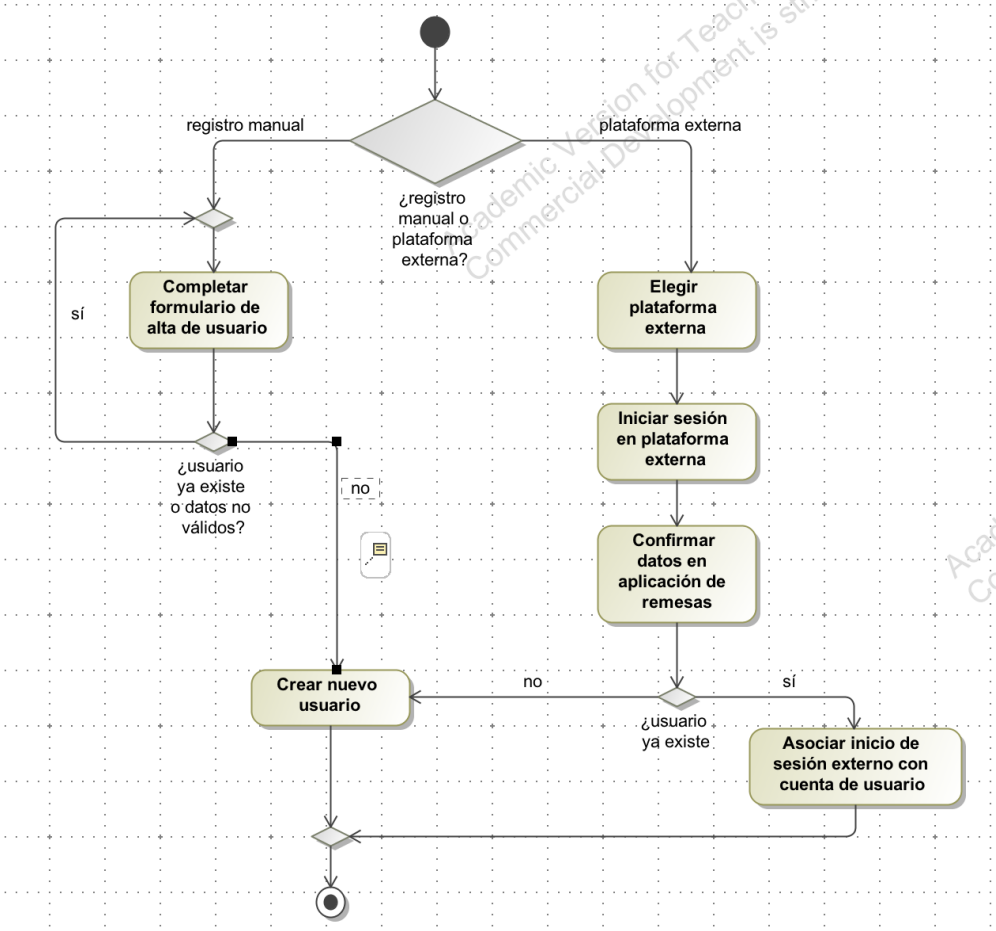



*Figura 11. Diagrama de actividades de la acción “Registro de usuario”*

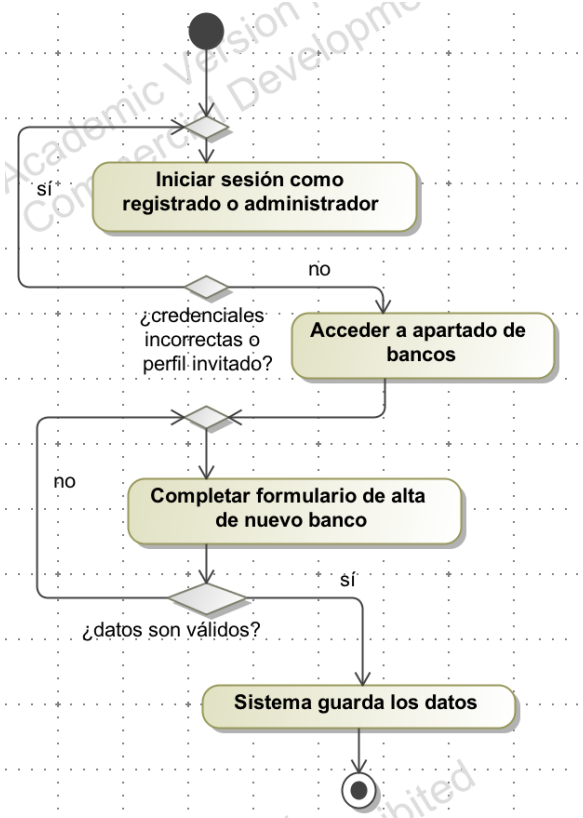



*Figura 12. Diagrama de actividades de la acción "Dar de alta un nuevo banco"*

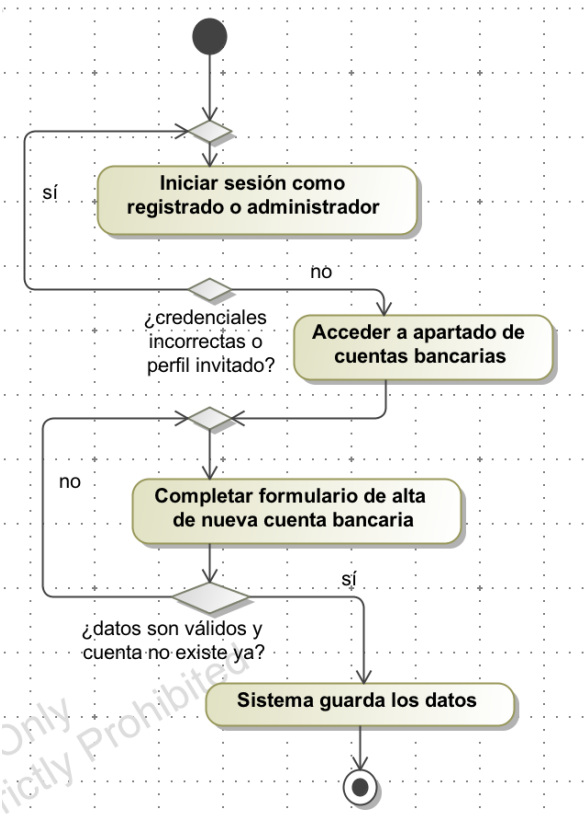



*Figura 13. Diagrama de actividades de la acción "Dar de alta nueva cuenta bancaria"*

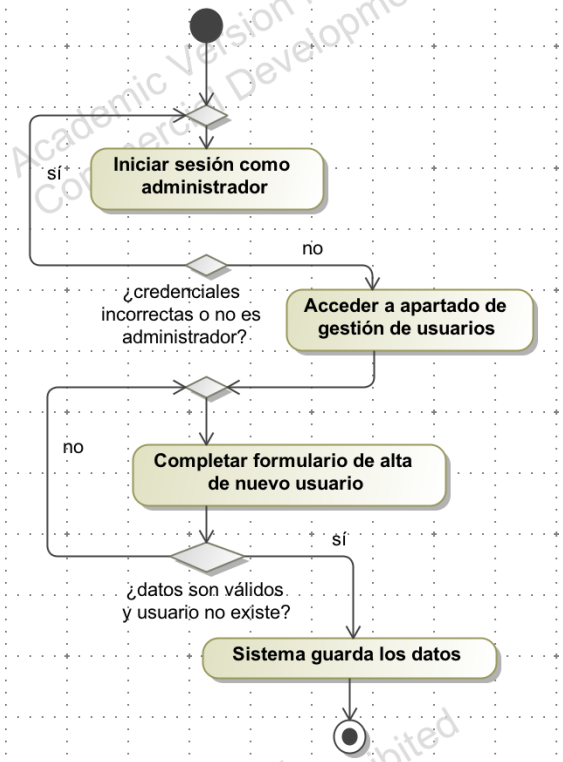



*Figura 14. Diagrama de actividades de la acción "Crear usuario con una cuenta de administrador"*

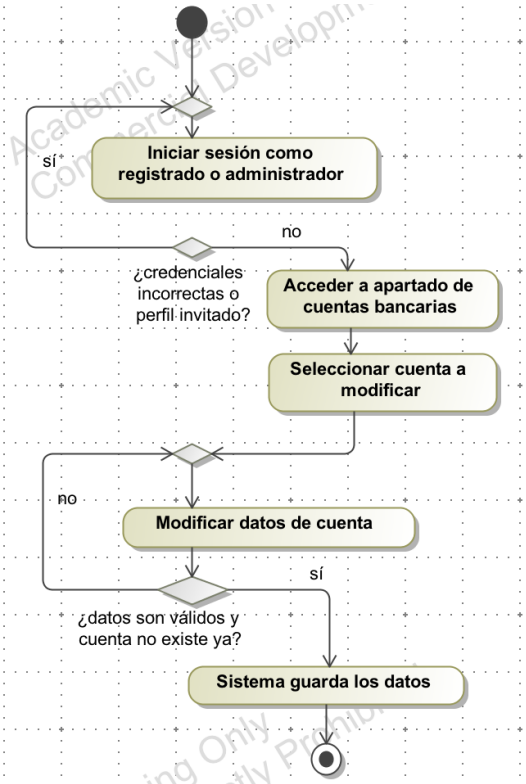



*Figura 15. Diagrama de actividades de la acción "Modificar datos de cuenta bancaria"*

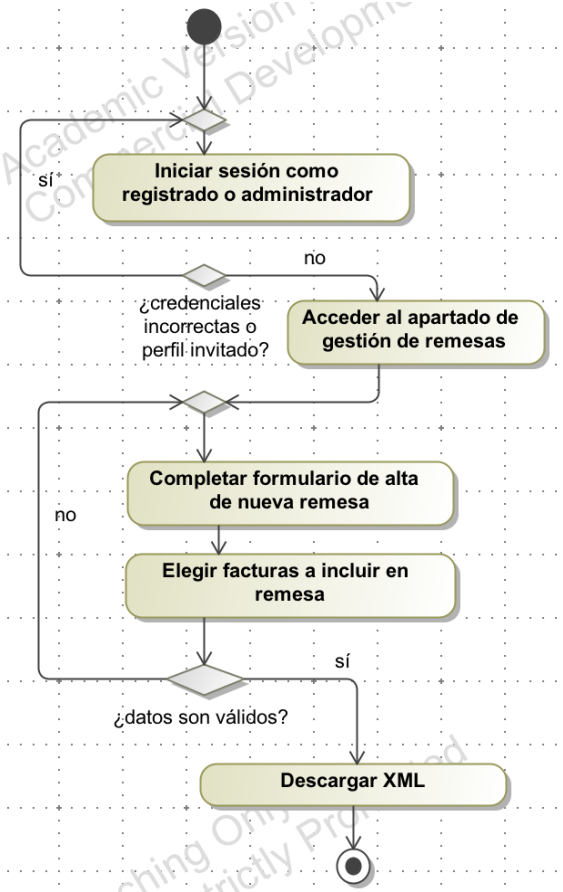



*Figura 16. Diagrama de actividades de la acción "Crear Remesa"*

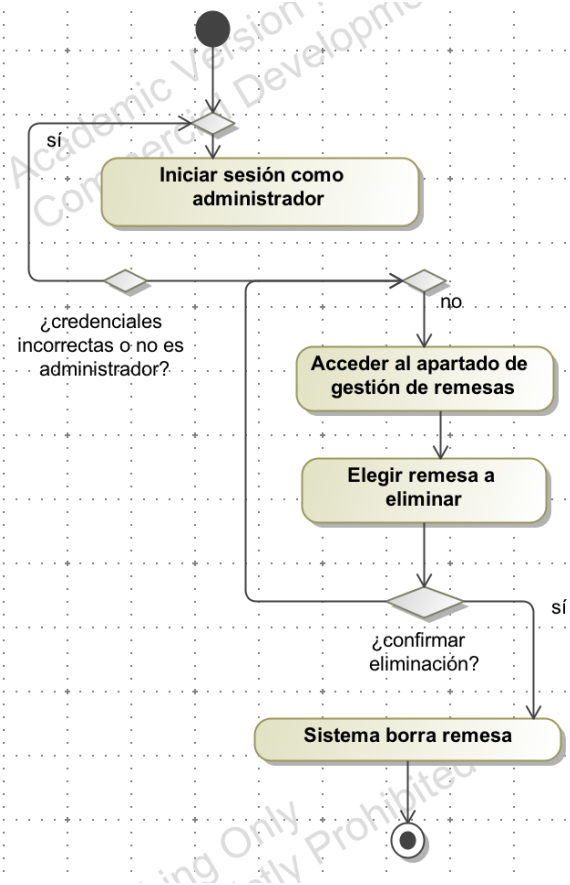



*Figura 17. Diagrama de actividades de la acción "Eliminar Remesa"*

**Requisitos de datos**

Las tablas de requisitos de datos contienen la información que se guarda de cada entidad del dominio de la aplicación. También describen cómo se relacionan las entidades entre sí mediante los requisitos asociados e incluyen la estimación aproximada de ocurrencias de cada dominio en el sistema, lo que se traduce en los registros de dicha entidad en la fuente de almacenamiento de datos.

A continuación se incluyen los requisitos de datos del sistema en castellano para una mejor comprensión, dado que el código de la aplicación se ha desarrollado en inglés (como se puede ver en el diagrama de clases, figura 19), por lo que en la aplicación real los nombres de las clases de dominio también están en inglés.

| Requisito de datos | |
|---|---|
| **Código** | RD-01 |
| **Nombre** | Usuario |
| **Versión** | 1.0 |
| **Autores** | Yoshiya Magaña Martinez |
| **Fuentes** | PDCnet Spain S.L. |

| | |
|---|---|
| **Requisitos asociados** | |
| **Datos específicos** | Nombre, apellidos, email, contraseña, nombre de usuario, fecha de registro, perfil |
| **Ocurrencias** | 1 |
| | 10 |
| **Importancia** | Alta |
| **Comentarios** | - El rol (perfil) podrá ser Invitado, Registrado o Administrador. Sólo podrá cambiarlo otro usuario con el rol de Administrador. |

*Tabla 8. Requisitos de datos de los usuarios*

| **Requisito de datos** | |
|---|---|
| **Código** | RD-02 |
| **Nombre** | Banco |
| **Versión** | 1.0 |
| **Autores** | Yoshiya Magaña Martinez |
| **Fuentes** | PDCnet Spain S.L. |
| **Requisitos asociados** | |
| **Datos específicos** | Nombre, código BIC |
| **Ocurrencias** | 1 |
| | 10 |
| **Importancia** | Alta |
| **Comentarios** | - El BIC es necesario para poder realizar las remesas. |

*Tabla 9. Requisitos de datos de los bancos*

| **Requisito de datos** | |
|---|---|
| **Código** | RD-03 |
| **Nombre** | Cuenta bancaria |
| **Versión** | 1.0 |
| **Autores** | Yoshiya Magaña Martinez |
| **Fuentes** | PDCnet Spain S.L. |
| **Requisitos asociados** | RD-02 |
| **Datos específicos** | IBAN |
| **Ocurrencias** | 1 |

| | |
|---|---|
| | 10 |
| **Importancia** | Alta |
| **Comentarios** | - El IBAN se utilizará en las remesas para recibir cobros. |

*Tabla 10. Requisitos de datos de las cuentas bancarias*

| **Requisito de datos** | |
|---|---|
| **Código** | RD-04 |
| **Nombre** | Parte iniciadora |
| **Versión** | 1.0 |
| **Autores** | Yoshiya Magaña Martinez |
| **Fuentes** | PDCnet Spain S.L. |
| **Requisitos asociados** | |
| **Datos específicos** | Nombre, nif |
| **Ocurrencias** | 1 |
| | 5 |
| **Importancia** | Alta |
| **Comentarios** | - Normalmente todas las remesas de una empresa siempre tendrán la misma parte iniciadora que es la propia empresa. No obstante, si cambiara de nombre fiscal tendrían que poder modificarla. Por esto se define como una entidad del dominio. Al crear una remesa, siempre se elegirá por defecto la última parte iniciadora utilizada (pudiendo el usuario modificar los datos manualmente). |

*Tabla 11. Requisitos de datos de la parte iniciadora de una remesa*

| **Requisito de datos** | |
|---|---|
| **Código** | RD-05 |
| **Nombre** | Remesa |
| **Versión** | 1.0 |
| **Autores** | Yoshiya Magaña Martinez |
| **Fuentes** | PDCnet Spain S.L. |
| **Requisitos asociados** | DR-04 |
| **Datos específicos** | Identificador del mensaje, fecha de creación, agrupar por cliente, identificador de pago, forma de pago, fecha de cobro, IBAN, BIC, nombre del banco, adeudos directos |
| **Ocurrencias** | 1 |
| | 10000 |

| | |
|---|---|
| **Importancia** | Alta |
| **Comentarios** | - **Identificador del mensaje**: El acreedor/presentador debe asegurarse que esta referencia es única para cada entidad destino del mensaje de presentación para un período de tiempo previamente acordado.<br>- **Fecha y hora de creación:** Formato: ISODateTime YYYY-MM-DDThh:mm:ss *(Ejemplo: 2010-11-05T09:25:15)*<br>- **Método de pago:** Solamente se admite el código "DD" (Direct Debit)<br>- **Fecha de cobro:** Formato**:** ISODate YYYY-MM-DD *(Ejemplo: 1990-03-08)* |

*Tabla 12. Requisitos de datos de las remesas*

| **Requisito de datos** | |
|---|---|
| **Código** | RD-07 |
| **Nombre** | Adeudo Directo |
| **Versión** | 1.0 |
| **Autores** | Yoshiya Magaña Martinez |
| **Fuentes** | PDCnet Spain S.L. |
| **Requisitos asociados** | RD-02, RD-03, RD-08 |
| **Datos específicos** | Identificador de instrucción, identificador de extremo a extremo, facturas |
| **Ocurrencias** | 1 |
| | 100000 |
| **Importancia** | Alta |
| **Comentarios** | Un adeudo directo comprende un bloque de facturas que se cobrarán a la misma cuenta. Ambos identificadores (identificador de instrucción e identificador de extremo a extremo) deben ser únicos dentro de la remesa.<br>La información de las facturas provendrá de la base de datos externa que contiene las facturas, con lo cual las clases de dominio serán auto-generadas. |

*Tabla 13. Requisitos de datos de los adeudos directos*

| **Requisito de datos** | |
|---|---|
| **Código** | RD-08 |
| **Nombre** | Factura |
| **Versión** | 1.0 |
| **Autores** | Yoshiya Magaña Martinez |
| **Fuentes** | PDCnet Spain S.L. |
| **Requisitos asociados** | RD-07 |
| **Datos específicos** | Número de factura, fecha de la factura, importe de la factura, nombre del cliente, NIF del cliente |
| **Ocurrencias** | 1 |

| | |
|---|---|
| | 100000 |
| **Importancia** | Alta |
| **Comentarios** | La información de las facturas provendrá de la base de datos externa que contiene las facturas, con lo cual esta clase de dominio será auto-generada. |

*Tabla 14. Requisitos de datos de las facturas*

## 4.2 Requisitos tecnológicos del proyecto y diseño de la arquitectura del sistema

Los requisitos tecnológicos del proyecto que indicó la empresa eran:

1. El lado servidor de la aplicación se debía programar en C# con ASP.NET.
2. Para el modelo y acceso a datos se debía utilizar el ORM Entity Framework y LINQ para las consultas.
3. El lado cliente se debía programar en HTML5 y CSS3 junto con Javascript.
4. Las interfaces web debían ser *responsive*, adaptándose al tamaño del dispositivo.
5. Se debían soportar los navegadores MS Edge, Mozilla Firefox, Google Chrome y Safari en sus últimas versiones.
6. La aplicación se debía desarrollar en el IDE Visual Studio en su versión más reciente.
7. Se debía implementar la autenticación de usuarios, pero sin exigir medidas de seguridad adicionales a las que están implementadas en una aplicación ASP.NET MVC por defecto.

**Arquitectura del proyecto**
La arquitectura de la aplicación se ha diseñado siguiendo el principio de la *separación de intereses*. La separación de intereses es un principio de diseño que busca separar un programa informático en secciones distintas, tal que cada sección enfoque un interés delimitado. Esto ofrece grandes ventajas como la fácil mantenibilidad del código, mayor flexibilidad y fácil extensión de las funcionalidades, así como mayor escalabilidad. Si está correctamente implementado también facilita la identificación y corrección de errores, así como la comprensión de la estructura y código del programa por desarrolladores nuevos o externos.

Un buen ejemplo de la separación de intereses es agrupar el código de las partes de una aplicación en distintas capas. Una arquitectura muy común es la que se basa en 3 capas: presentación, lógica de negocio y capa de acceso a datos. El patrón de diseño Modelo-Vista-Controlador que se ha utilizado en este proyecto y se detalla más adelante se basa en esta arquitectura. Otras arquitecturas incluyen *n* capas buscando una estructura aún más definida.

**Arquitectura del servidor**
En este apartado se debe diferenciar la arquitectura del servidor durante el desarrollo del proyecto y la arquitectura en producción una vez lanzada la aplicación.

Para el desarrollo, se instaló un servidor web IIS Express en el equipo local. Este servidor, al ser también de Microsoft, se integra perfectamente en Visual Studio. Al ejecutar la

aplicación en Visual Studio arranca el servidor y se puede acceder a ella mediante la URL *http://localhost:puerto* indicando el puerto especificado en la configuración del proyecto. Esto permite hacer *debugging* (depuración de código) de manera muy rápida durante el desarrollo. No obstante, las bases de datos a las que se conectaba el sistema no eran locales, residían en Azure, por lo que la aplicación se enlazaba con ellas remotamente.

Cabe recordar que un proyecto Web API, a pesar de no tener interfaz web, se instancia como una aplicación distinta y debe tener una URL distinta para que puedan realizar peticiones HTTP desde la aplicación principal. Para solucionar esto, se asigna un puerto distinto al proyecto Web API en la configuración del mismo y al ejecutar la solución arrancan ambos proyectos (Web MVC y Web API), cada uno en su respectivo puerto.

La arquitectura del servidor en producción es distinta, ya que todo debe residir en la nube para que pueda estar disponible para todos los usuarios desde cualquier parte del mundo. En este caso el servidor de aplicaciones es el servicio *Azure App Service*, y las bases de datos, igual que durante el desarrollo, se alojan con el servicio *SQL Database*. También está diseñado para utilizar una un servicio llamado *Azure Service Bus* implementando el patrón de diseño Balanceo de carga mediante una cola, que se explica en el apartado Patrones de Diseño. Aunque técnicamente se usan más servicios como cachés, la figura 18 muestra la arquitectura principal del proyecto en producción.

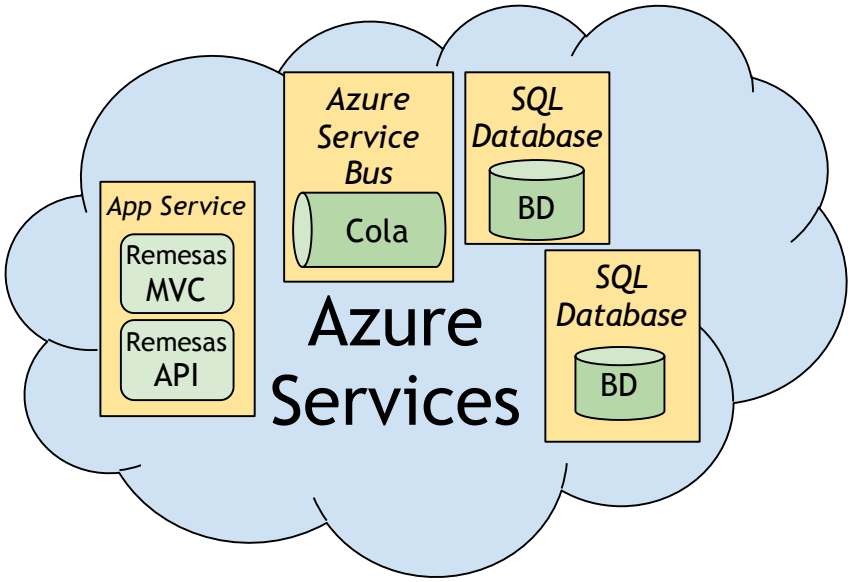



*Figura 18. Arquitectura del servidor en producción*

## 4.3 Clases del sistema y diseño funcional

**Diagrama de clases**

Un diagrama de clases es un tipo de diagrama de estructura estática en Lenguaje Unificado de Modelado (UML) que describe la estructura de un sistema mostrando las clases del sistema, sus atributos, operaciones (o métodos), y las relaciones entre los objetos. Es un diagrama vital en el apartado del diseño del sistema, puesto que es la base sobre la que se desarrolla la estructura del programa, y evitar errores en este diagrama ayuda a no perder tiempo realizando correcciones posteriormente.

Una aplicación de software tiene muchas clases distribuidas en cada una de las capas (presentación, lógica de negocio, dominio, etc.). El diagrama de clases más importante es el que modela los datos en su nivel más bajo y cómo se relacionan, es decir, la capa de dominio. La figura 19 muestra este diagrama de clases del proyecto.

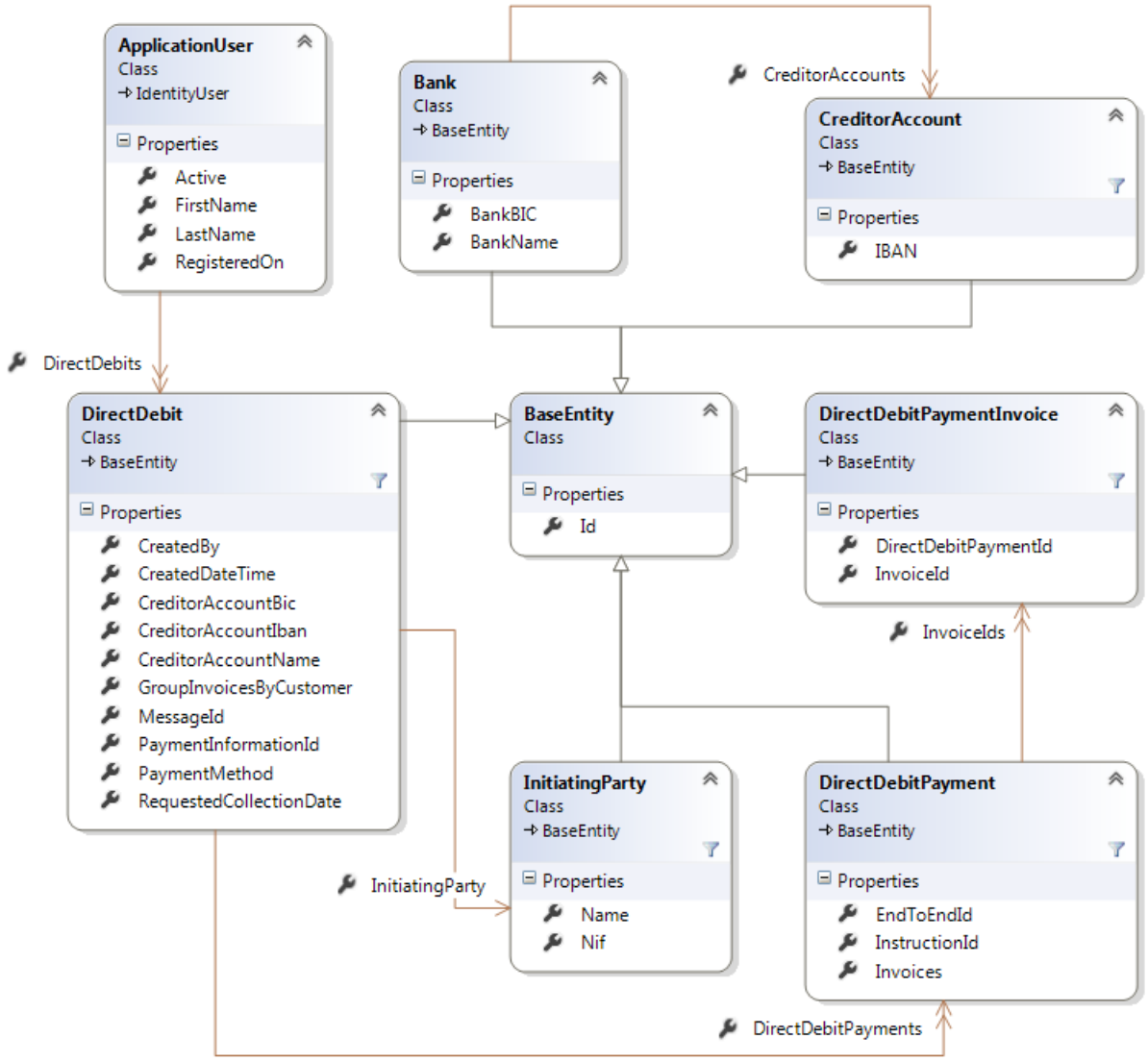



*Figura 19. Diagrama de clases generado por Visual Studio*

Se puede observar que la mayoría de las clases del dominio derivan de la clase *BaseEntity*, que únicamente contiene un atributo llamado *Id* que define el identificador de cada objeto. Esta estructura de herencia de clases nos permite no tener que duplicar este atributo en cada clase.

En cuanto a los usuarios, se puede observar que la clase *ApplicationUser* hereda de una clase llamada *IdentityUser*. Esta clase es la implementación de Entity Framework para el sistema de autenticación y autorización de usuarios de ASP.NET Identity, un paquete de Nuget que se utiliza para facilitar la gestión de usuarios y la autenticación mediante proveedores externos (más detalles en el apartado de detalles de implementación del capítulo 5). La clase *IdentityUser* contiene otros atributos importantes de los usuarios como

id, nombre de usuario y email. El diagrama de clases de Entity Framework para la gestión de usuarios de ASP.NET Identity se muestra en la figura 20.

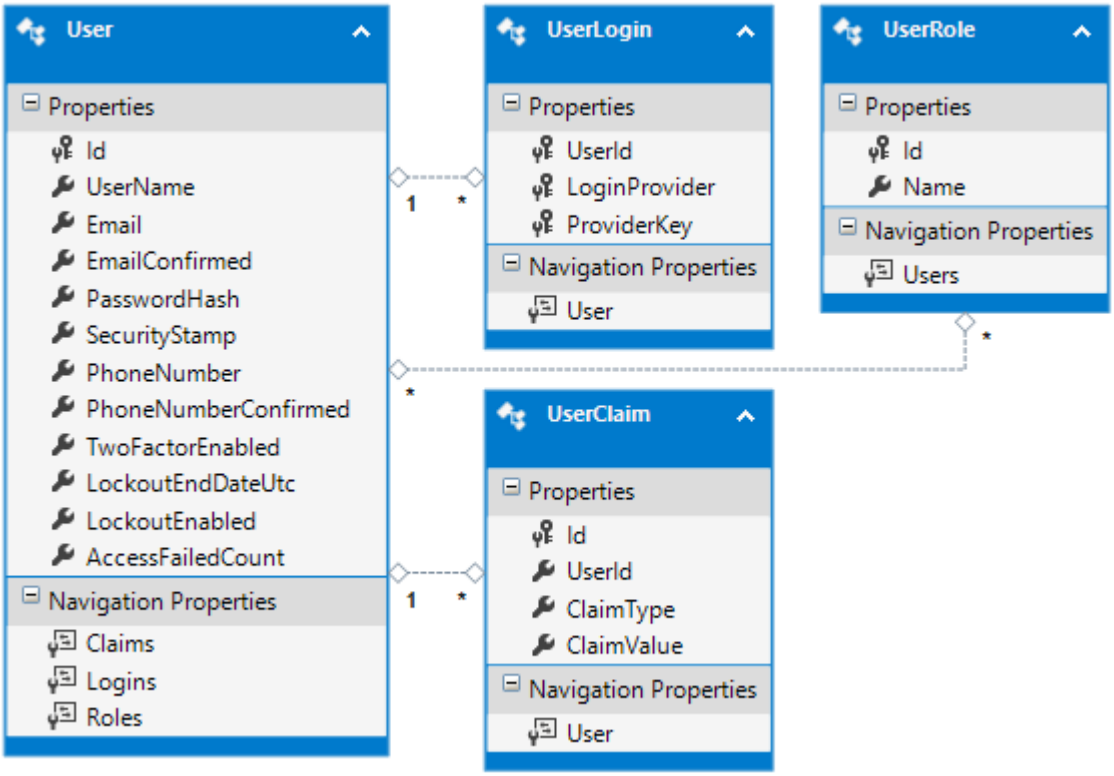



*Figura 20. Diagrama de clases de gestión de usuarios de Entity Framework [6]*

Las clases *UserClaim* y *UserLogin* se utilizan para la autenticación de usuarios mediante proveedores externos. Aunque el proceso de registro está explicado con mayor detalle en el apartado de detalles de implementación 5.1, en resumidas cuentas al crear una cuenta a través de una plataforma externa, ésta facilita información sobre el usuario en formato [clave => valor]. Esta información se almacena en la clase *UserClaim*, que a su vez es un atributo de la clase *User*. En la clase *UserLogin* se asocia el usuario con el listado de proveedores externos que ha registrado. La clase *UserRole* contiene los roles asignados al usuario. Todas estas clases y su lógica relacionada se podrían crear manualmente en el proyecto, pero utilizar ASP.NET Identity junto con Entiy Framework nos facilita el proceso de autenticación y registro ya que implementa toda la lógica como una librería de muy sencilla utilización.

Por otra parte, se han omitido del diagrama de clases aquellas que no están implementadas por la propia aplicación sino que dependen del diseño de terceras aplicaciones, en este caso la clase Factura y Cliente, ya que dependen del diseño de la aplicación que suministre los datos. En cualquier caso, los atributos que necesitamos de estas clases son el número de factura, la fecha de la factura, el importe total, el nombre del cliente y el NIF del cliente.

Un análisis del diseño más detallado incluye las funciones relacionadas con cada clase del dominio. En el listado de funciones se excluyen los *getters* y *setters* ya que se consideran imprescindibles para leer o modificar los atributos de cualquier objeto. La figura 21 muestra

el análisis funcional de la aplicación, incluyendo también las clases externas Factura y Cliente para visualizar los datos que se requieren de cada una de ellas para poder generar una remesa según los requisitos.

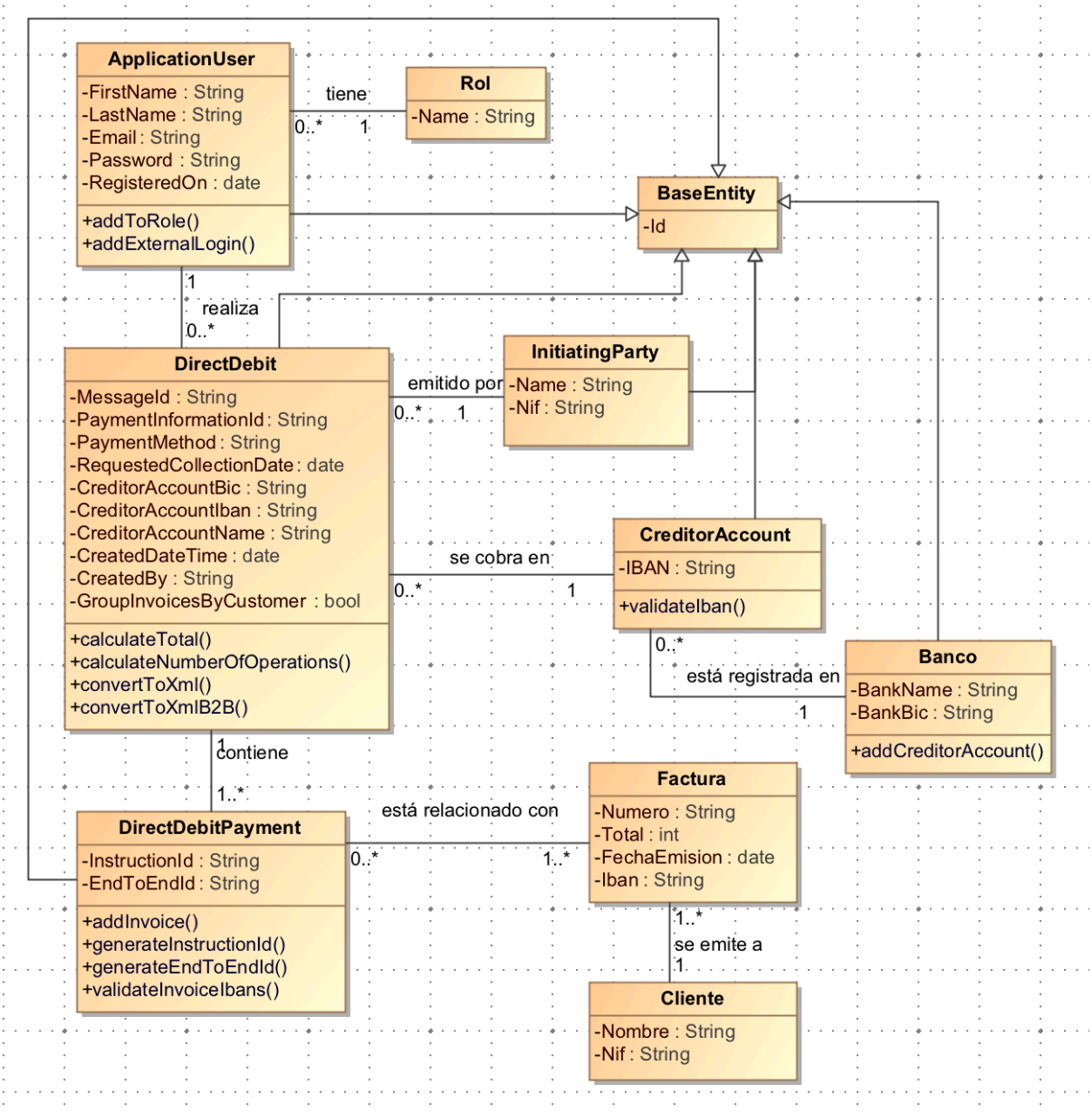



*Figura 21. Diagrama de clases refinado con funciones*

## 4.4 Diseño de la capa de presentación

La empresa no indicó ningún requisito en cuanto al diseño de la presentación de las interfaces o ventanas de la aplicación. Solamente que se debía utilizar Bootstrap como *framework responsive*. Por lo tanto, la elección de los colores y tanto la posición como estilos de los elementos quedaba a decisión del estudiante.

Para el diseño de las interfaces se ha intentado seguir lo más posible las 8 reglas de oro para el diseño de interfaces estudiadas en la carrera:

- ❏ Consistencia.
- ❏ Adaptar el diseño al nivel de conocimiento de los usuarios.
- ❏ Retroalimentación informativa.
- ❏ Diálogo. Mostrar textos de diálogo al cerrar procesos.
- ❏ Prevenir errores.
- ❏ Permitir deshacer cambios fácilmente.
- ❏ Fomentar la sensación de control.
- ❏ Reducir la carga de memoria a corto plazo.

El objetivo de esta guía de usabilidad es que la interacción del usuario con el sistema sea sencilla, intuitiva, cómoda y que su uso no requiera una carga de memoria importante. El sistema debe guiar al usuario siendo el usuario quien tenga el control de lo que pasa en cada momento.

**Colores y fuentes**

Para crear un diseño agradable a la vista con un texto legible se ha elegido una fuente muy común que es sencilla, bonita y no requiere licencia para su uso. La fuente se llama Roboto y se puede cargar en la aplicación directamente de Google Fonts. La figura 22 muestra las variantes de la fuente, así como el número de veces que la API de Google Fonts la ha servido a día 23/06/2017 y su distribución por países, superando las 35 mil millones de peticiones en las más de 12 millones de páginas web que la utilizan.

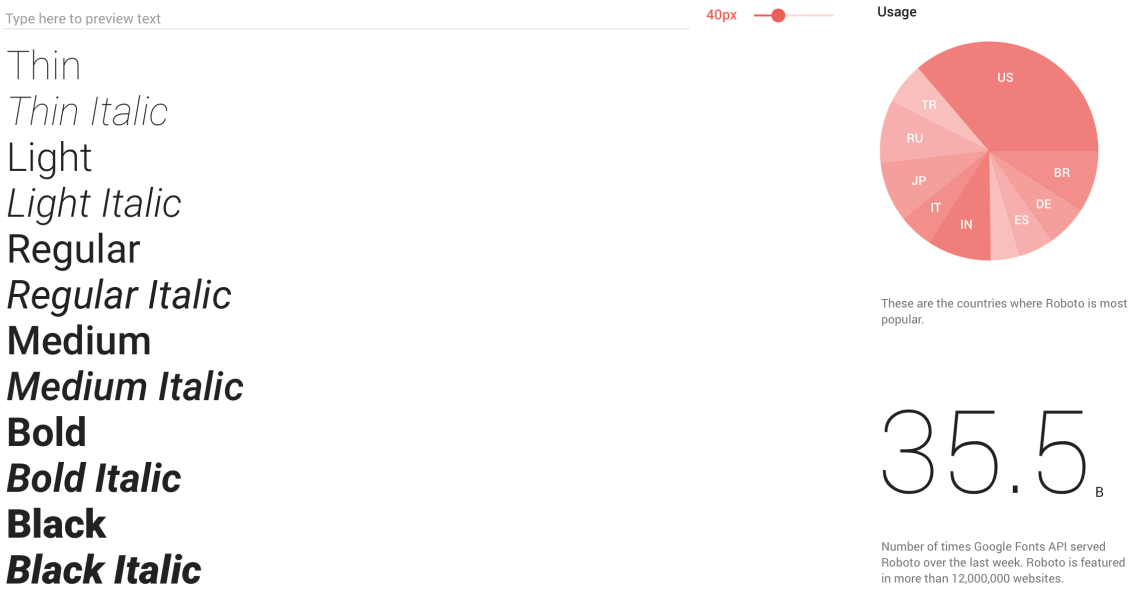



*Figura 22. Estilos de la fuente Roboto y sus estadísticas en Google Fonts [7]*

La elección de colores también quedaba libre para el estudiante. Aunque la variedad de colores que se podrían haber utilizado es extremadamente amplia, se han elegido 3 colores para toda la aplicación, dentro de una gama de colores pastel. Los dos colores principales para encabezados y menús tienen una coherencia estética entre sí. Por último el color rojo es representativo de mensajes de error. Cuando un usuario ve algo rojo lo primero que piensa es que algo en el sistema va mal o que ha cometido algún error, es un patrón de

usabilidad común compartido entre todo tipo de aplicaciones. La figura 23 muestra los colores así como los componentes que los utilizan.

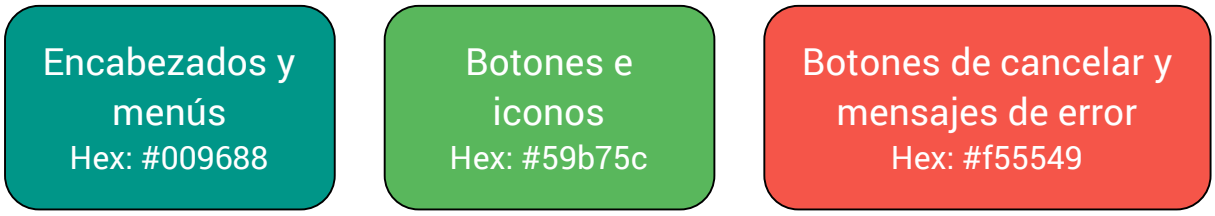



*Figura 23. Colores utilizados para las interfaces GUI*

**Diagrama de navegación**

Un diagrama de navegación muestra cómo se estructuran las secciones o contenidos de una aplicación web con respecto a los enlaces entre las vistas. Permite visualizar de forma gráfica qué ruta se debe seguir para llegar a una vista en concreto. La figura 24 muestra el diagrama de navegación del proyecto. Como se puede observar, es muy sencillo puesto que cada entidad (banco, cuenta bancaria, usuario, remesa) tiene las mismas operaciones. Y para cada operación se necesita una vista.

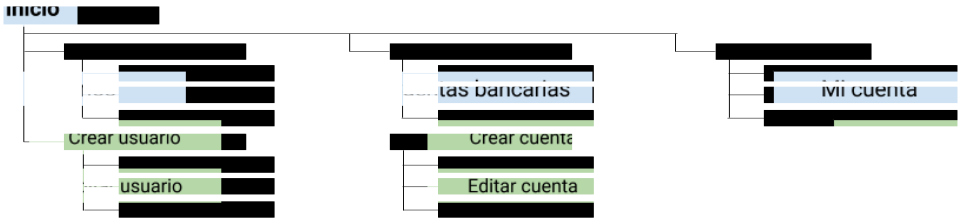



*Figura 24. Diagrama de navegación*

**Proceso de diseño centrado en el usuario**

En general el diseño está basado en las ocho reglas de oro del diseño de interfaces. Se ha conseguido consistencia entre todas las páginas de la web, tanto en sus estilos como en la forma en que se controlan. El objetivo es conseguir que la mayoría de usuarios de la aplicación sean capaces de entender y utilizar la aplicación rápidamente. Para ello, se genera *feedback* informativo ante las acciones del usuario y se le avisa debidamente del cumplimiento de tareas. Se previenen errores limitando las posibles entradas de datos por parte del usuario (en lugar de escribir una fecha debe seleccionarla de un calendario desplegable, por ejemplo) y realizando validación de los campos antes de enviar los datos. Finalmente, se evita que el usuario tenga que recordar datos entre pantallas, mostrando en la pantalla en la que se encuentra cualquier dato que pueda necesitar de pantallas anteriores.

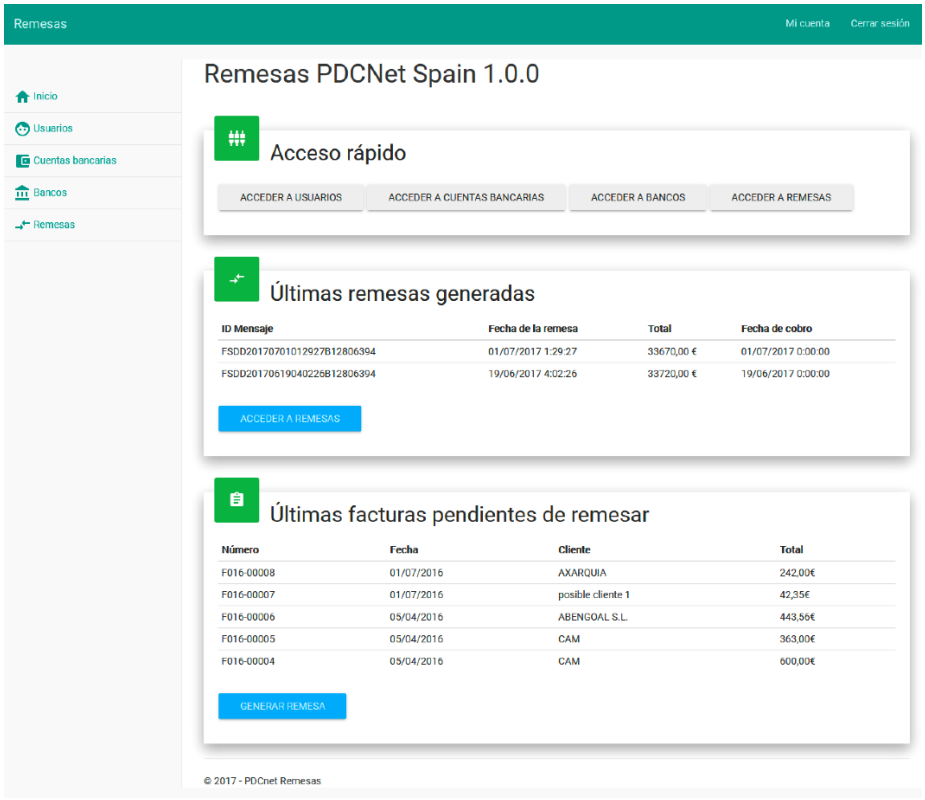



*Figura 25. Diseño final de la página de inicio*

**Caso de uso: Inicio de sesión/registro**

**Escenario 1:** Un usuario, previamente registrado, desea iniciar sesión en la aplicación


*Figura 26. Boceto de la vista de login*

La figura 26 muestra el boceto de la vista de inicio de sesión de usuario. Como se puede observar, se ofrecen dos métodos para iniciar sesión: bien con una cuenta externa de Facebook, LinkedIn, Twitter o Google, o de la forma tradicional con email y contraseña.

Si se inicia sesión con una cuenta externa después de haberse registrado con email y contraseña, la cuenta externa se enlaza con el email ya registrado (siempre que el email de la cuenta externa y el de la aplicación coincidan, si no se crea una cuenta nueva). Si el

usuario introduce un email o contraseña no válidos se le muestra un mensaje de error para que lo intente de nuevo con otras claves.

La figura 27 muestra el diseño final de la interfaz. La base es la misma pero los estilos son más elegantes. También se añadió Microsoft como plataforma externa.

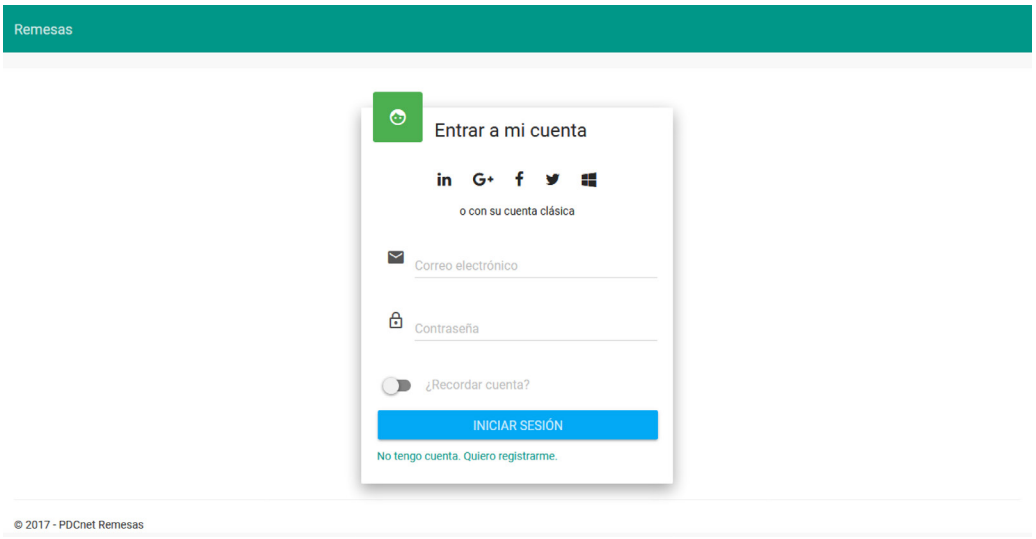



*Figura 27. Diseño final de la vista de login*

**Escenario 2:** Un usuario desea registrarse en la aplicación para poder utilizarla


*Figura 28. Boceto de la vista de registro*

En el formulario de registro se presentan las mismas opciones para registrarse con cuentas externas o manualmente. Al seleccionar una de estas cuentas e iniciar sesión en la correspondiente plataforma el usuario debe confirmar sus datos antes de que el sistema cree su cuenta en la aplicación (ver figura 30 con ejemplo de autenticación con Facebook).

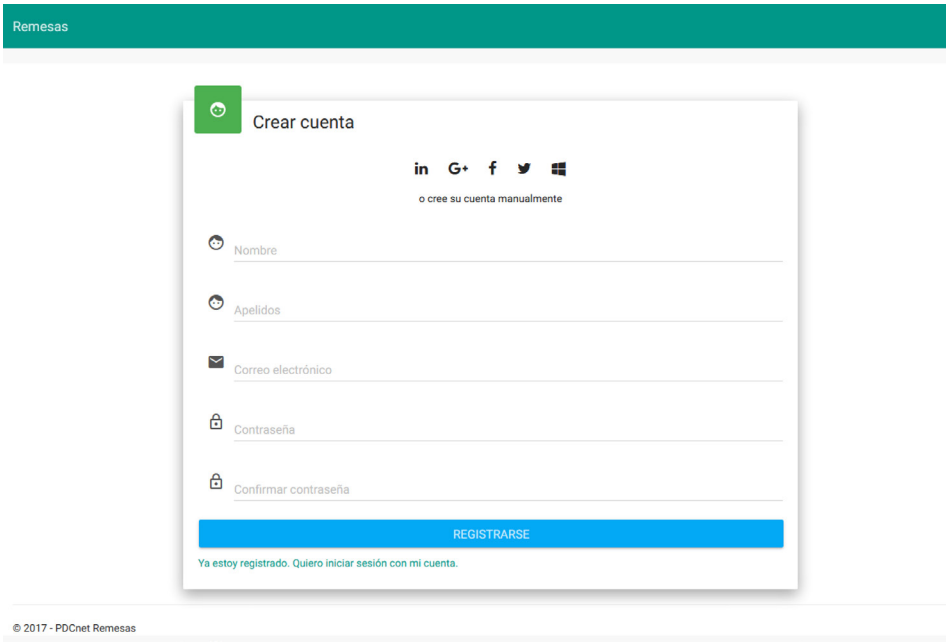



*Figura 29. Diseño final de la vista de registro*

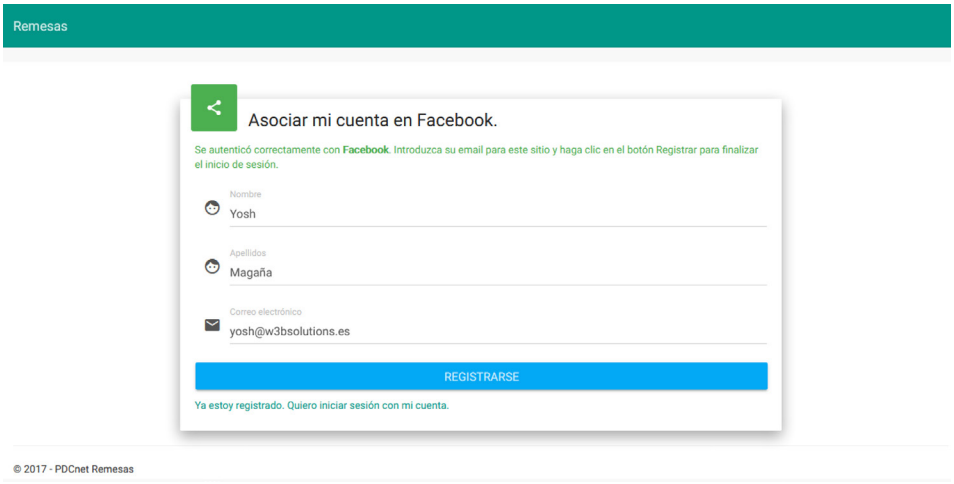



*Figura 30. Vista de confirmación de datos de plataforma externa*

Si un usuario que se ha registrado mediante una plataforma externa desea poder iniciar sesión también manualmente, deberá establecer una contraseña para su cuenta desde el apartado “Mi cuenta” (figura 31).

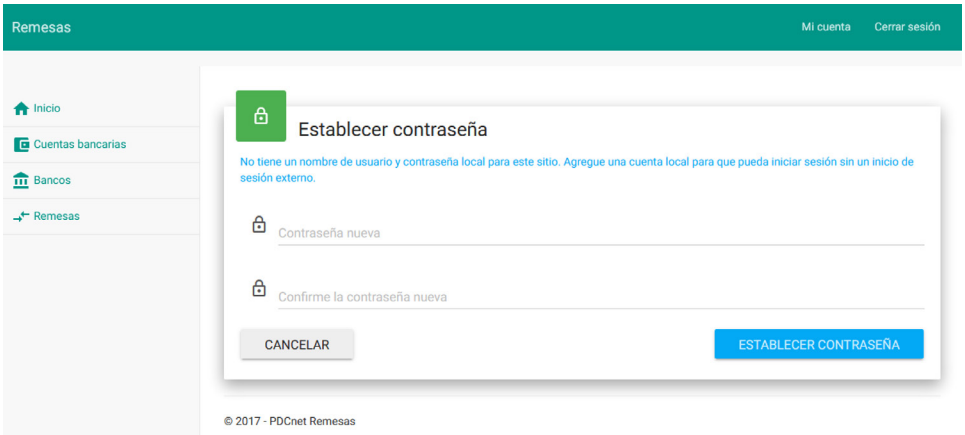



*Figura 31. Establecer contraseña de cuenta*

Si el usuario intenta enviar cualquier formulario de la aplicación con datos incorrectos o campos vacíos dichos campos se resaltan en rojo con un mensaje de error. En la figura 32 se muestra un ejemplo con el formulario de registro manual, pero esto se aplica a todos los formularios.

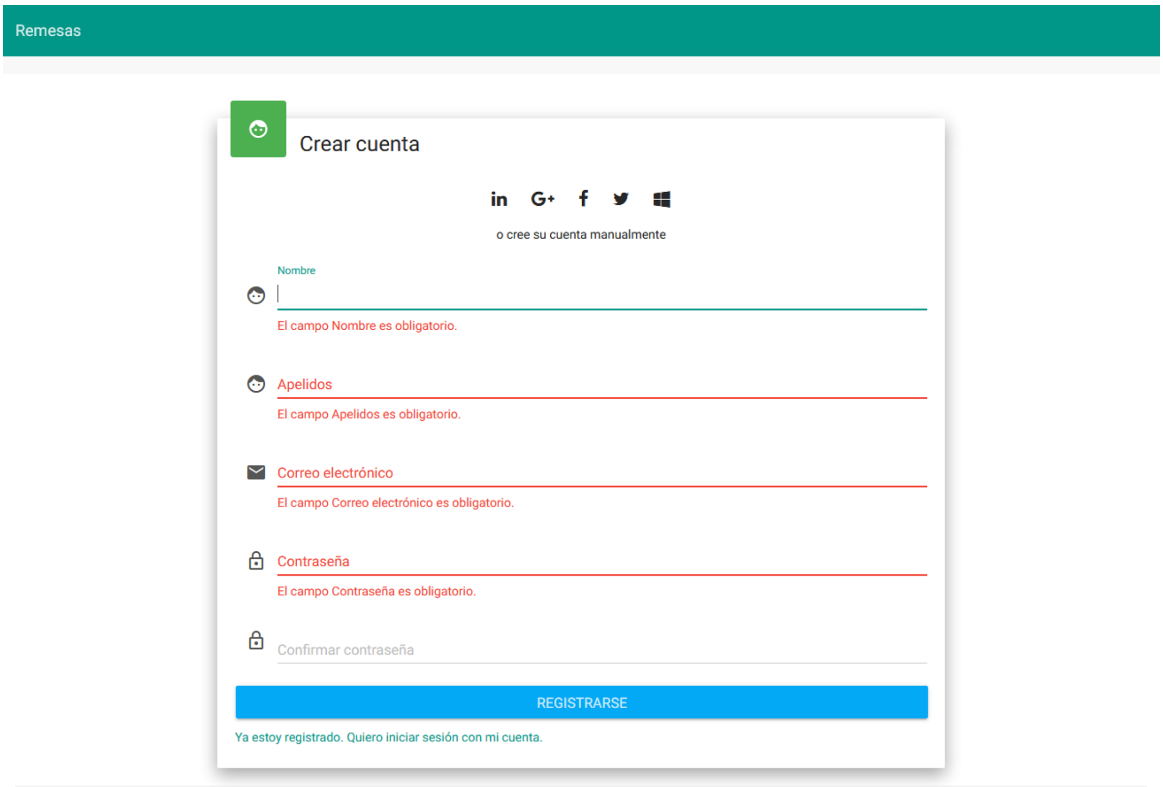



*Figura 32. Validación de formulario de registro*

**Escenario 3:** Un usuario desea modificar los datos de su perfil
Un usuario puede modificar los datos de su perfil accediendo al apartado “Mi cuenta”. Se le presentan 3 opciones como se observa en la figura 33: modificar los datos de su cuenta (figura 34), establecer o modificar su contraseña (figura 31) y gestionar las cuentas enlazadas (figura 35).

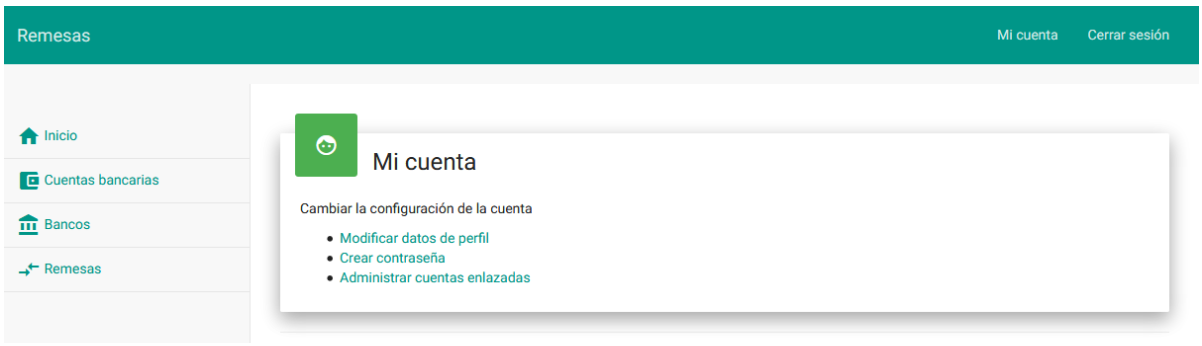



*Figura 33. Apartado “Mi cuenta”*

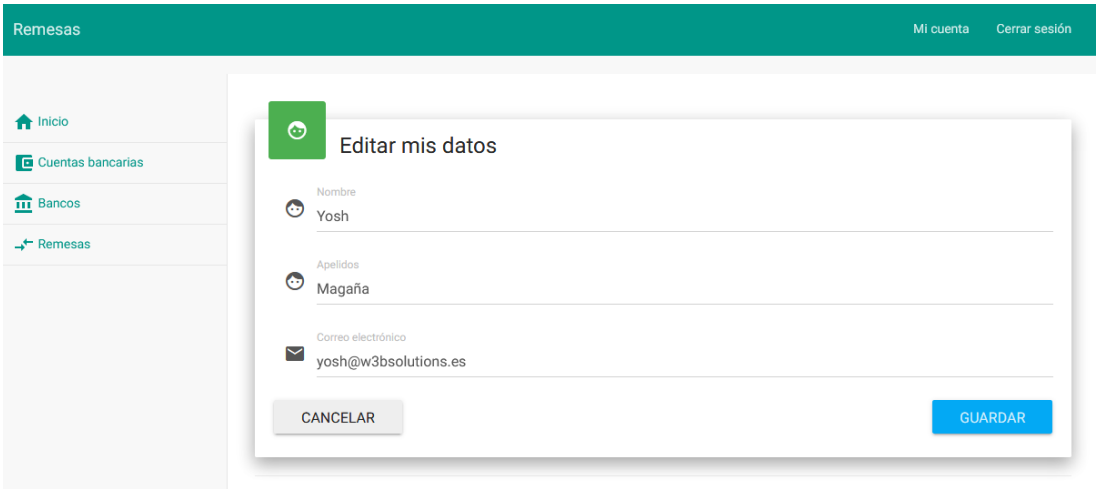



*Figura 34. Modificar datos de perfil de usuario*

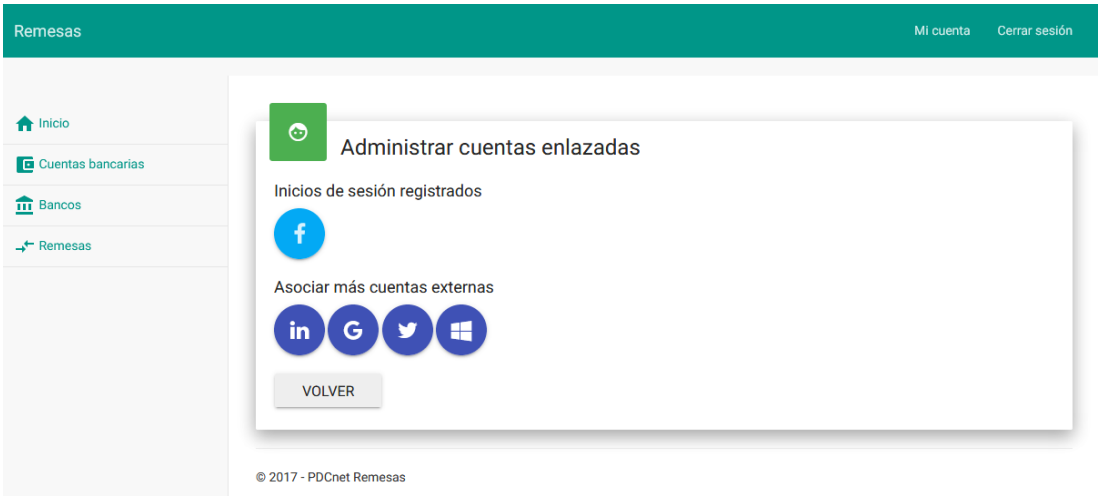



*Figura 35. Gestionar cuentas externas enlazadas con cuenta*

**Caso de uso: Mantener usuarios**

**Escenario 1:** Un administrador quiere dar de alta un nuevo usuario

1. El administrador elige dar de alta un nuevo usuario.
2. Introduce los datos necesarios.
3. Selecciona el rol del nuevo usuario.

Una tabla Objeto-Atributos-Acciones (tabla 15) ayuda a visualizar qué datos se mostrarán de cada objeto y qué acciones se podrán realizar sobre dicho objeto.

| Objeto | Atributos | Acciones |
|---|---|---|
| Usuario | Nombre<br>Apellidos<br>Correo electrónico<br>Contraseña<br>Rol | Añadir<br>Ver<br>Modificar<br>Eliminar |
| Rol | Nombre | Ver<br>Seleccionar |

*Tabla 15. Tabla Objeto-Atributos-Acciones del alta de un usuario*

La figura 36 muestra el diseño final del formulario de alta de un usuario por parte de un administrador. No se han realizado bocetos de todas las vistas ya que los formularios son similares. En este caso se incluye sólo el diseño final.

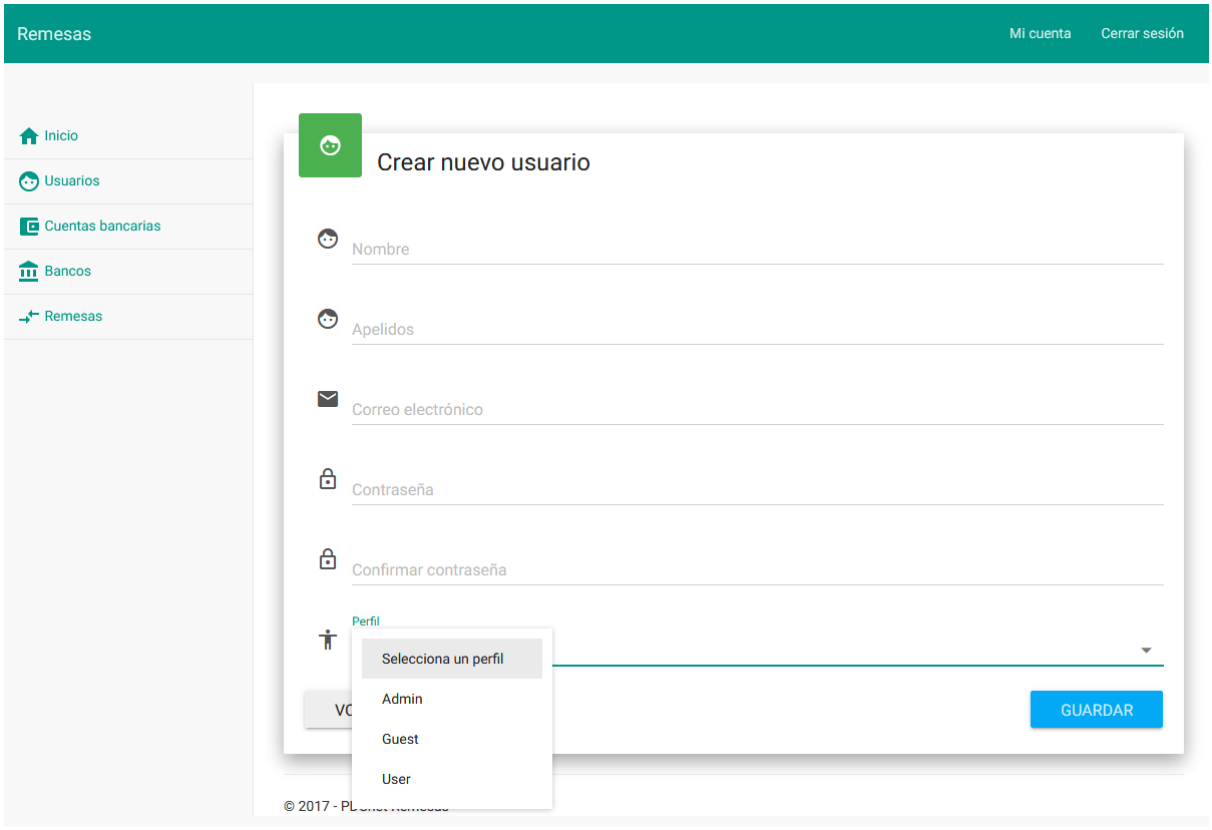



*Figura 36. Formulario de alta de usuario nuevo*

**Escenario 2:** Un administrador quiere modificar los datos de un usuario

1. El administrador lista los usuarios registrados
2. El administrador elige el usuario que quiere modificar.
3. El administrador modifica los datos necesarios.

Como los atributos y acciones del objeto usuario ya se indican en la tabla 15, en la tabla 16 se muestran sólo los de los objetos nuevos, que son la lista de usuarios.

| Objeto | Atributos | Acciones |
|---|---|---|
| Lista de usuarios | Activo<br>Nombre<br>Apellidos<br>Correo electrónico<br>Fecha de registro | Ver<br>Recorrer<br>Filtrar<br>Ordenar |

*Tabla 16. Tabla Objeto-Atributos-Acciones del alta de un usuario*

La figura 37 muestra el boceto de la vista de listado de usuarios diseñada inicialmente. La columna “activo” se diseñó para que al hacer clic sobre el icono del estado de un usuario, éste cambiara de estado. Es decir, si un usuario está inactivo y se hace clic en el icono, se envía la petición al servidor y el usuario pasa a estar activo y el icono se cambia. En el

diseño final, en lugar de poner las acciones en botones individuales se han agrupado en un desplegable a la derecha de cada fila.

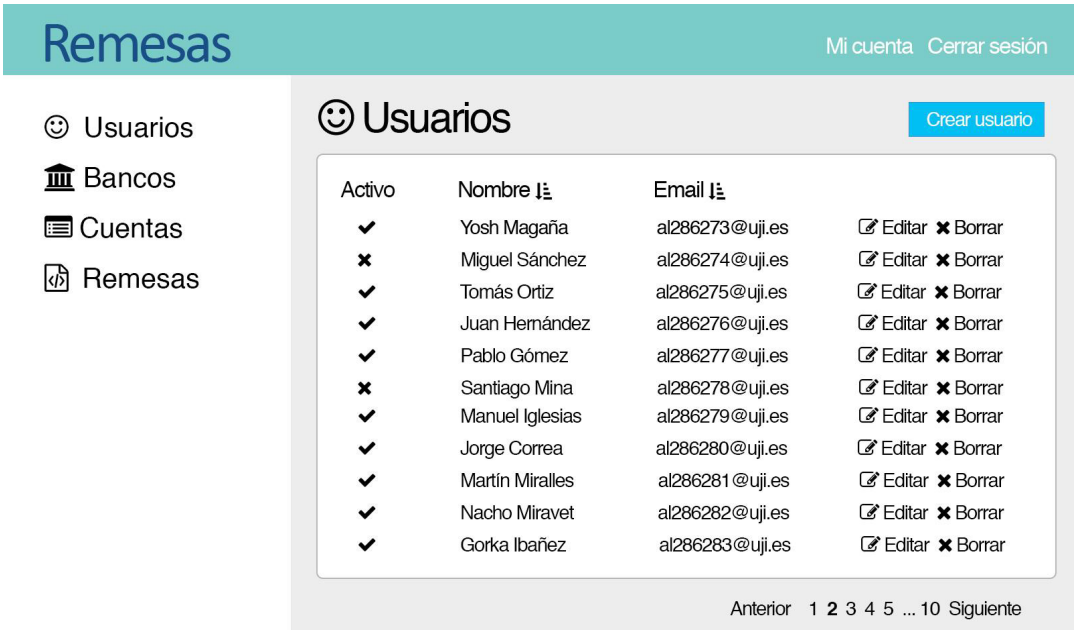



*Figura 37. Boceto de listado de usuarios registrados*

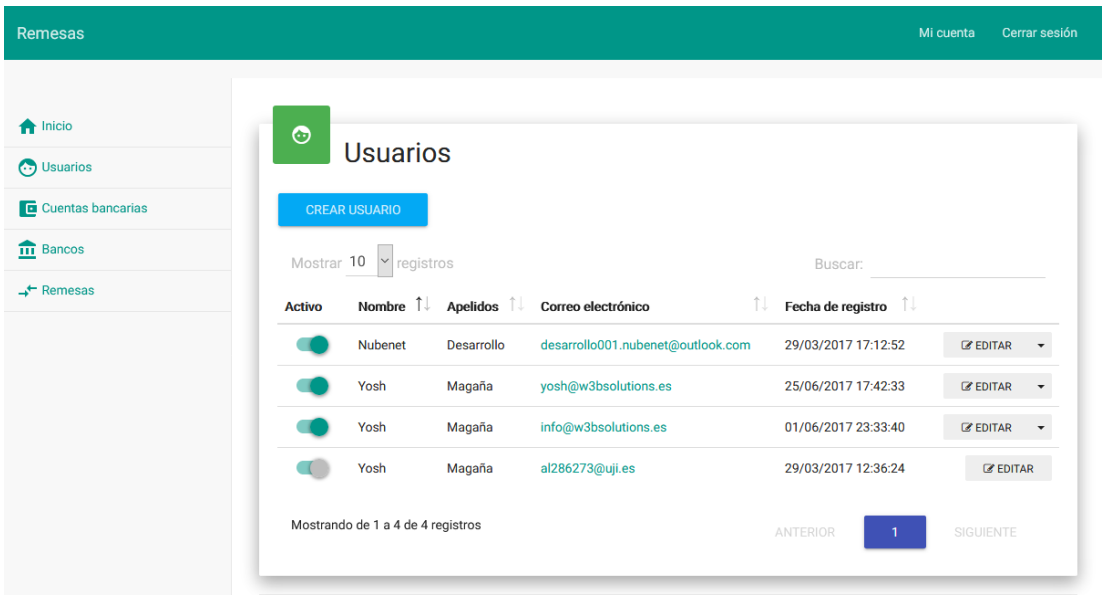



*Figura 38. Vista final de listado de usuarios registrados*

La vista de edición de usuario no se incluye puesto que es la misma que para el alta de un usuario nuevo (figura 36) salvo que el formulario se carga con los datos del usuario en los campos.

**Escenario 3:** Un administrador quiere eliminar un usuario

1. El administrador lista los usuarios registrados.
2. El administrador elige el usuario que quiere eliminar.
3. El administrador confirma la eliminación del usuario.

En este escenario se desea que el administrador confirme que desea borrar el usuario.

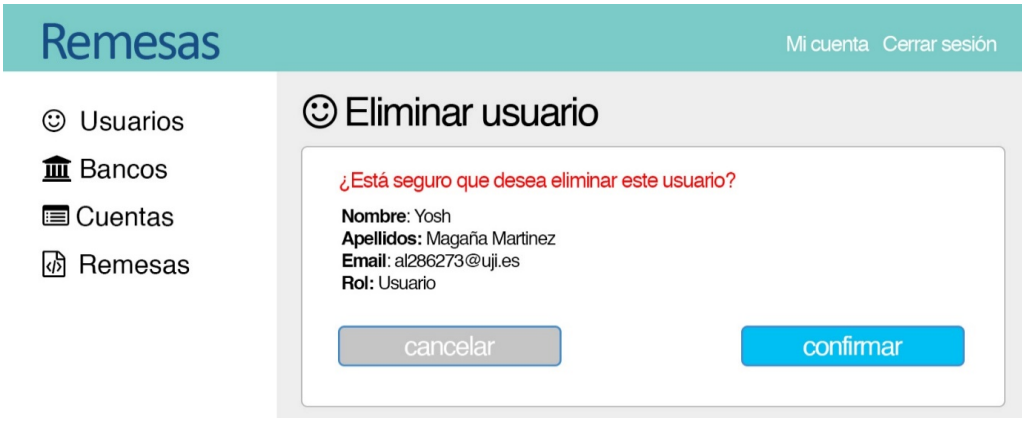



*Figura 39. Boceto de vista de confirmación de borrado de un usuario*

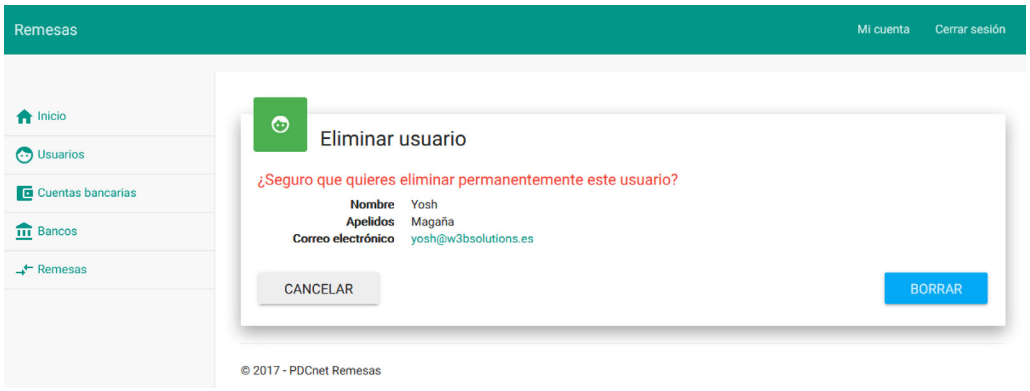



*Figura 40. Vista final de confirmación de borrado de un usuario*

**Caso de uso: Mantener entidades bancarias**

**Escenario 1:** Un usuario desea dar de alta una nueva entidad bancaria

1. El administrador elige dar de alta un nuevo banco.
2. Introduce los datos necesarios.

| Objeto | Atributos | Acciones |
|---|---|---|
| Banco | Nombre<br>BIC | Ver<br>Crear<br>Modificar<br>Eliminar |

*Tabla 17. Tabla Objeto-Atributos-Acciones del alta de un banco*


*Figura 41. Boceto de la vista de alta de nuevo banco*

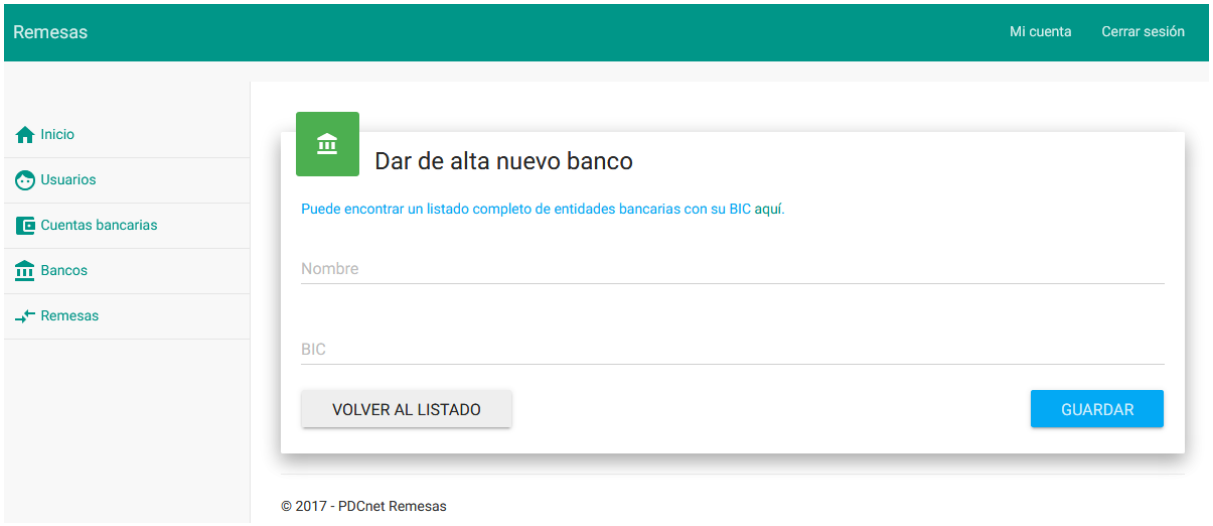



*Figura 42. Diseño final de la vista de alta de nuevo banco*

Se puede apreciar en la figura 42 que en la vista del alta de un nuevo banco se ha incluido un enlace a una página web que tiene una relación de todos los bancos españoles con su correspondiente BIC para facilitar al usuario la introducción de estos datos.

**Escenario 2:** Un usuario desea modificar una entidad bancaria

1. El usuario lista los bancos creados.
2. El usuario elige el banco que quiere modificar.
3. El usuario modifica los datos deseados.

| Objeto | Atributos | Acciones |
|---|---|---|
| Lista de bancos | Nombre<br>BIC<br>Nº de cuentas asociadas | Ver<br>Recorrer<br>Filtrar<br>Ordenar<br>Imprimir |

*Tabla 18. Tabla Objeto-Atributos-Acciones de la modificación de un banco*

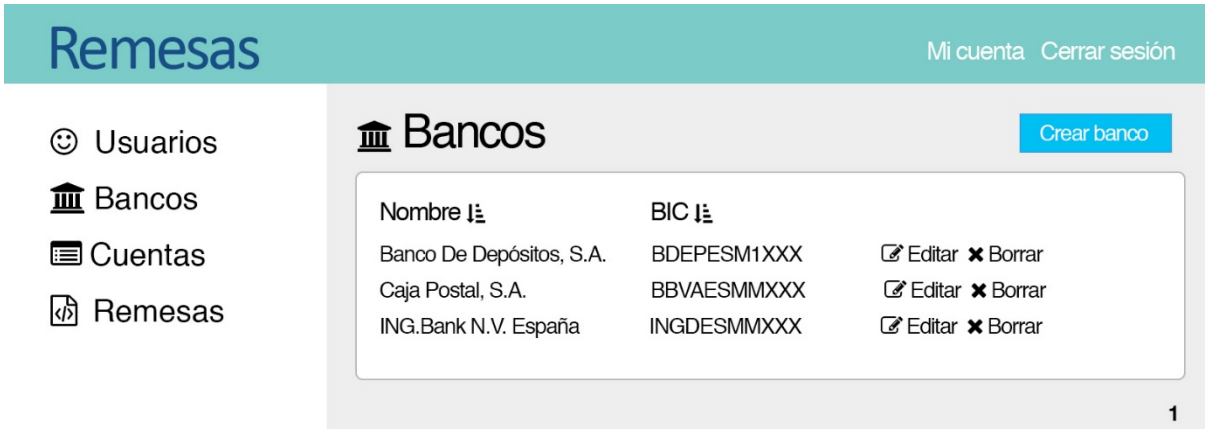



*Figura 43. Boceto de la vista de listado de bancos creados*

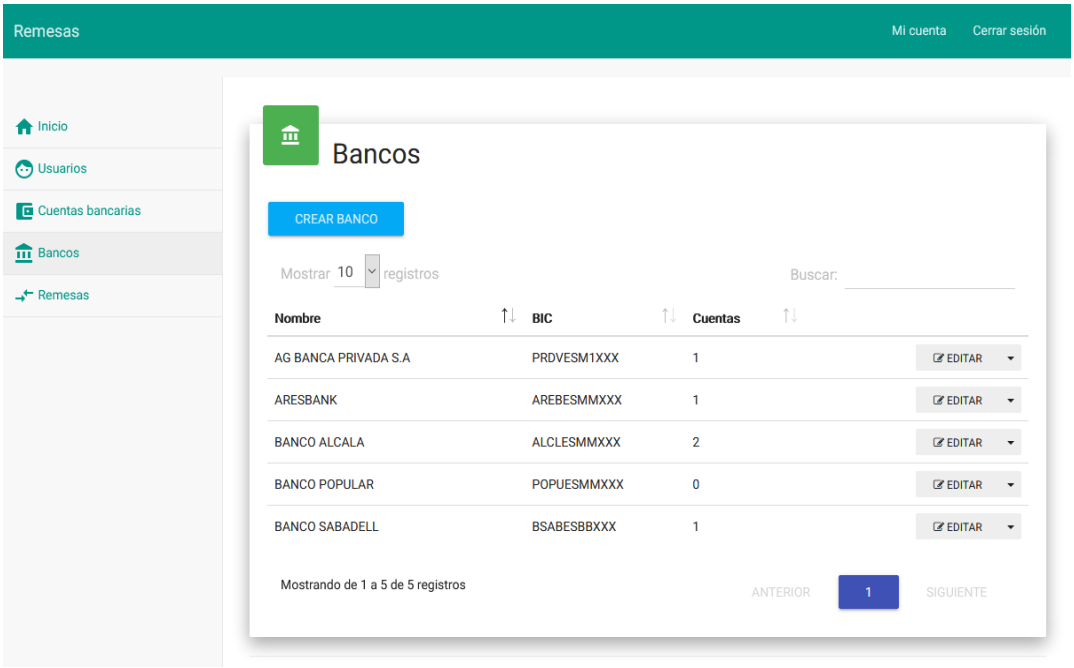



*Figura 44. Diseño final de la vista de alta de bancos*

En el listado de bancos de la figura 44 no se puede eliminar un banco que tiene cuentas bancarias asociadas. En caso de tenerlas, se deben eliminar primero para poder eliminar el banco al que pertenecen. La vista del formulario de edición de bancos (figura 45) es igual que la de alta de un nuevo banco, salvo que si un banco ya tiene cuentas bancarias asociadas se incluyen para que el usuario sepa que esas cuentas se verán afectadas por los cambios.

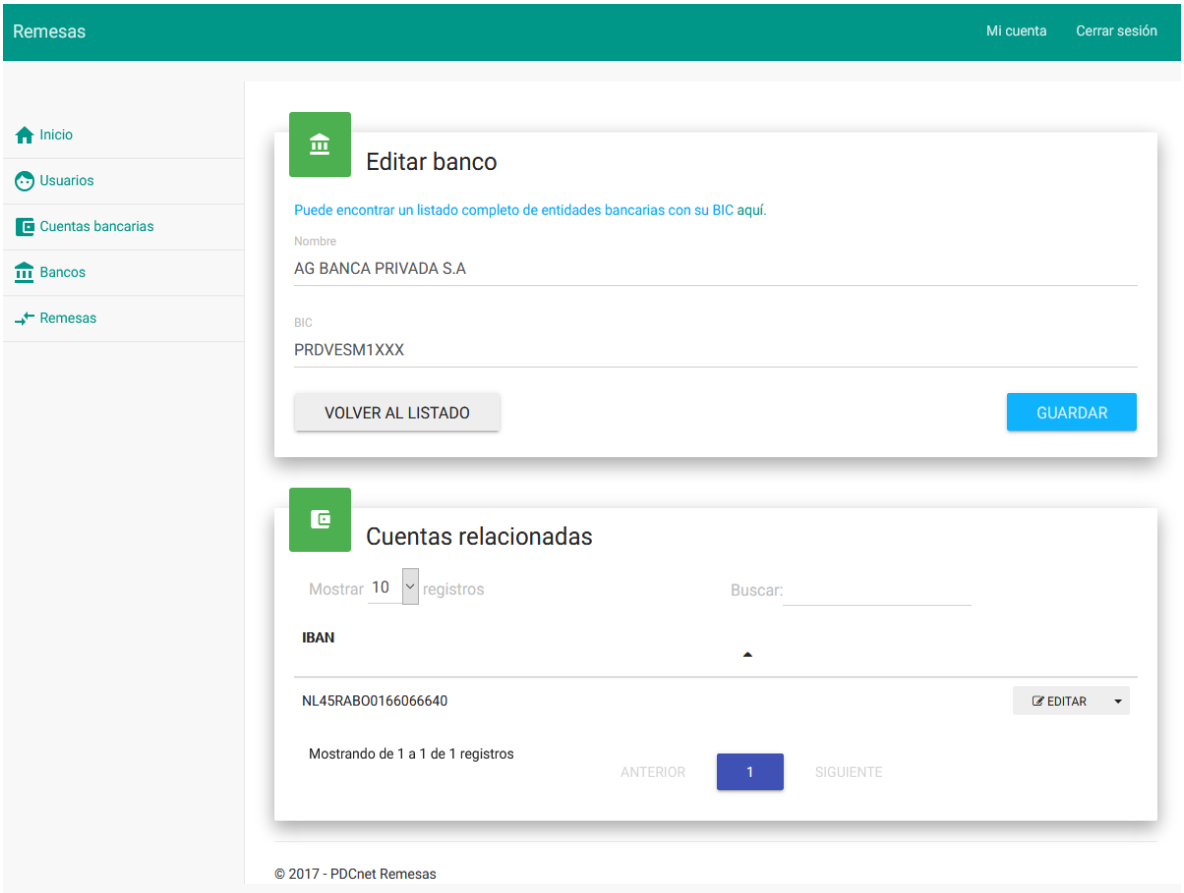



*Figura 45. Vista de modificación de banco con cuentas asociadas*

**Escenario 3:** Un usuario desea eliminar una entidad bancaria

1. El usuario lista las entidades bancarias registradas.
2. El usuario elige la entidad bancaria que quiere eliminar.
3. El usuario confirma la eliminación de la entidad.

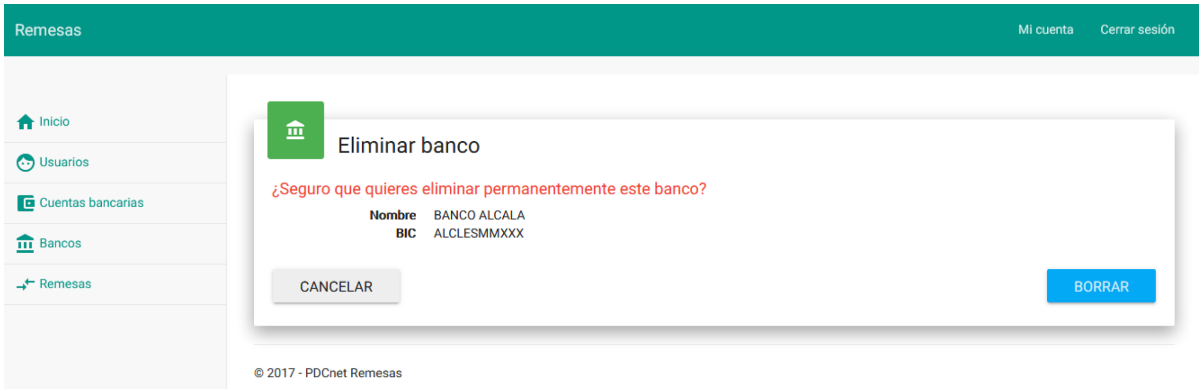



*Figura 46. Vista de confirmación de eliminación de banco*

**Caso de uso: Mantener cuentas bancarias**

**Escenario 1:** Un usuario desea dar de alta una cuenta bancaria

1. El administrador elige dar de alta una nueva cuenta bancaria.
2. Introduce los datos necesarios.
3. Selecciona el banco a la que pertenece la cuenta bancaria.

| Objeto | Atributos | Acciones |
|---|---|---|
| Cuenta bancaria | Nombre<br>BIC | Ver<br>Crear<br>Modificar<br>Eliminar |
| Banco | Nombre | Ver<br>Seleccionar |

*Tabla 19. Tabla Objeto-Atributos-Acciones de la vista de alta de nueva cuenta bancaria*

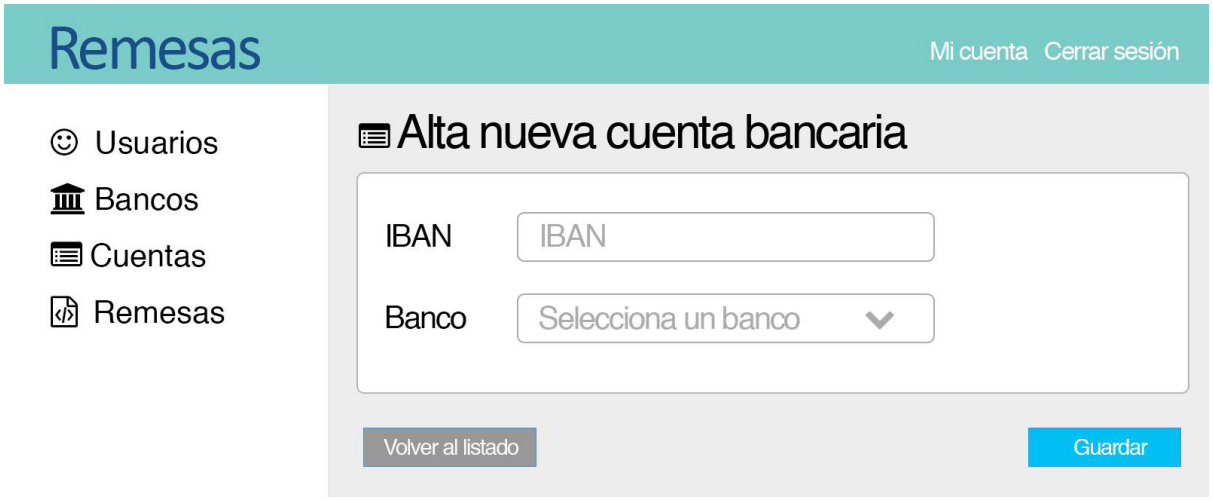



*Figura 47. Boceto de la vista de alta de cuenta bancaria*

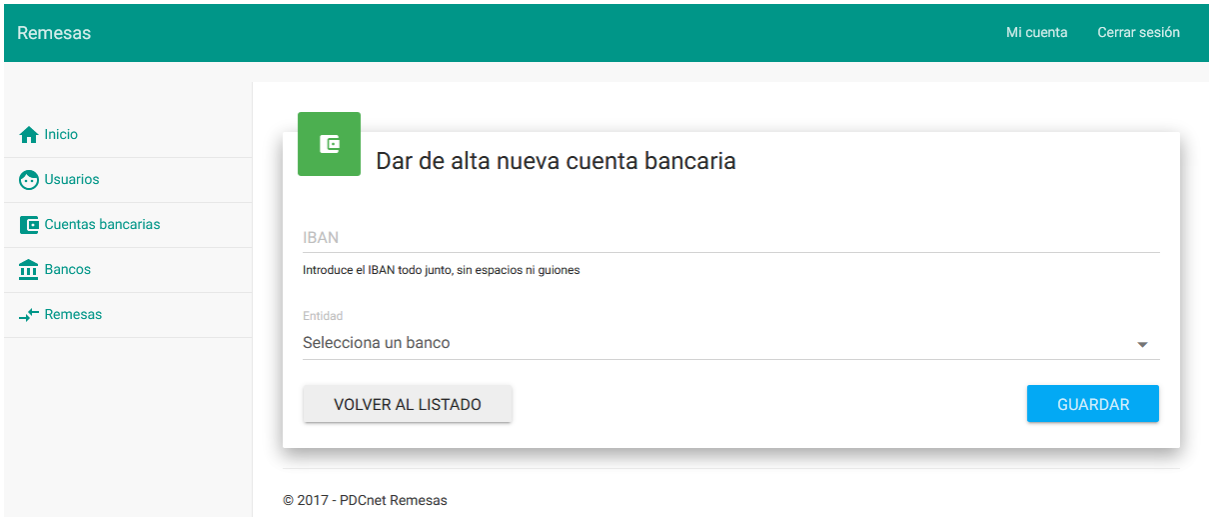



*Figura 48. Vista final de alta de cuenta bancaria*

**Escenario 2:** Un usuario desea modificar una cuenta bancaria

1. El usuario lista las cuentas bancarias creadas.
2. El usuario elige la cuenta que quiere modificar.
3. El usuario modifica los datos deseados.

| Objeto | Atributos | Acciones |
|---|---|---|
| Lista de cuentas bancarias | IBAN<br>Entidad | Ver<br>Recorrer<br>Filtrar<br>Ordenar<br>Imprimir |

*Tabla 20. Tabla Objeto-Atributos-Acciones de la vista de modificación de nueva cuenta bancaria*


*Figura 49. Boceto de listado de cuentas bancarias*

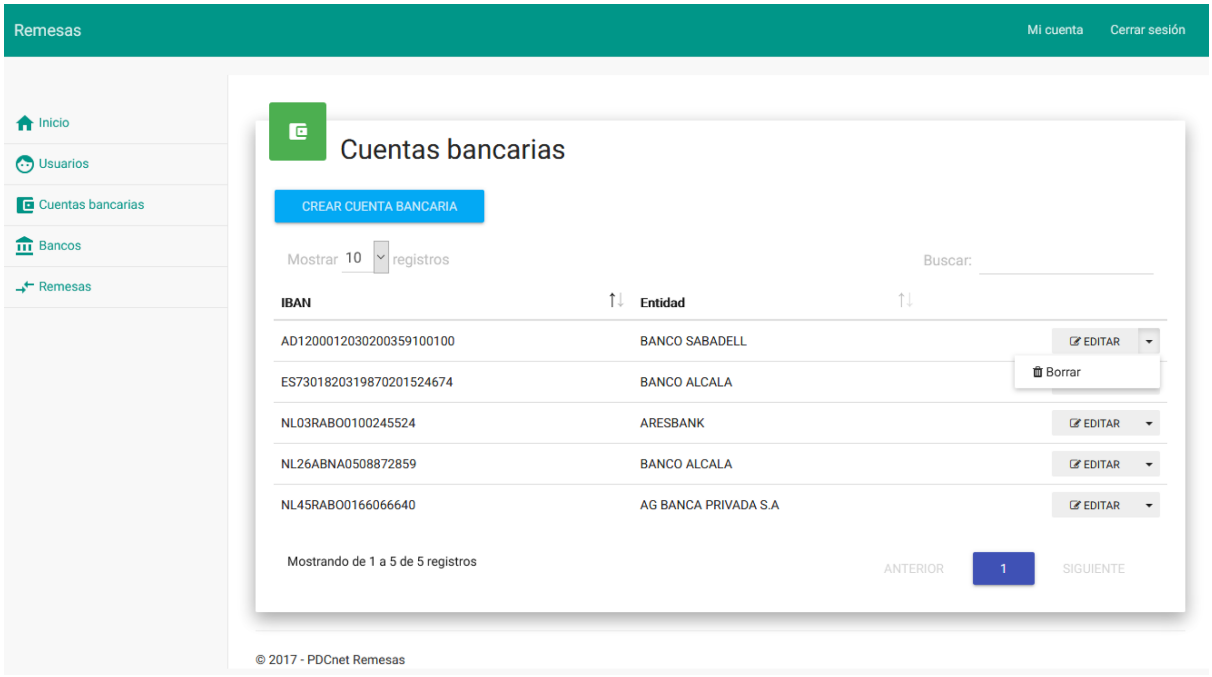



*Figura 50. Vista final de listado de cuentas bancarias*

**Escenario 3:** Un usuario desea eliminar una cuenta bancaria

1. El usuario lista las cuentas bancarias registradas.
2. El usuario elige la cuenta bancaria que quiere eliminar.
3. El usuario confirma la eliminación de la cuenta.

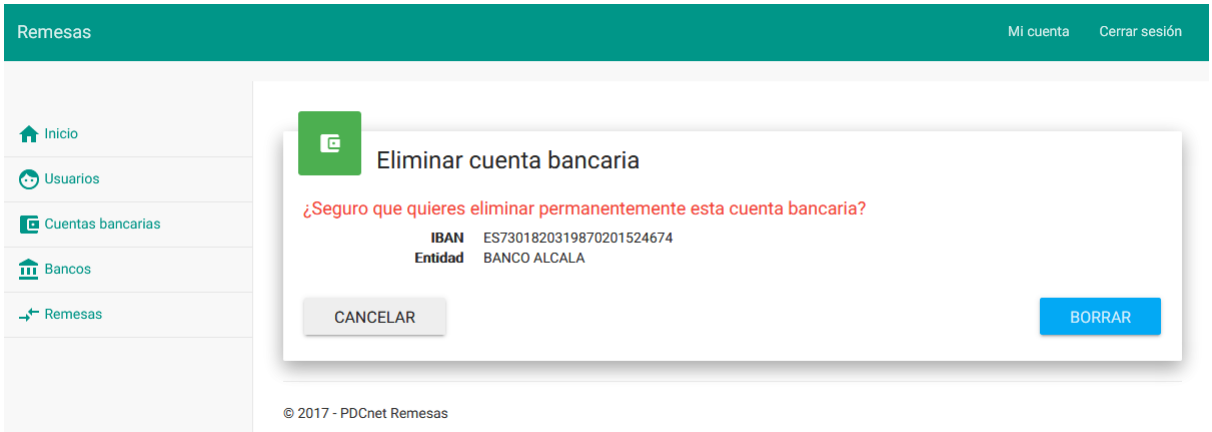



*Figura 51. Vista de confirmación de borrado de cuenta bancaria*

**Caso de uso: Mantener remesas**

**Escenario 1:** Un usuario desea crear una remesa

1. El usuario elige generar una nueva remesa.
2. Introduce los datos necesarios.
3. Selecciona la cuenta bancaria donde quiere recibir el cobro.
4. Selecciona las facturas que se incluyen en la nueva remesa del listado de facturas disponibles.

| Objeto | Atributos | Acciones |
|---|---|---|
| Remesa | Nombre del presentador<br>NIF del presentador<br>Fecha de la remesa<br>Fecha de cargo<br>Cuenta de ingreso<br>Facturas | Ver<br>Crear<br>Modificar<br>Eliminar<br>Descargar XML<br>Descargar XML B2B |
| Lista de facturas disponibles | Número<br>Cliente<br>Fecha de factura<br>Total | Ver detalles de factura<br>Recorrer<br>Filtrar<br>Ordenar |
| Factura | Número<br>Cliente<br>Fecha de factura<br>Total | Incluir en remesa<br>Excluir de remesa |
| Cuenta bancaria | IBAN<br>Entidad bancaria | Ver<br>Seleccionar |

*Tabla 21. Tabla Objeto-Atributos-Acciones de la creación de una nueva remesa*


*Figura 52. Boceto de la vista de creación de remesa*

La figura 52 muestra el boceto inicial de la ventana de creación de remesa, incluyendo un bloque superior con la información relevante para la remesa y un bloque inferior con el listado de facturas disponibles, pudiendo filtrarlas según su número, cliente, fechas o si están previamente remesadas o no. Para cada factura se incluye un botón para "Ver detalles" de la factura que no caben en la fila. La figura 53 muestra el boceto inicial de esta vista de detalles de una factura.


*Figura 53. Vista de detalles de factura en bloque de selección de facturas*

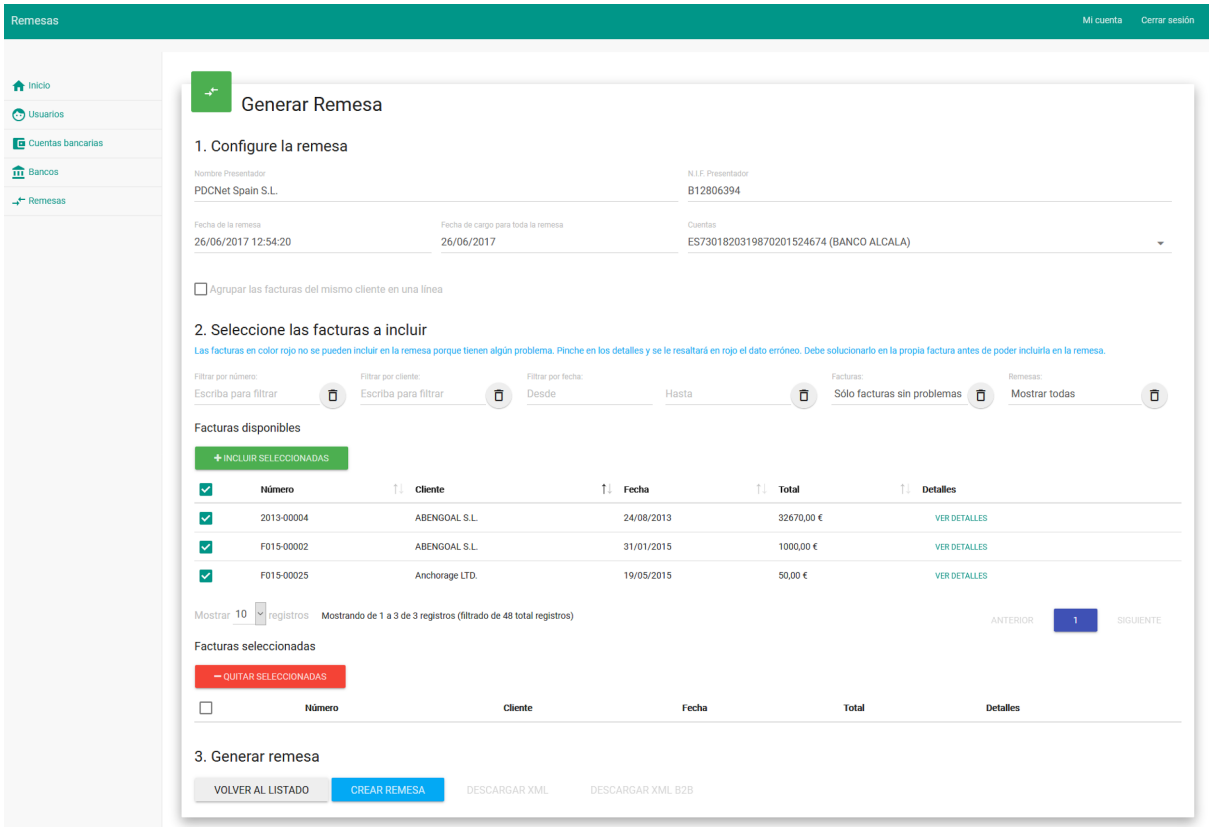



*Figura 54. Diseño final de vista de creación de remesa*

En el diseño final de la vista para generar una remesa se han creado dos bloques con listas de facturas: un bloque para las facturas disponibles para elección y otro para las facturas seleccionadas. Al seleccionar facturas y pinchar en "incluir seleccionadas" estas facturas se

mueven al bloque de “facturas seleccionadas”. Si luego el usuario desea deshacer el movimiento basta con seleccionarlas de nuevo y hacer clic en “quitar seleccionadas”.

Entre los filtros del bloque de selección de facturas hay uno que permite elegir entre “Sólo facturas sin problemas”, “Sólo facturas con problemas” o “Mostrar todas”. Este desplegable permite visualizar si se han creado facturas que contienen datos erróneos que impiden que puedan ser seleccionadas para incluir en una remesa. Si el usuario elige ver las facturas con problemas y hay facturas de este tipo se obtiene un resultado como la figura 55, y si hace clic en “ver detalles” se muestra en rojo qué datos son incorrectos de la factura, en el caso de la figura 55 el IBAN no cumple las reglas de validación. En estos casos el usuario debería rectificar los datos incorrectos de la factura en la aplicación externa donde crea las facturas previo a incluirla en la remesa.

También se puede apreciar en la figura 54 que en la barra inferior hay dos botones deshabilitados: “descargar XML” y “descargar XML B2B”. Estos botones se habilitan automáticamente cuando la remesa se ha guardado. Es decir, después de introducir los datos de la remesa y seleccionar las facturas que se deben incluir, el usuario hace clic en “Crear remesa” y al guardarse los datos en el servidor se activan estos botones, pudiendo el usuario descargar ya los ficheros XML que debe enviar al banco.

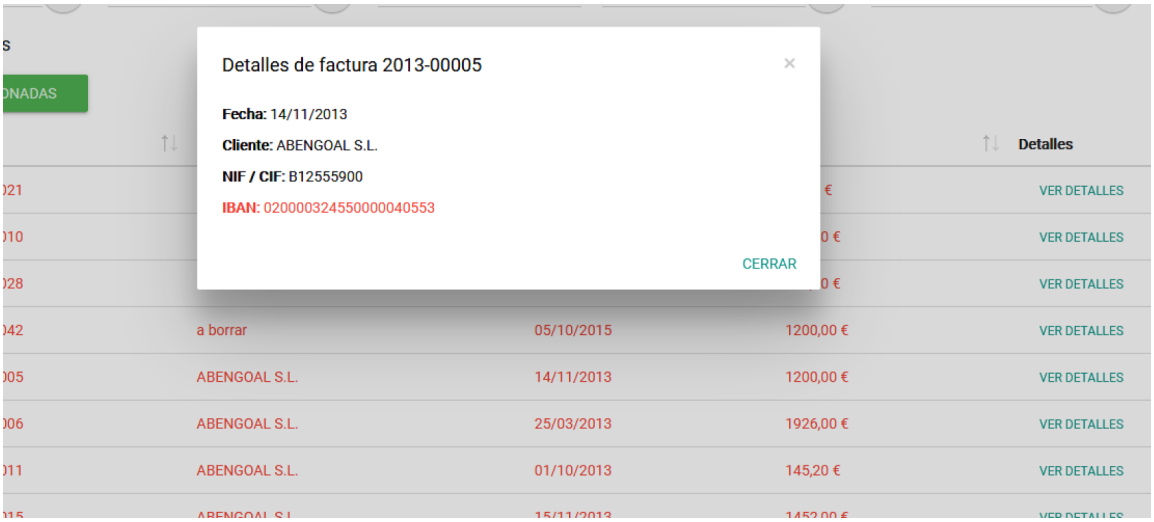



*Figura 55. Validación de datos de facturas en creación de remesa y vista de detalles final*

**Escenario 2:** Un usuario desea editar una remesa

1. El usuario lista las remesas creadas.
2. El usuario elige la remesa que quiere modificar.
3. El usuario modifica los datos o facturas incluidas.

| Objeto | Atributos | Acciones |
|---|---|---|
| Lista de remesas | ID Mensaje<br>Fecha de la remesa<br>Fecha de cobro<br>Número de operaciones<br>Total | Ver<br>Recorrer<br>Filtrar<br>Ordenar<br>Imprimir |

*Tabla 22. Tabla Objeto-Atributos-Acciones del listado de remesas*


*Figura 56. Boceto del listado de remesas*

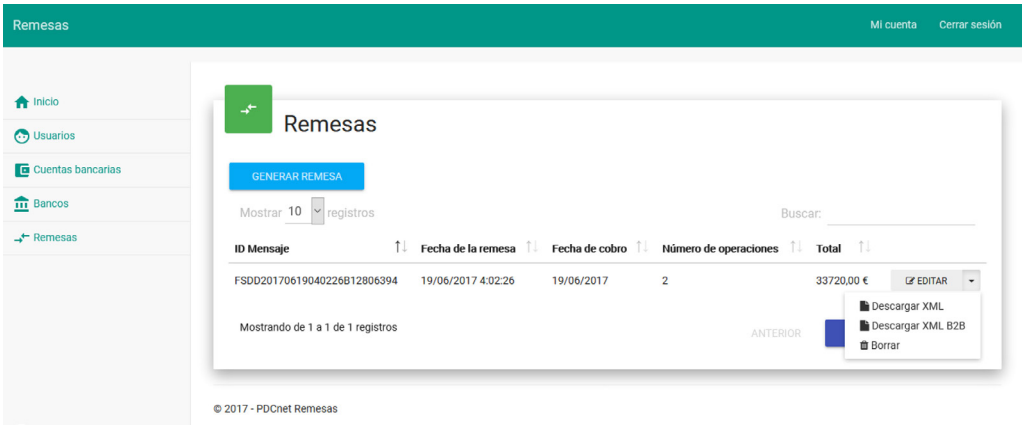



*Figura 57. Diseño final del listado de remesas*

La vista de edición de una remesa es la misma que para el alta de una remesa, sólo que se carga con los datos y las facturas ya seleccionadas de la remesa que se está editando.

**Escenario 3:** Un usuario desea borrar una remesa

1. El usuario lista las remesas registradas.
2. El usuario elige la remesa que quiere eliminar.
3. El usuario confirma la eliminación de la remesa.

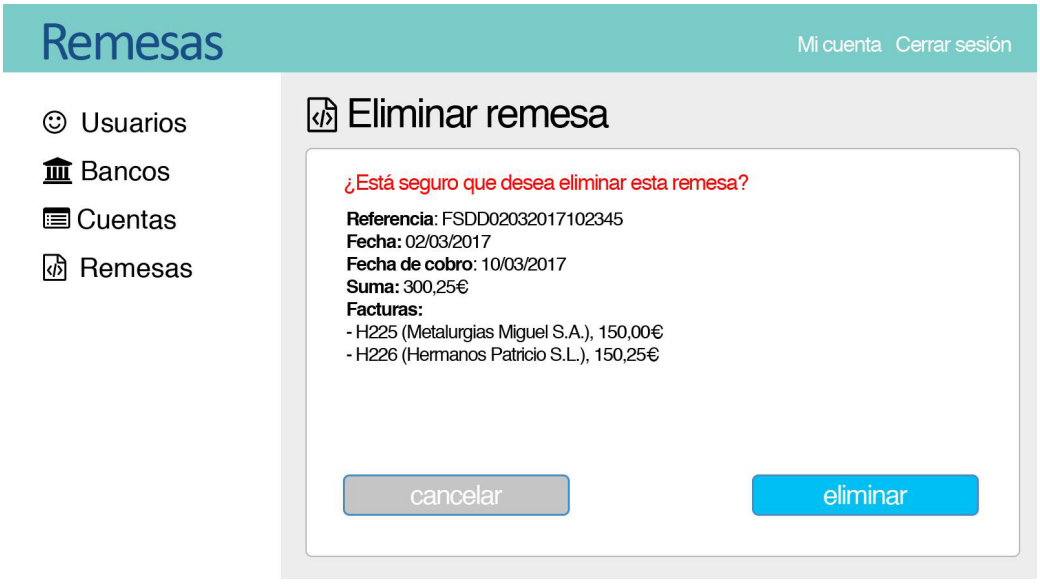



*Figura 58. Boceto de confirmación de borrado de remesa*

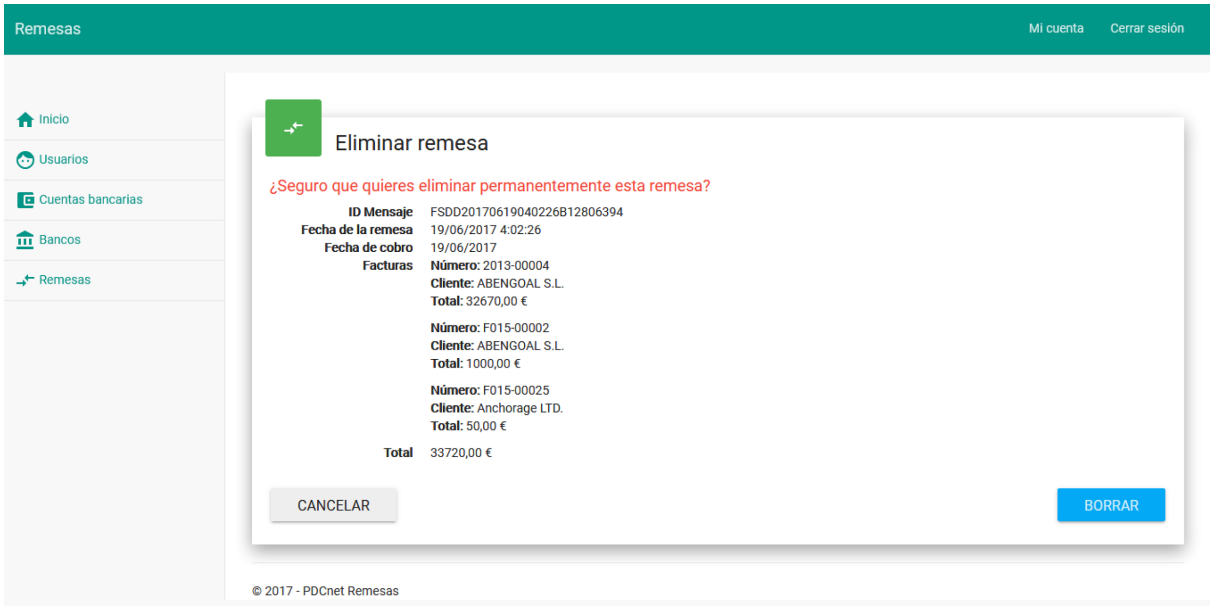



*Figura 59. Diseño final de confirmación de borrado de remesa*

**Escenario 4:** Un usuario desea descargar el XML de una remesa

1. El usuario lista las remesas registradas.
2. El usuario elige la remesa de la que quiere descargar el XML.

Este escenario no incluye una vista específica ya que se trata más de una acción. Para ejecutarla el usuario deberá hacer clic en el botón correspondiente al XML que desea descargar (bien Core o B2B) del listado de remesas (véase la figura 57) y se le presenta directamente la ventana del navegador para guardar el fichero (figura 60).

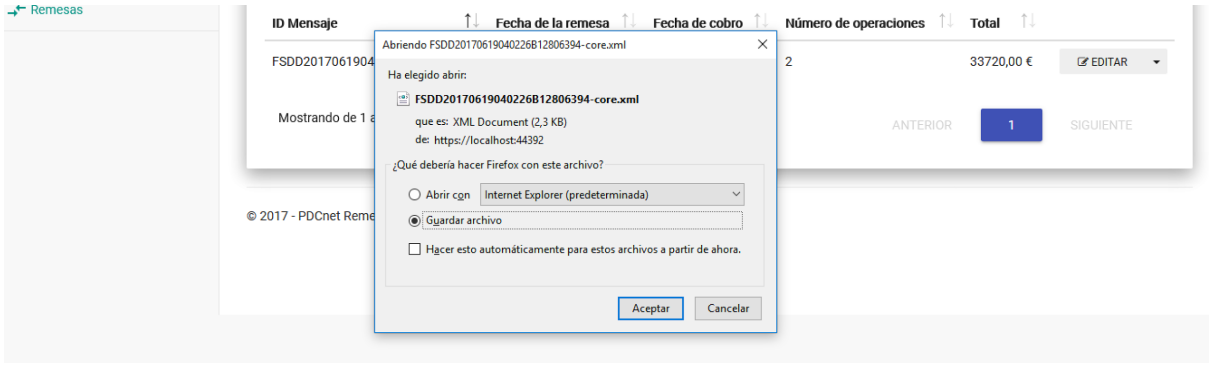



*Figura 60. Ventana de descarga de fichero en Firefox*

**Versión móvil (*responsive*)**

Todas las vistas de la aplicación se han desarrollado teniendo en cuenta el requisito de que la interfaz web debía ser adaptable. Las tablas automáticamente ocultan información y la añaden un botón en cada fila para mostrar u ocultar dicha información. La figura 61 muestra algunos apartados en su versión móvil.

Tanto el menú de usuario para modificar su perfil como el menú principal de la aplicación para acceder a los distintos apartados se juntan en un sólo menú desplegable al que se accede mediante las tres rayitas de la barra superior.

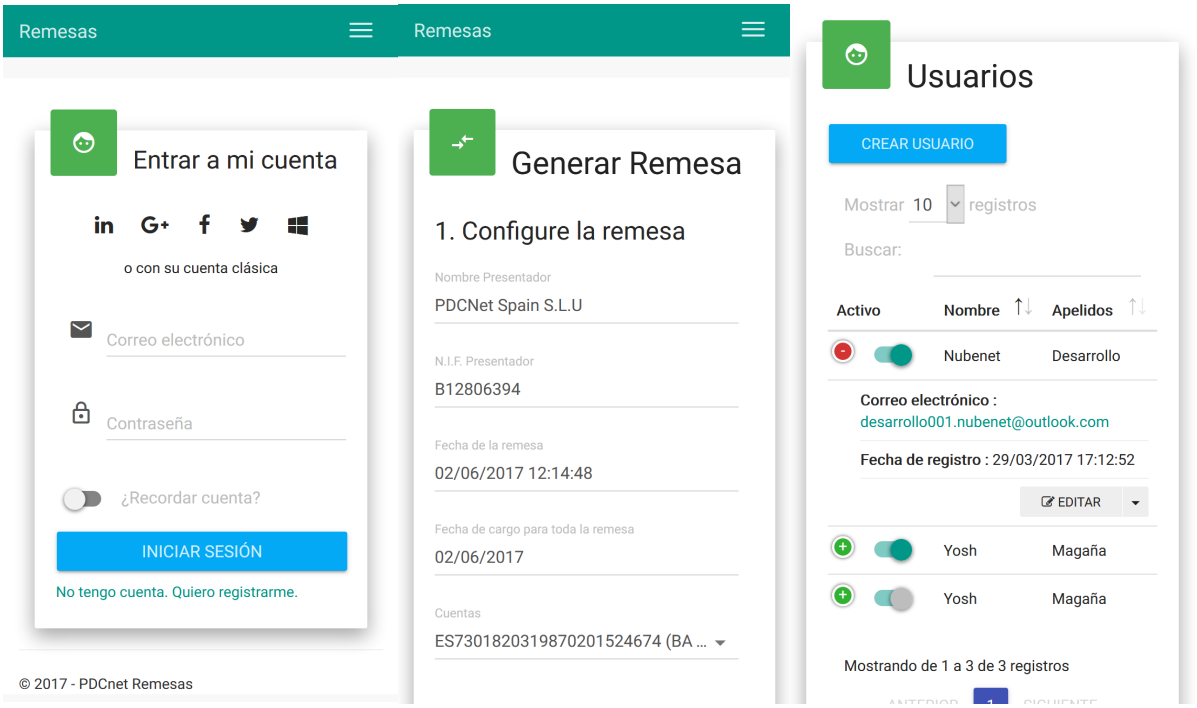



*Figura 61. Diseños finales responsive de la vista de inicio de sesión, creación de remesa y listado de usuarios*

# Capítulo 5. Implementación y verificación

Este capítulo trata detalles de la implementación del proyecto, como los patrones de diseño que se han utilizado, la estructura del código y los componentes de la aplicación y los lenguajes de programación utilizados.

## 5.1 Patrones de diseño

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción entre componentes o interfaces. Un patrón de diseño resulta ser una solución conocida a un problema de diseño. Las ventajas principales de los patrones de diseño son que proporcionan catálogos de elementos reusables, evitan la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente y estandarizan el modo en que se realiza el diseño [8]. En el desarrollo de este proyecto se ha decidido utilizar algunos patrones de diseño que se describen a continuación con su justificación y ventajas.

**Modelo-Vista-Controlador (MVC)**
El objetivo principal del patrón de diseño MVC es separar los datos de la lógica de negocio y la presentación de un sistema de software. La capa de datos se conoce como Modelo, la capa de lógica de negocio como Controlador y la capa de presentación como Vista. A su vez, estas capas se pueden dividir en otras capas para implementar incluso una mayor separación de intereses. MVC define ciertas reglas que indican cómo deben interactuar estas capas entre sí:

- El Modelo gestiona todos los accesos a los datos, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio).

- El Controlador responde a eventos del usuario e invoca peticiones al modelo cuando se hace alguna solicitud sobre la información. También puede enviar comandos a su vista asociada si se solicita un cambio en la forma en que se presenta el modelo.

- La Vista es la interfaz de usuario, que presenta el modelo en un formato adecuado para interactuar [4].

En la figura 62 se muestra el diseño del patrón MVC en aplicación ASP.net.

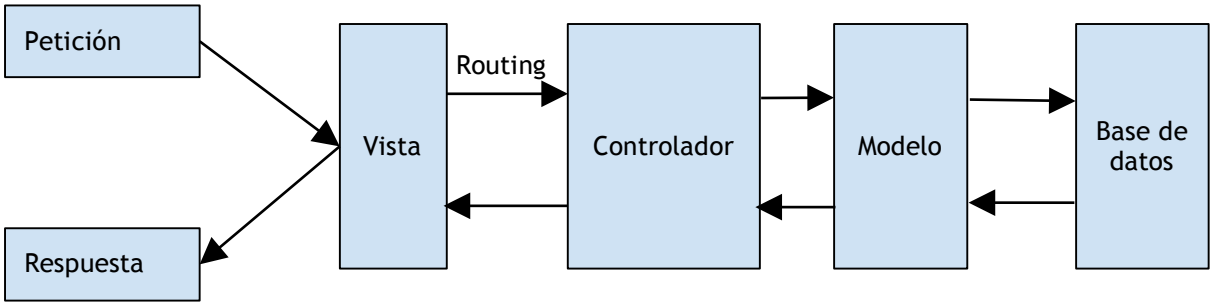



*Figura 62. Capas del patrón de diseño MVC en aplicaciones ASP.net MVC*

*Routing* entre la vista y el controlador se refiere al patrón de búsqueda del controlador encargado de responder a la petición del usuario, es decir, el *enrutado*. El controlador se localiza mediante la estructura de la URL.

**Inversión de Control e Inyección de Dependencias**

Inversión de control (*Inversion of Control* en inglés, o IoC) es un método de programación en el que el flujo de ejecución de un programa se invierte respecto a los métodos de programación tradicionales, en los que la interacción se expresa de forma imperativa haciendo llamadas a procedimientos o funciones. Es el principio subyacente a la técnica de Inyección de Dependencias que también se ha utilizado en este proyecto [9].

La Inyección de Dependencias es un patrón que consiste en colocar dentro de un objeto otros que puedan cambiar su comportamiento o le permitan acceder a las funcionalidades o métodos que ofrece dicho objeto, sin que esto implique volver a crear el objeto. Esto permite tener un objeto que puede hacer un conjunto de tareas, siendo cada una de esas tareas, una responsabilidad que puede ser ejecutada por otro objeto especializado y dedicado a ello. La inyección de dependencias es responsable de establecer las referencias entre objetos en tiempo de ejecución sin que estos tengan que instanciarse dentro del código.

Una forma de implementación muy habitual de la técnica de inyección de dependencias es el uso de interfaces. La ventaja principal del uso de interfaces es que permite sustituir la implementación de un componente sin tener que modificar el código que utiliza dicho componente. Esta ventaja tiene a su vez otra gran ventaja, que es facilitar el diseño de pruebas de validación del código con implementaciones distintas de los componentes. Este proyecto incluye varias interfaces que se han creado principalmente con el fin de simular su comportamiento en la implementación de las pruebas unitarias de validación de los controladores. La figura 63 muestra el diagrama de interfaces del proyecto.

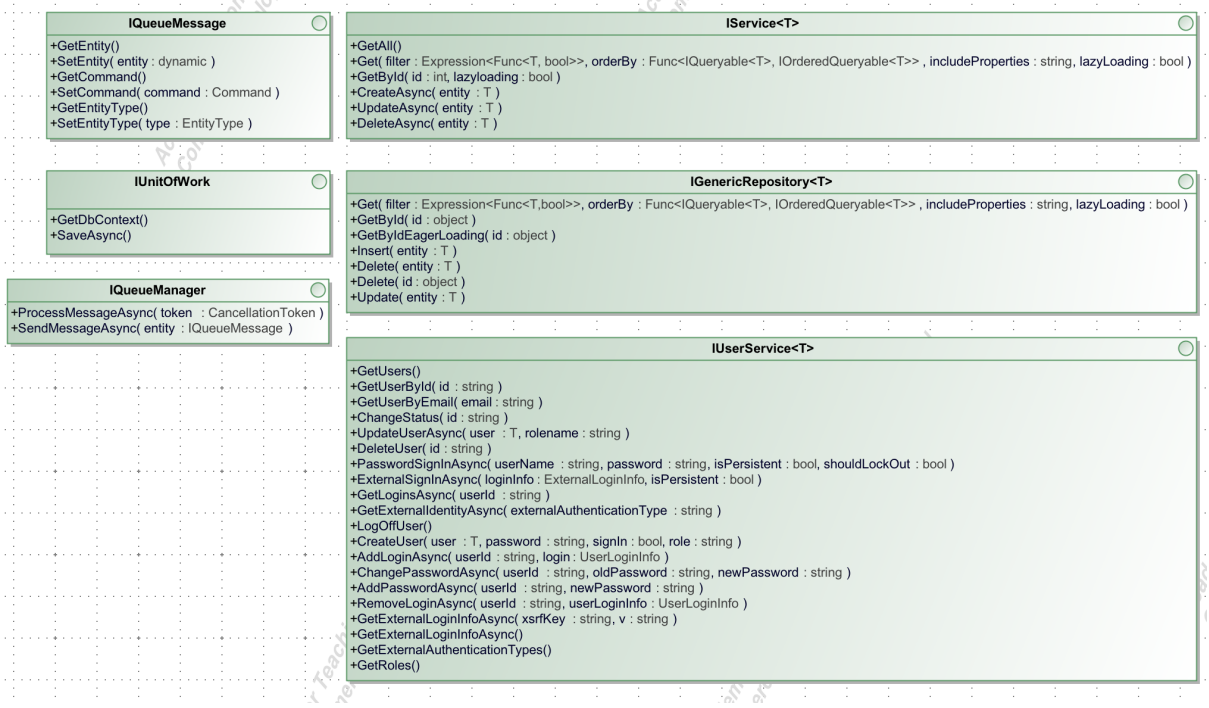



*Figura 63. Diagrama de interfaces de la aplicación*

La inyección de dependencias se ha implementado en este proyecto con el paquete de Nuget Unity de Microsoft. Unity Container es un contenedor de inyección de dependencias ligero y extensible. Facilita la creación de aplicaciones con bajo acoplamiento y proporciona las siguientes ventajas principales:

1. Creación simplificada de objetos, especialmente para estructuras y dependencias de objetos jerárquicos.
2. Abstracción de requisitos: esto permite especificar dependencias en tiempo de ejecución o en su configuración.
3. Capacidad de localización de servicios: esto permite a los clientes tanto almacenar como crear una caché del contenedor.
4. Intercepción de tipo e instancia.
5. Registro por convención.

Al añadir el paquete de Unity se crea un fichero de configuración en la carpeta App_Start del proyecto MVC principal llamado UnityConfig.cs donde se deben registrar los servicios que se desean poder inyectar en los controladores o cualquier otra clase. Después, para inyectarlos basta con crear un constructor de la clase o controlador que incluya los servicios que se necesitan como parámetros. Unity se encarga de instanciar dichos servicios automáticamente. La figura 64 muestra una captura del registro de los servicios en el fichero de configuración. Para cada interfaz se define qué implementación se debe utilizar en toda la aplicación. Esto permite mucha flexibilidad, puesto que para utilizar una implementación distinta de cualquier interfaz basta con cambiarla en este fichero de configuración de Unity.

```
/// <summary>Registers the type mappings with the Unity container.</summary>
/// <param name="container">The unity container to configure.</param>
/// <remarks>There is no need to register concrete types such as controllers or API controllers (unless you want to
/// change the defaults), as Unity allows resolving a concrete type even if it was not previously registered.</remarks>
1 reference | Nubenet Desarrollo, 2 days ago | 1 author, 1 change | 0 exceptions
public static void RegisterTypes(IUnityContainer container)
{
    // NOTE: To load from web.config uncomment the line below. Make sure to add a Microsoft.Practices.Unity.Configuration
    // container.LoadConfiguration();

    container.RegisterType<ApplicationDbContext>(new PerRequestLifetimeManager());
    container.RegisterType<ExternalSource>(new PerRequestLifetimeManager());

    container.RegisterType<ApplicationSignInManager>(new PerRequestLifetimeManager());
    container.RegisterType<ApplicationUserManager>(new PerRequestLifetimeManager());
    container.RegisterType<ApplicationRoleManager>(new PerRequestLifetimeManager());
    container.RegisterType<IAuthenticationManager>(new PerRequestLifetimeManager(),
    new InjectionFactory(c => HttpContext.Current.GetOwinContext().Authentication));

    container.RegisterType<IUserStore<ApplicationUser>, UserStore<ApplicationUser>>(
    new PerRequestLifetimeManager(), new InjectionConstructor(typeof(ApplicationDbContext)));

    container.RegisterType<IRoleStore <IdentityRole, string>, RoleStore<IdentityRole>>(
    new PerRequestLifetimeManager(), new InjectionConstructor(typeof(ApplicationDbContext)));

    // Register unit of work and repositories
    container.RegisterType<IUnitOfWork, UnitOfWork>( new PerRequestLifetimeManager());
    container.RegisterType(typeof(IGenericRepository<>), typeof(GenericRepository<>), new PerRequestLifetimeManager());
    container.RegisterType<InvoiceRepository>(new PerRequestLifetimeManager());

    // Register Services
    container.RegisterType<IUserService<ApplicationUser>, UserService>(new PerRequestLifetimeManager());
    container.RegisterType(typeof(IService<>), typeof(GenericService<>), new PerRequestLifetimeManager());
    container.RegisterType<IService<Bank>, BankService>(new PerRequestLifetimeManager());
    container.RegisterType<IService<CreditorAccount>, CreditorAccountservice>(new PerRequestLifetimeManager());
    container.RegisterType<IService<DirectDebit>, DirectDebitService>(new PerRequestLifetimeManager());

    // Register Queue Manager
    container.RegisterType<IQueueManager, QueueManager>(new PerRequestLifetimeManager());
}
```

*Figura 64. Fichero de configuración de Unity*

Al registrar cada servicio se puede indicar el *LifetimeManager* de dicho servicio. Un *LifetimeManager* indica el tipo de ciclo de vida que tendrán los objetos que se instancien de esa clase. En este proyecto se ha utilizado un *PerRequestLifetimeManager* que crea los objetos para cada petición http y los destruye al enviar la respuesta.

**Repositorio**

En muchas aplicaciones, la lógica de negocio accede a los datos almacenados en centros de datos como bases de datos o servicios web. Acceder a los datos directamente tiene algunas desventajas como código duplicado, un potencial más elevado de provocar errores informáticos o la imposibilidad de probar fácilmente la lógica de negocio aislada de dependencias externas.

Para evitar estos problemas se introduce el patrón de diseño Repositorio (en inglés *Repository*). Este patrón tiene como objetivos principales maximizar la cantidad de código que puede ser probado con automatización y aislar la capa de datos en las pruebas unitarias, acceder a la fuente de datos desde muchos orígenes aplicando reglas de acceso y lógica consistente de forma centralizada y aplicar un modelo de dominio para simplificar la lógica de negocio compleja.

Esto se consigue creando un repositorio que mapea los datos de la fuente de datos con las entidades de la lógica de negocio. El repositorio actúa como intermediario entre la fuente de datos y las capas de lógica de negocio. La figura 65 muestra el diseño de la capa Repositorio.

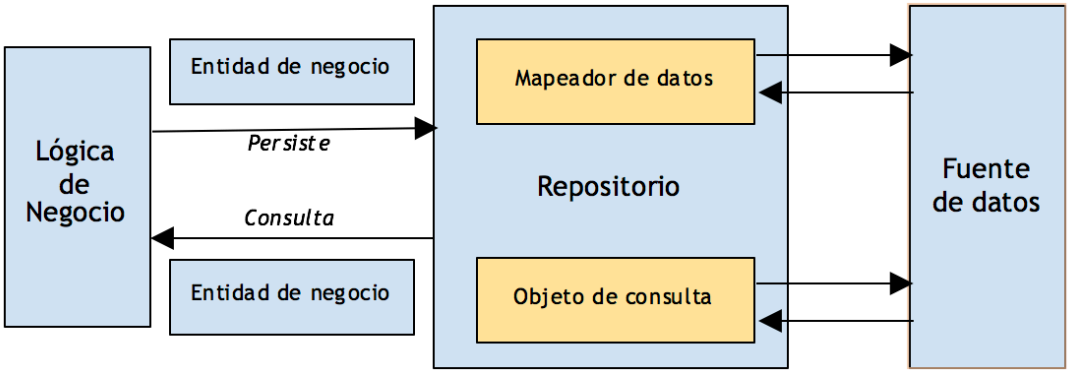



*Figura 65. Diseño del patrón Repositorio [10]*

Con esta estructura es muy sencillo implementar un repositorio genérico parametrizado con la clase de la entidad de negocio. Esto permite realizar cambios en la capa de acceso a los datos de forma global en una sola clase. En este proyecto la clase repositorio implementa la interfaz *IGenericRepository* que se puede observar en la figura 63.

**Unidad de Trabajo**

El objetivo del patrón Unidad de Trabajo (en inglés *Unit of Work* y abreviado como UoW) es persistir a la fuente de datos en una sola operación transaccional todos los cambios aplicados en entidades de negocio relacionadas. Sin una unidad de trabajo, si se modifican dos entidades de negocio que están relacionadas pero son distintas, cada entidad de negocio se persiste (guarda) en una transacción diferente. En ocasiones puede suceder que haya restricciones relacionales entre ambas entidades que impidan que puedan guardarse individualmente puesto que son dependientes entre sí. Este problema lo soluciona el patrón

Unidad de Trabajo que nos permite guardar los cambios aplicados a conjuntos de entidades de distintos tipos de una sola vez. Igual que con el patrón repositorio, el patrón *UoW* nos facilita también la simulación de la capa de datos a la hora de crear pruebas unitarias para validar el código.

La figura 66 muestra la diferencia entre el diseño de acceso y persistencia de los datos con el patrón Unidad de Trabajo y sin él.

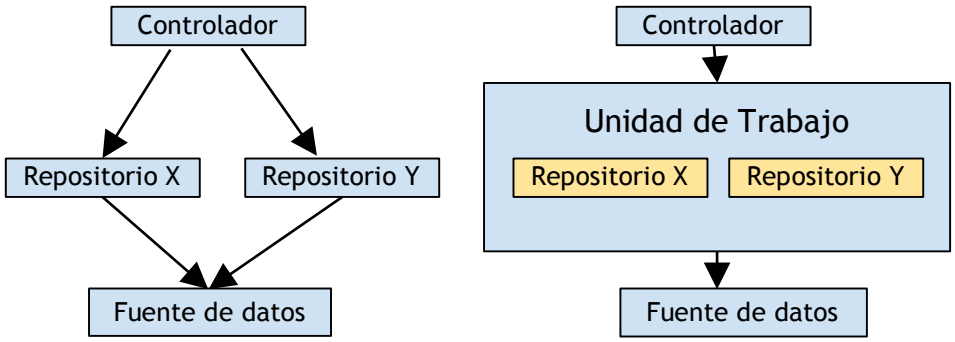



*Figura 66. Diseño del patrón Unidad de Trabajo*

Igual que con el patrón Repositorio, en este proyecto utilizamos una Unidad de Trabajo genérica en los controladores que implementa la interfaz *IUnitOfWork* a la que se le inyectan los repositorios utilizando el patrón explicado previamente de inyección de dependencias.

**Capa de Servicio**

El patrón de diseño Capa de Servicio (*Service Layer* en inglés), se aplica dentro del paradigma de diseño orientado a servicios, que busca organizarlos dentro de un catálogo de servicios organizado por un conjunto de capas lógicas. El objetivo de este patrón es simplificar el mantenimiento del código. Este patrón dicta que los servicios deben ser diseñados para maximizar su reutilización. De ahí la idea de agrupar los servicios en pequeñas capas según su funcionalidad.

En este proyecto no eran necesarios tantos servicios, pero para aprender cómo se desarrolla esta capa, se ha realizado un único catálogo de servicios en la capa del modelo de negocio, entre los controladores y el repositorio. Igual que con el patrón de diseño Repositorio y Unidad de Trabajo, para la mayoría de las entidades se ha creado un servicio genérico que implementa la interfaz *IService* parametrizada. La figura 67 muestra el diseño estándar del patrón Capa de Servicio con respecto a no utilizarlo.

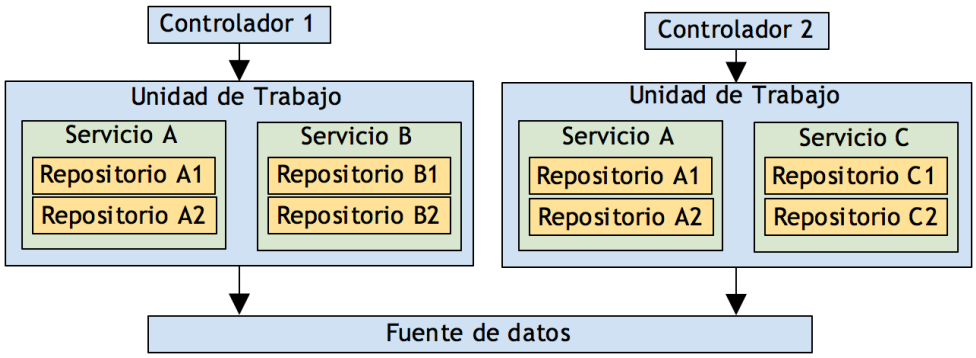



*Figura 67. Diseño del patrón Capa de Servicio*

En este proyecto toda la funcionalidad se ha implementado bien en los controladores o en la capa de servicio. Aunque en el diagrama de clases funcional del capítulo 4 se mostraran las funciones en las clases del dominio, finalmente no se han implementado en la capa de datos sino en las capas de lógica de negocio.

**Equilibrio de carga basado en una cola**

Este es un patrón de diseño específico para la computación en la nube que tiene como objetivo resolver los problemas de disponibilidad y fiabilidad que se pueden producir debido a un volumen alto (pico) de llamadas a servicios de forma inesperada.

Para solucionar este problema, este patrón introduce una cola entre las tareas y los servicios. Tanto las tareas como los servicios se ejecutan de forma asíncrona. Las tareas envían un mensaje que contiene la información que necesita el servicio a una cola. La cola actúa como un buffer, almacenando el mensaje hasta que lo solicita el servicio. Todas las peticiones de distintas tareas, que se pueden generar a ritmos variables, se envían a la misma cola (ver figura 68). De esta forma la carga del servicio es estable y no sufre picos que no puede procesar correctamente.

Para implementar este patrón se introduce el concepto de *roles de servicio*. En Azure se conocen como *Cloud Service Roles* y se definen como una colección de máquinas virtuales gestionadas, con equilibrio de carga, y ofrecidas como PaaS (*Platform-as-a-Service)* que trabajan juntas para ejecutar tareas comunes. Existen dos tipos de roles:

- Rol Web: ejecutan la aplicación en un servidor web y reciben las peticiones. Cada petición recibida crea una instancia del rol.

- Rol Trabajador: se ejecuta en segundo plano como un servicio. Recibe las tareas que envía un rol web y las ejecuta de forma asíncrona. Se establece un número máximo de conexiones concurrentes que puede atender para que la carga esté equilibrada.

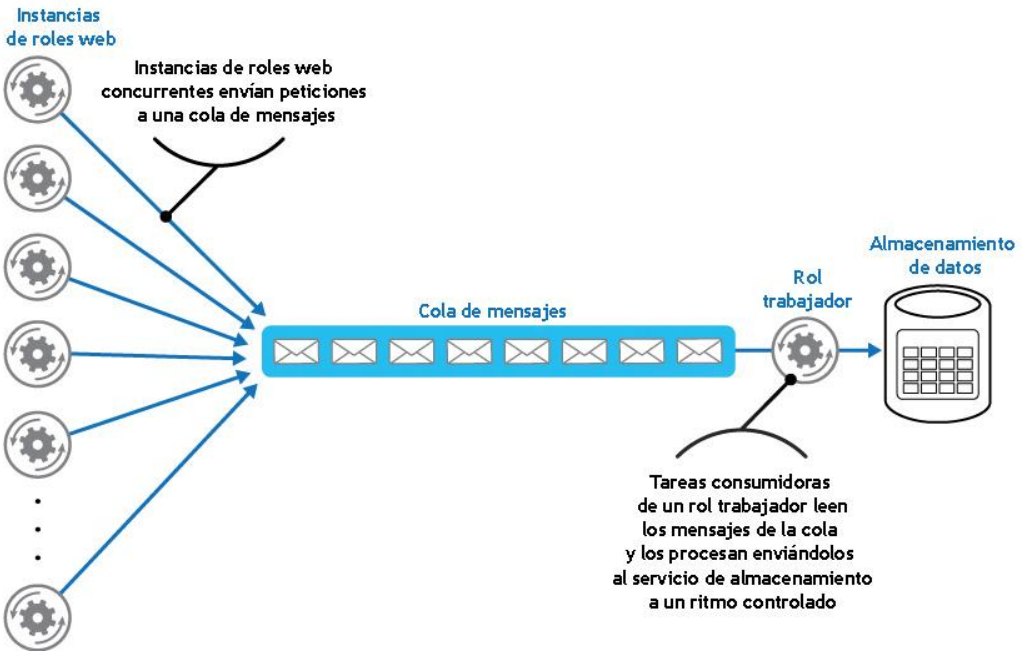



*Figura 68. Técnica de equilibrio de carga basado en una cola de Azure y roles de servicio [11]*

Esta aplicación tiene un rol web y un rol trabajador. El rol web es el proyecto principal Web MVC y el rol trabajador está implementado en el proyecto Worker Role.

Al arrancar la aplicación, ambos roles se ejecutan, estando el rol trabajador en segundo plano. Cada usuario de la aplicación crea una instancia del rol web al acceder a ella. Al ejecutar acciones de actualización o eliminación de entidades, como bancos, cuentas bancarias o remesas, se envían mensajes con las acciones a la cola de Azure. El rol trabajador está constantemente chequeando la cola en segundo plano para ver si hay algún mensaje que tenga que tratar. Cuando recibe un mensaje lo consume y lo procesa.

## 5.2 Estructura del código de la aplicación

El código de la aplicación se ha separado en distintos módulos o librerías (denominadas *proyectos* en Visual Studio) según su interés en el proyecto y la capa a la que pertenecen. Todos los proyectos de una aplicación se agrupan en lo que se denomina una *solución*. Se han creado proyectos separados para la capa de datos, las interfaces, las pruebas, el Web API, la integración con los servicios de Azure y cada una de las partes del sistema que se puede abstraer según lo que representa. Estos proyectos están relacionados entre sí mediante dependencias, que se pueden observar en la figura 69.

Por otra parte, el código de algunos proyectos requiere de librerías externas de ASP.net para implementar ciertas funcionalidades, como la autenticación de usuarios, la compilación de ficheros, etc. Para gestionar las librerías y dependencias de cada proyecto se ha utilizado NuGet.

NuGet es una extensión de Visual Studio que hace más fácil agregar, eliminar y actualizar referencias a librerías y herramientas. Estas librerías en NuGet se llaman paquetes. Cuando se agrega un paquete a un proyecto, NuGet copia los archivos y realiza automáticamente los cambios que sean necesarios en el proyecto, tales como añadir las referencias y cambiar el archivo *app.config* o *web.config*, que son los ficheros de configuración del proyecto. Cuando se elimina una biblioteca, NuGet elimina los archivos y revierte los cambios que hizo en su proyecto. NuGet proporciona una manera rápida y fácil de agregar características a una aplicación existente siempre que estas características están integradas en control de código fuente.

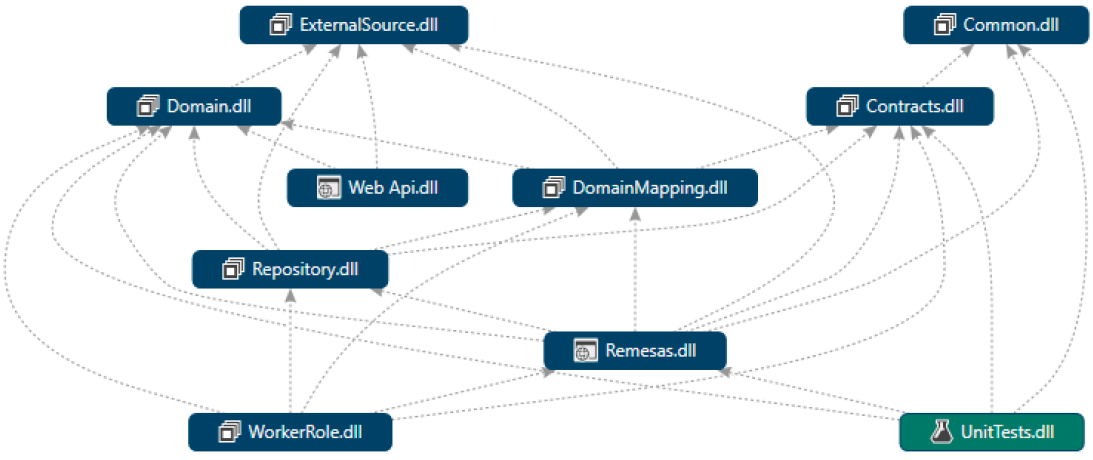



*Figura 69. Arquitectura del proyecto. Imagen generada con Visual Studio.*

**ADO.NET Entity Framework**
Entity Framework es un Object /Relational Mapping (ORM) de Microsoft que permite trabajar con datos relacionales como objetos de dominios. Un mapeo objeto-relacional ORM es una técnica de programación para convertir datos entre el sistema de tipos de un lenguaje de programación orientado a objetos como Java o C# al sistema de tipos de una base de datos relacional, como SQL, por ejemplo. En la práctica, un ORM crea una base de datos orientada a objetos virtual sobre la base de datos relacional. Es decir, se puede acceder a las colecciones de objetos y filtrarlas según el criterio deseado creando consultas utilizando los atributos de los objetos en lugar de los campos de la base de datos, que normalmente sólo pueden almacenar valores escalares como enteros y cadenas.

Sin un ORM, si un objeto tiene un atributo que es una colección de otros objetos, por ejemplo, el desarrollador tendría que convertir esos objetos en grupos de valores simples para almacenarlos en la base de datos y luego volver a convertirlos en objetos al recuperarlos de la base de datos. Este proceso de conversión se realiza automáticamente con un mapeo objeto-relacional, facilitando en gran manera la gestión de datos.

Hay varios ORM que se pueden utilizar en una aplicación de ASP.NET además de Entity Framework, como NHibernate u Open ORM. En este proyecto se ha elegido Entity Framework según los requisitos de la empresa donde se ha desarrollado, por lo que los demás ORM no se han valorado.

Entiy Framework crea una capa de persistencia que se denomina contexto. El contexto contiene la representación de la base de datos virtual. Al llamar a una colección de objetos, o a un atributo de un objeto que sea una referencia a otro objeto, se realiza la llamada a la base de datos y se carga en memoria. Cuando se modifica un objeto, los cambios se realizan solamente en la capa del contexto, hasta que se realiza una llamada al método SaveChanges del contexto que es el que persiste los cambios a la base de datos. En este proyecto esta llamada se realiza en las unidades de trabajo (UoW) tras realizar los cambios solicitados por el usuario.

### 5.2.1 Proyecto Common
Este proyecto contiene únicamente dos enumeraciones, Commands y EntityTypes. Estas enumeraciones se usan en la implementación de la cola para ejecutar servicios en segundo plano. Commands enumera los tipos de acciones que se pueden ejecutar y EntityTypes sobre qué objetos del dominio se pueden ejecutar. Un mensaje de la cola de balanceo de carga contiene una combinación de ambos junto con el objeto sobre el que se realiza la acción.
Este proyecto no depende de ningún otro ni necesita ningún paquete de Nuget.

### 5.2.2 Proyecto Contracts
El proyecto Contracts contiene todas las interfaces que se utilizan en la aplicación, mostradas en el diagrama de la figura 63 Estas interfaces se utilizan para inyectar dependencias en otros proyectos y para facilitar la implementación de pruebas unitarias, simulando los resultados de los métodos de estas interfaces. Este proyecto sólo depende del proyecto Common puesto que la interfaz para la gestión de la cola de balanceo utiliza las enumeraciones explicadas en el apartado anterior.

En cuanto a paquetes de Nuget, este proyecto tiene una interfaz para la implementación del servicio de gestión de usuarios. Puesto que los usuarios extienden la implementación de Entity Framework del modelo de usuarios de ASP.NET Identity, el proyecto necesita los paquetes de Nuget Entity Framework y todos los relacionados con ASP.NET Identity, incluyendo los necesarios para la autenticación de usuarios mediante el protocolo OAuth para plataformas externas.

### 5.2.3 Proyecto Domain

Este proyecto contiene todas las clases de la capa de dominio, que se muestran en el diagrama de clases de la figura 19. Estas clases son el modelo de datos de toda la aplicación y se utilizan en prácticamente todos los demás proyectos. Sólo depende del proyecto External Source para enlazar la clase de remesas con la clase de Facturas que proviene de una fuente externa, que en este caso se mapea en el proyecto External Source.

Para la implementación de esta capa de dominio también se requieren los paquetes de Nuget Entity Framework y ASP.NET Identity, puesto que la clase ApplicationUser para el modelo de usuarios extiende la clase por defecto de Entity Framework, que a su vez requiere ASP.NET Identity.

### 5.2.4 Proyecto Domain Mapping

Este proyecto contiene la configuración del mapeo de Entity Framework de las clases de dominio del proyecto Domain a la base de datos. El proyecto está diseñado para que las clases del dominio sean completamente independientes de su mapeo con la base de datos. Así, el dominio se puede modificar sin que su mapeo con la base de datos se vea alterado.

#### Entity Framework Code-First

Code-First es una técnica de diseño condicionado por el dominio (*Domain Driven Design* en inglés) para elaborar el modelo de datos de una aplicación partiendo de las clases de dominio. Con Code-First el desarrollador no necesita crear la base de datos, ni las tablas para las entidades de dominio. Basta con configurar una conexión con la base de datos, crear las clases para las entidades de dominio y configurar el mapeo de las entidades opcionalmente si quiere establecer restricciones o parámetros distintos a los que Code-First aplica por defecto según su convención de nombres. Una vez esto se ha realizado, al ejecutar la aplicación se crea la base de datos (si aún no existe) y todas las tablas para las entidades de dominio y sus asociaciones.

Hay dos maneras de mapear los atributos de una entidad de dominio con la base de datos. Una forma es utilizando Data Annotations, que consiste en poner anotaciones sobre cada atributo en las clases del dominio indicando las restricciones que debe tener en la base de datos tales como su límite de tamaño, si es clave externa y a qué campo referencia, etc. La otra forma es utilizando Fluent API. Con Fluent API se sobrescribe el método que crea el modelo de la base de datos modificando la configuración de cada entidad. En este proyecto se ha utilizado Fluent API, asignando manualmente el nombre de las tablas que debe tener cada entidad y las relaciones entre entidades, estableciendo las que pueden ser nulas y las que son obligatorias, así como si se deben borrar en cascada o no. La configuración de Fluent API es mucho más flexible que utilizando Data Annotations.

La figura 70 muestra parte de la configuración del mapeo con Fluent API.

```
modelBuilder.Entity<IdentityUserLogin>()
    .ToTable("UserLogins");

// Creditor Account info
modelBuilder.Entity<CreditorAccount>()
    .ToTable("CreditorAccounts")
    .HasRequired(e => e.Bank)
    .WithMany(p => p.CreditorAccounts)
    .HasForeignKey(e => e.BankId)
    .WillCascadeOnDelete(false);

modelBuilder.Entity<CreditorAccount>()
    .Property(e => e.IBAN)
    .IsRequired()
    .HasMaxLength(34)
    .HasColumnAnnotation(IndexAnnotation.AnnotationName,
    new IndexAnnotation(new IndexAttribute("IX_iban") { IsUnique = true }));

// Bank info
modelBuilder.Entity<Bank>()
    .ToTable("Banks")
    .Property(b => b.BankName)
    .IsRequired();

modelBuilder.Entity<Bank>()
    .Property(b => b.BankBIC)
    .IsRequired();

// Direct Debits info
modelBuilder.Entity<DirectDebit>()
    .ToTable("DirectDebits")
    .Property(d => d.MessageId)
    .IsRequired()
    .HasMaxLength(35)
    .HasColumnAnnotation(IndexAnnotation.AnnotationName,
    new IndexAnnotation(new IndexAttribute("IX_messageId") { IsUnique = true }));

modelBuilder.Entity<DirectDebit>()
    .Property(d => d.CreatedDateTime)
    .IsRequired();
```

*Figura 70. Mapeo de entidades de dominio a base de datos con Fluent API*

**Migraciones**

Las migraciones en Code-First son los cambios que se aplican sobre la base de datos cuando cambian las clases de las entidades. Es decir, cuando se añade un nuevo atributo a una entidad, éste también debe añadirse a su correspondiente tabla en la base de datos. Para poder añadir dicho campo se debe ejecutar una consulta de modificación sobre la tabla. Cuando se realizan muchos cambios en el modelo, todos los cambios deben aplicarse en la base de datos. Este conjunto de cambios que debe aplicarse se llama una migración.

Hay dos formas de aplicar las migraciones sobre la base de datos: manualmente, ejecutando un comando que busca los cambios realizados y genera la consulta de actualización y

posteriormente ejecutando dicha consulta con otro comando, o automáticamente, que al lanzar la aplicación se busquen dichos cambios, se genere la consulta y se ejecute sobre la base de datos, todo sin intervención del desarrollador. La forma en la que deben aplicarse las migraciones se define según el tipo de inicializador del contexto de la base de datos. En este proyecto se ha elegido aplicar las migraciones automáticas.

Una característica de las migraciones es que se puede definir si se deben añadir datos o entidades después de cada migración, en caso de que no existan. Esto puede ser necesario cuando se aplican migraciones que deben borrar datos previamente debido a cambios en la estructura o definición de las tablas. Esto se consigue sobrescribiendo el método *Seed()* que se ejecuta después de cada migración.

En este proyecto se utiliza para añadir un usuario administrador siempre que por las migraciones se haya borrado, así se asegura que un usuario siempre tendrá acceso a la aplicación. Los datos del usuario que se han especificado son los siguientes: email: al286273@uji.es, contraseña: al286273.

El proyecto Domain Mapping sólo depende del proyecto Domain, que es el que mapea, y External Source, para indicar en la configuración que se omita el mapeo del atributo Facturas de la clase Remesas, puesto que ese atributo se debe cargar del contexto de la fuente externa de datos.

En cuanto a los paquetes de Nuget, requiere Entity Framework para realizar el mapeo y también los paquetes de Microsoft ASP.NET Identity, puesto que también mapea la clase ApplicationUser que representa la entidad de los usuarios de la aplicación.

### 5.2.5 Proyecto External Source

Este proyecto contiene únicamente las entidades de la fuente de datos externa, es decir las Facturas y los Clientes. Dado que, al contrario que en Code-First, las entidades se crean en otras aplicaciones, no se pueden crear clases para tales entidades, sino que se debe realizar el mapeo contrario, partiendo de la base de datos.

Para esto se utiliza Entity Framework Database-First. El nombre ya lo dice todo: es un tipo de proyecto que se enlaza con una base de datos, pero en este caso las clases del modelo se crean automáticamente a partir de la base de datos. El sistema crea los atributos de las entidades con los nombres de los campos y los tipos de datos asignados en la base de datos. Realizado este paso estas entidades se pueden utilizar en cualquier parte de la aplicación como las propias de la capa de dominio, realizando consultas y filtros con LINQ igual que si fueran Code-First. El inconveniente de este modelo de datos es que si se realiza un cambio del diseño físico de la base de datos de origen el cambio no se detecta automáticamente. En este caso hay que entrar en el modelo de datos creado por Database-First y elegir la opción “Actualizar a partir de base de datos”. Haciendo esto se buscan los cambios producidos en la base de datos y se recrean las entidades.

Este proyecto sólo necesita el paquete de Nuget de Entity Framework para funcionar, y no depende de ningún otro proyecto de la solución.

### 5.2.7 Proyecto Repository

Este proyecto contiene la capa de repositorio explicada en el patrón de diseño Repositorio. Es la capa entre los servicios y el acceso a los datos. Los repositorios utilizan el contexto de Entity Framework para realizar consultas sobre las entidades del dominio, modificaciones y eliminaciones.

Puesto que esta capa trata con ambos contextos, tanto el propio de la aplicación como el externo, y a su vez implementa la interfaz IRepository, este proyecto tiene referencias a los proyectos Contracts, Domain, Domain Mapping y External Source.

Este proyecto depende de bastantes paquetes de Nuget de Microsoft para el tratamiento de las entidades, así como Entity Framework para el mapeo a los objetos y Linq para las consultas sobre la base de datos. No se entra en los detalles de cada paquete puesto que no aportan valor a esta memoria.

### 5.2.8 Proyecto Worker Role

Este proyecto es un proyecto del tipo Rol Trabajador. Como se explica en el apartado del patrón de diseño balanceo de carga mediante una cola, este proyecto contiene únicamente una clase que es la encarga de procesar los mensajes que se envían a la cola desde el proyecto Web MVC, que responde a las peticiones de los usuarios. Cuando un usuario solicita modificar o eliminar una entidad, se envía un mensaje a la cola con la orden y la entidad asociada. La clase WorkerRole en este proyecto trabaja en segundo plano desde que se arranca la aplicación, revisando la cola cada poco tiempo para comprobar si ha recibido mensajes nuevos y procesarlos.

Para procesar los mensajes debe conocer el modelo de la capa de dominio y también la capa de servicios y repositorio, que son quienes finalmente ejecutan las modificaciones sobre la entidad recibida. Es por esta razón que este proyecto depende de los otros proyectos Contracts, Domain, Domain Mapping, Repository y Web MVC.

Los paquetes de Nuget de los que depende son los que se añaden por defecto al crear un proyecto de tipo Rol Trabajador en Visual Studio. Junto con Entity Framework, son varios paquetes de Microsoft necesarios para la ejecución en segundo plano y la comunicación con Azure, que es donde se aloja la cola de balanceo.

### 5.2.9 Proyecto Web MVC

Este es el proyecto principal de la aplicación, el que se ejecuta en un navegador al arrancar la aplicación, junto con Web API en segundo plano, y el servicio de Worker Role. Contiene todos los controladores y las vistas de la aplicación, así como los recursos estáticos (hojas de estilos CSS, ficheros Javascript y extensiones).

El proyecto contiene una carpeta que se llama App_Start. Esta carpeta contiene todas las clases de configuración y parámetros de la aplicación que se ejecutan al arrancarla. Entre dichas clases se encuentra el fichero de configuración de Unity que se ha explicado en el apartado de Inyección de Dependencias, y el fichero de configuración de las identidades de usuario IdentityConfig.cs, que define el gestor de usuarios, el gestor de inicio de sesión de usuarios y el gestor de roles de usuarios. En esta clase también se definen las reglas que

deben cumplir las contraseñas de los usuarios. También se encuentra en esta carpeta el inicializador de los métodos de autenticación, en la clase Startup.Auth.cs. Aquí se define que el método de autenticación debe utilizar una cookie, así como la configuración de cada proveedor de inicio de sesión externo: Microsoft, Twitter, Facebook y Google utilizando el protocolo OAuth2.

El proceso de autenticación mediante el protocolo OAuth2 para una página web A (en adelante PDCnet Remesas) en una página web B (en adelante Plataforma Externa) es el siguiente:

1. PDCnet Remesas se registra como cliente de OAuth2 API en la Plataforma Externa, obteniendo una clave secreta y un ID.

2. Cuando un usuario indica a PDCnet Remesas que acceda a la Plataforma Externa, el usuario es redirigido a la Plataforma Externa, donde confirma que quiere dar permisos a PDCnet Remesas para obtener algunos datos específicos de su cuenta.

3. La Plataforma Externa redirige al usuario de vuelta a PDCnet Remesas junto con un código de autorización.

4. PDCnet Remesas envía dicho código de autorización junto con la clave secreta a la Plataforma Externa, que devuelve un token de seguridad.

5. En este momento PDCnet Remesas realiza una petición de los datos de la cuenta a la Plataforma Externa a nombre del usuario añadiendo el token de seguridad a dicha petición.

Para no guardar claves en los ficheros de código de configuración, las claves secretas de cada proveedor externo se almacenan en el fichero Constants.cs dentro de este proyecto, que además contiene otras constantes necesarias para la creación de remesas. La figura 71 muestra el *flow* de los eventos durante la autenticación con el protocolo Auth2.

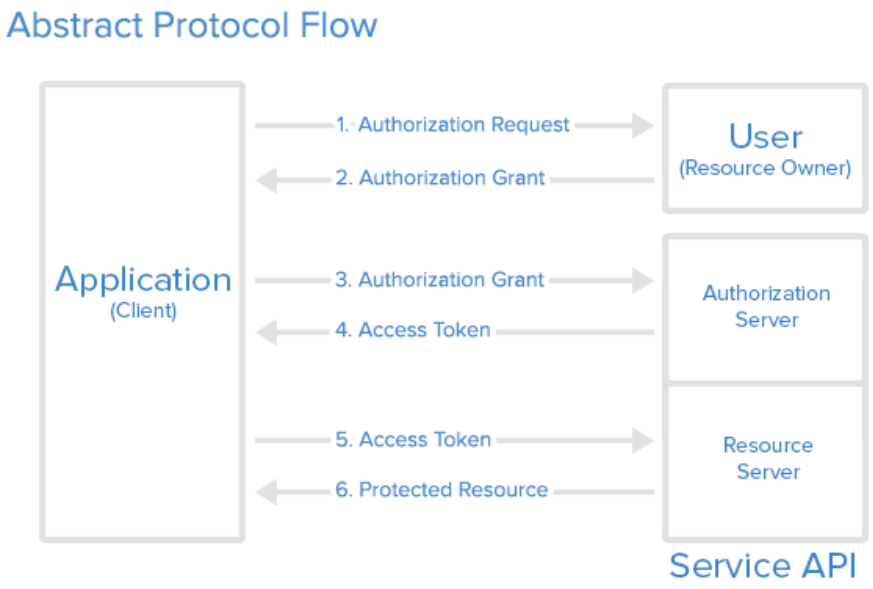



*Figura 71. Pasos del proceso de autenticación con Auth2 [12]*

Los controladores y vistas de la aplicación se han organizado en las denominadas áreas. Un área agrupa todos los ficheros del patrón Modelo-Vista-Controlador que están relacionados siguiente un criterio lógico. Por ejemplo, en una aplicación que gestiona proveedores, clientes y productos, los ficheros de cada uno de estos tres sectores estarían en áreas distintas. Este proyecto tiene poca envergadura, pero con el fin de aprender a utilizarlas y tener el código más ordenado, se han creado tres áreas: una para la gestión de usuarios, otra para los bancos y cuentas bancarias, y la última para las remesas. Excluido de estas áreas queda el controlador y la vista de la página de inicio, cuyas carpetas cuelgan directamente de la carpeta raíz del proyecto Web MVC.

Ahora bien, cada una de estas áreas contiene como mínimo tres subcarpetas: Controllers, Models y Views. Los controladores son quienes reciben las peticiones http y devuelven las vistas. Cada controlador tiene una subcarpeta con su nombre dentro de la carpeta de Views. Dentro de dicha carpeta hay un fichero .cshtml para cada método del controlador que devuelve una vista; para la mayoría de controladores es un fichero para el listado de entidades, el alta de una entidad nueva, la modificación de una entidad y por último la confirmación de borrado de una entidad. La carpeta Models no contiene el modelo de datos propiamente dicho, puesto que ya se ha explicado que el modelo del dominio se guarda en el proyecto Domain. Esta carpeta contiene lo que se conoce en ASP.NET MVC como ViewModels, y conceptualmente representan el modelo de datos, pero sólo contienen aquellos datos que se desean mostrar o modificar en cada vista. Es decir, cuando un controlador recibe una petición para modificar los datos de una entidad, tras cargar el objeto de la base de datos mapea los atributos que se deben poder modificar a un ViewModel, que es el modelo que utiliza la vista para mostrar los datos. Cuando se envía un formulario con los datos modificados, el controlador vuelve a mapear los datos recibidos con un objeto de la clase de dominio correspondiente para luego persistir los cambios. Cualquier campo que no pertenezca el ViewModel que se espera recibir se ignora automáticamente. La ventaja de un ViewModel es que permite un nivel más de abstracción sobre los datos en relación con la vista. Permite mostrar en cada vista solamente los datos que son necesarios para dicha vista y evitar así cargar en el código de la vista información que no es relevante y que el usuario podría modificar malintencionadamente. Esto aporta un nivel de seguridad adicional.

Este proyecto es el principal y por lo tanto depende de los proyectos Common, Contracts, Domain, Domain Mapping, External Source y Repository.

En cuanto a los paquetes de Nuget, este es el proyecto del que más paquetes dependen. Por una parte están todos los relacionados con la autenticación con proveedores externos mediante OAuth y los paquetes de seguridad del protocolo para cada proveedor, creados por Microsoft en el paquete global de Owin. Por otra parte hay muchas librerías de ASP.NET que se requieren para la configuración del arranque de la aplicación y todas las relacionadas con Entity Framework, MVC y Unity, así como los compiladores de LESS y minificadores de Javascript. Por último se referencian los paquetes para el diagnóstico del sistema y los necesarios para la entrada y salida de datos, así como el paquete de Azure Storage que se necesita para utilizar la cola de balanceo de carga.

Las conexiones de la aplicación con las fuentes de datos se definen en el fichero de configuración web.config. Este proyecto utiliza dos fuentes de datos, una para los datos de la propia aplicación y otra para las facturas, con lo cual se definen dos *connection strings*, que incluyen la información del servidor y las claves de autenticación a la base de datos en cuestión.

### 5.2.10 Proyecto "Web API"

Un Web API es un *framework* diseñado para exponer datos a dispositivos o servicios de cualquier tipo mediante peticiones REST. No tiene estado, por lo que las peticiones deben contener toda la información que necesita el API para enviar la respuesta. Normalmente se utiliza para obtener datos con peticiones con la cabecera GET, crear datos con la cabecera POST, modificarlos con la cabecera PUT o borrarlos con la cabecera DELETE.

El único objetivo de este proyecto es generar el fichero XML de una remesa, recibiendo los datos del proyecto Web MVC y devolviendo el fichero XML correspondiente, bien el Core o el B2B, según la petición del cliente. El proyecto depende de los proyectos Domain y External Source puesto que al recibir la remesa en JSON (explicado con más detalle a continuación) la convierte a un objeto Remesa del dominio (con su respectivo atributo Facturas) para facilitar la generación del fichero XML.

En cuanto a los paquetes de Nuget, depende de todos los paquetes de MVC y de Web API de Microsoft. Estos paquetes se añaden a las dependencias del proyecto por defecto al crear un proyecto de tipo Web API en Visual Studio. Crear un proyecto eligiendo este tipo facilita el desarrollo del API. También configura el proyecto automáticamente para que se ejecute en un puerto distinto del proyecto Web MVC. En esta aplicación el proyecto Web MVC se ejecuta en el puerto 44392 y el proyecto Web API se ejecuta en segundo plano en el puerto 53842.

Así, la petición del usuario de descargar un fichero XML se convierte en una petición del controlador de Remesas en el proyecto Web MVC al Web API. La petición es muy sencilla, simplemente envía los datos de la remesa en formato JSON en el cuerpo de la petición (indicando el formato en la cabecera de la petición) y espera una respuesta, que lee como XML y escribe a un fichero, que finalmente envía al navegador.

**JSON (Javascript Object Notation)**

JSON es un formato de texto ligero muy compacto para el intercambio de datos. En este proyecto se utiliza JSON para enviar los datos de las facturas desde el proyecto Web MVC que recibe las peticiones web al proyecto Web Api que se ejecute como una aplicación independiente en segundo plano. Es decir, cuando el usuario hace clic en el botón "Descargar XML", envía una petición al controlador de las Remesas, que a su vez realiza otra petición http interna al proyecto Web API con todos los datos de la remesa en formato JSON. Esto permite enviar gran cantidad de datos en el cuerpo de una petición ocupando poco. Al recibirlo, el Web API analiza los datos en JSON y crea el fichero XML que devuelve al controlador, que a su vez lo devuelve al navegador del usuario.

**XML (Extensible Markup Language)**

XML es un meta-lenguaje que permite definir lenguajes de marcas. Se utiliza para almacenar datos en forma legible. Es muy útil cuando varias aplicaciones deben comunicarse entre sí

o integrar información, como es el caso de este proyecto donde se debe transmitir información entre la aplicación que genera las remesas y el sistema bancario que efectúa los cobros. Las principales ventajas de XML son que es extensible, permitiendo añadir nuevas etiquetas por lo que la flexibilidad para estructurar la información no tiene límites y debido a su simplicidad, es fácil implementar el procesado de un fichero XML en cualquier tipo de aplicación para convertir los datos en información y asociarlos a un contexto específico, lo que ofrece mucha libertad a la hora de estructurar los documentos.

La estructura y definición de los campos del fichero XML que se genera se encuentra en el anexo A.1 de esta memoria. Aunque en el anexo se incluyen todos los campos posibles, muchos son opcionales y en esta primera versión del proyecto sólo se han incluido los campos obligatorios que necesitan los bancos para poder efectuar los cobros de las facturas incluidas.

**Diagrama de componentes del proyecto**

Una vez explicados todos los patrones de diseño que han implementado en el proyecto y las distintas capas, se muestra el diagrama de componentes final agrupado por las 3 capas principales en las que se puede abstraer el diseño de la aplicación: datos, negocio y presentación en la figura 72.

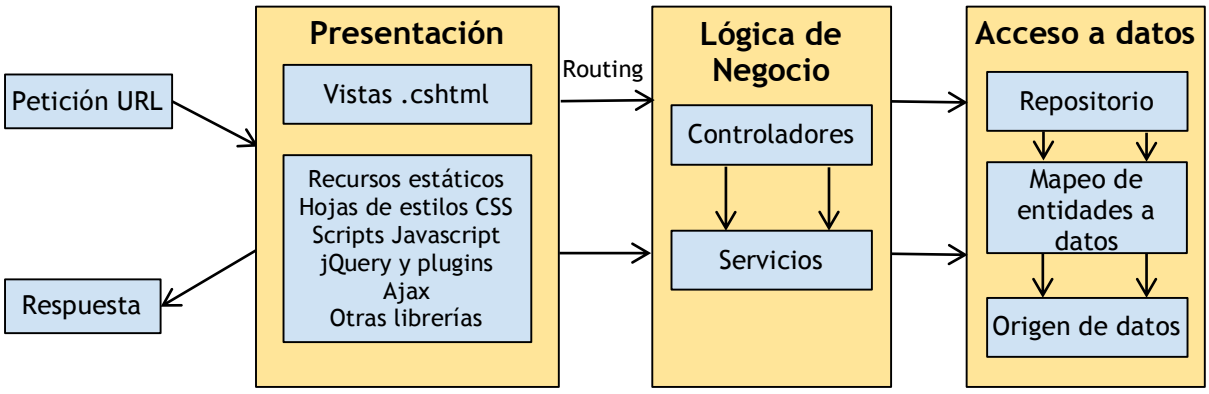



*Figura 72. Diagrama de componentes agrupados por capas*

## 5.3 Capa de presentación

**Lenguajes de la capa de presentación**

Como en la gran mayoría de páginas web, el lenguaje básico utilizado para las interfaces gráficas es HTML (*HyperText Markup Language*). El aspecto de los componentes de las páginas se define mediante hojas de estilo CSS (*Cascading Style Sheets*).

En este proyecto los estilos se han definido utilizando LESS, un preprocesador de CSS que permite definir los estilos utilizando variables y funciones. Esto tiene la ventaja de tener que escribir mucho menos código. El fichero .less se debe compilar después de cada cambio a un fichero .css. Para automatizar la compilación de la hoja de estilos tras cada cambio se ha utilizado un *plugin* de Visual Studio llamado Web Compiler que tras guardar un fichero LESS lo compila automáticamente.

Para ofrecer una mejor interacción con el usuario y crear efectos o animaciones como desplegables o botones para mostrar u ocultar información se ha utilizado Javascript, que

es un lenguaje dinámico, orientado a objetos y basado en prototipos. Es el lenguaje más utilizado y soportado por todos los navegadores para responder a eventos de usuario y modificar el DOM (*Document Object Model*) en tiempo real sin modificar el código HTML original.

**Razor y HTML Helpers**

Razor es una sintaxis de marcaje que permite incrustar código de lado servidor (Visual Basic y C#) en páginas web. El código servidor puede crear contenidos dinámicos en tiempo real, mientras la página web se está enviando al navegador.

Razor está basado en ASP.NET y está diseñado para crear aplicaciones web. Tiene el poder de la sintaxis tradicional de ASP.NET pero es más sencillo de aprender y de utilizar. En este proyecto todas las vistas, que se crean en ficheros .cshtml incluyen sintaxis de Razor.

Un HTML Helper es simplemente un método que devuelve una cadena. La cadena puede representar cualquier tipo de contenido que se desee. Por ejemplo, se puede utilizar un HTML Helper para generar etiquetas HTML estándar como <input> o <img>. También se pueden utilizar para generar código más complejo como una tabla con datos de una base de datos.

Todos estos Helpers se han utilizado en el proyecto para mostrar los formularios. La figura 73 muestra un ejemplo (ampliado para que se pueda ver bien) del código que incluye código servidor de Razor y HTML Helpers.

```
<tbody>
    @foreach (var item in Model)
    {
        <tr>
            <td>@Html.DisplayFor(modelItem => item.BankName)</td>
            <td>@Html.DisplayFor(modelItem => item.BankBIC)</td>
            <td>@Html.DisplayFor(modelItem => item.Accounts.Count)</td>
            <td>
                @if (User.IsInRole("User") || User.IsInRole("Admin"))
                {
                    <div class="btn-group pull-right">
                        <a href="@Url.Action(nameof(BankController.Edit), nameof(BankController).RemoveControl
                            <i class="fa fa-edit"> </i>@Resources.Editar
                        </a>
                        @if (User.IsInRole("Admin") && !item.Accounts.Any())
                        {
                            <button type="button" class="btn btn-default btn-sm dropdown-toggle" data-toggle='
                                <span class="caret"></span>
                            </button>
                            <ul class="dropdown-menu" role="menu">
                                <li>
                                    <a href="@Url.Action(nameof(BankController.Delete), nameof(BankController)
                                        <i class="fa fa-trash"> </i>@Resources.Borrar
                                    </a>
                                </li>
                            </ul>
```

*Figura 73. Código Razor y HTML Helpers*

**Layout**

Para evitar repetir código de las interfaces en cada vista como los encabezados, menús globales, pie de página y demás elementos que son comunes a todas las vistas, se crea una plantilla. En ASP.net esta plantilla se llama Layout y contiene las partes comunes a todas las vistas que la utilizan. El espacio que muestra los contenidos variables se muestra con

una llamada a @RenderBody() (código Razor) en la plantilla. La figura 74 muestra el diseño de la plantilla de este proyecto.

Un detalle de los *Layout* es que permite definir variables y secciones que se pueden asignar en cada vista. Por ejemplo, el título de cada página HTML que se define en el bloque <head> es distinto. Esto se consigue definiendo una variable para el atributo *Title* que se imprime en el <*head*> pero se define en cada vista. Esta característica también se utiliza para cargar ficheros CSS y JS que son específicos para algunas vistas y no se cargan globalmente para no aumentar el tamaño de descarga de la página.

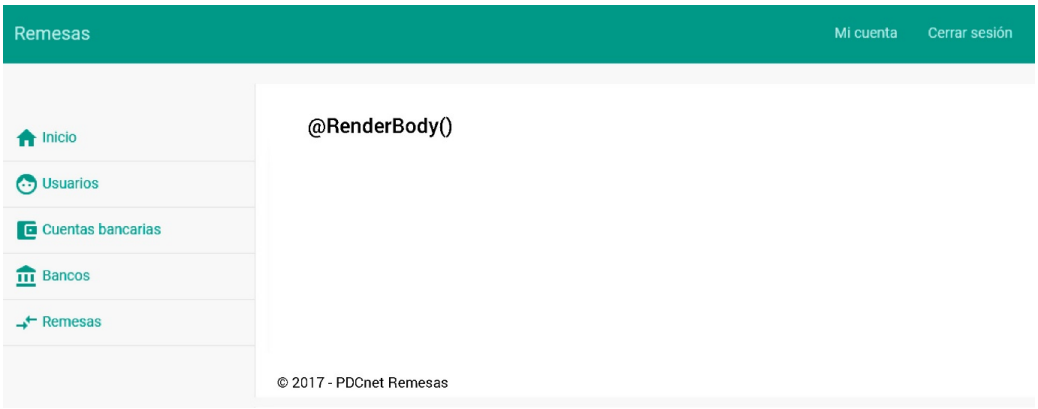



*Figura 74. Layout de las interfaces*

**Bootstrap**

Como se ha explicado previamente, Bootstrap es un *framework responsive* para el diseño de páginas web. Es una librería desarrollada y mantenida por Twitter que incluye muchos componentes con sus respectivas hojas de estilo y Javascript que facilitan la construcción de plantillas muy diferentes de forma muy sencilla.

Una de las ventajas más importantes de Bootstrap es su *grid system*, un sistema de diseño de páginas HTML basado en filas y columnas que da total flexibilidad para distribuir y mover la información en la página con estructuras muy diferentes. Por defecto Bootstrap está diseñado para 4 tamaños: extra pequeño (móviles), pequeños (tableta), mediano (pantallas de baja resolución), grande (pantallas grandes o de alta resolución). También se puede extender si el desarrollador quiere definir nuevos *viewports* (tamaño de pantalla). La figura 75 muestra un ejemplo de una distribución de filas y columnas con tamaños diferentes utilizando Bootstrap.

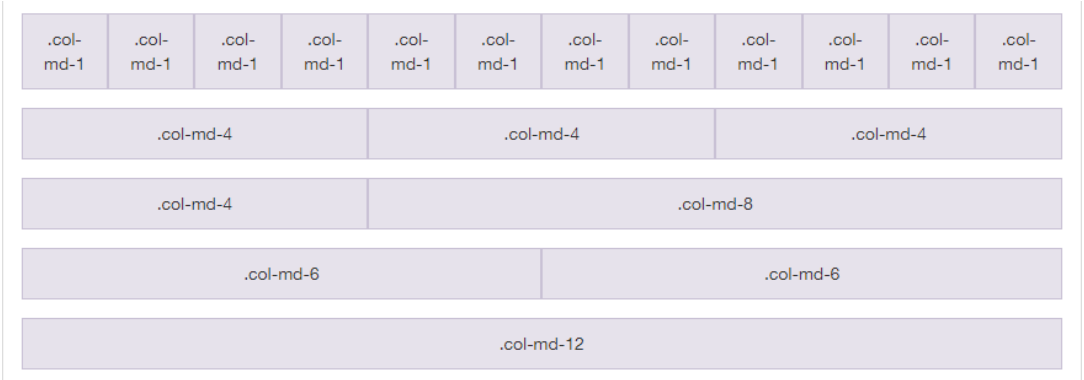



*Figura 75. Grid system de Bootstrap 3*

**Material Design**

Material es una normativa de diseño desarrollada por Google enfocada en la visualización del sistema operativo Android, además de poder aplicarse a la web y cualquier otra plataforma. Según su diseñador:

> *"A diferencia del papel verdadero, nuestro material digital puede ampliar y reformar de forma inteligente. El material tiene superficies físicas y bordes. Las costuras y las sombras que tengan sentido de lo que se puede tocar."*

Se trata de un diseño más limpio, en el que predominan animaciones y transiciones de respuesta, el relleno y los efectos de profundidad tales como la iluminación y las sombras (ver figura 76) [13].

Para este proyecto se ha decidido utilizar esta normativa de diseño para las interfaces, ya que es muy novedoso y agradable a la vista debido a su similitud con las superficies físicas. No obstante, Material incluye también las hojas de estilos y ficheros Javascript para sus componentes que requieren clases CSS distintas que las de Bootstrap, por lo que se ha tenido que utilizar una adaptación de Material para Bootstrap. Así, aplicando las clases CSS de Bootstrap a los componentes se obtiene el resultado de ambas librerías. La adaptación está realizada por un desarrollador que la ofrece como código abierto [14].

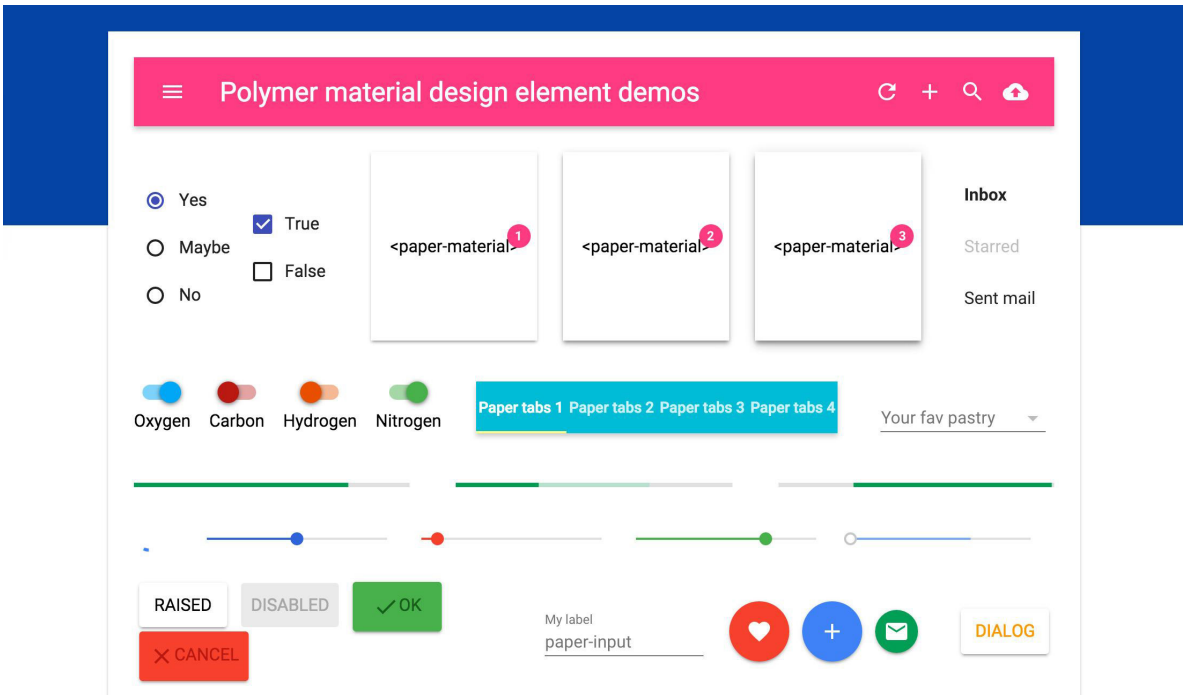



*Figura 76. Guía de diseño de Material Design [15]*

**jQuery y extensiones**

jQuery es la biblioteca multiplataforma de JavaScript más utilizada, y permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web.

Hay muchos *plugins* o extensiones de jQuery que añaden aún más funcionalidad a la que añade la librería por defecto. En este proyecto se han utilizado varias, algunas muy sencillas

como Dropdown JS para modificar el aspecto de los desplegables para que resulten más bonitos de ver, otros como Moment JS para tener un formato de fechas común en los campos de texto donde el usuario debe introducir fechas, y otros más completos como Datatables para mostrar información en tablas.

Datatables es un *plugin* de jQuery muy flexible para mostrar grandes cantidades de datos en tablas. Añade automáticamente controles a la tabla como un buscador, paginación, posibilidad de ordenar las filas por cada columna y un sinfín de opciones más que se pueden añadir. El procesado de los datos (búsqueda, filtrado, etc) se puede realizar tanto en el lado cliente para evitar peticiones al servidor o en el lado servidor, que es más óptimo cuando se están tratando tantos datos que el rendimiento del lado cliente se podría ver afecta. Además, junto con una pequeña extensión, las tablas también son *responsive*, ocultando automáticamente la información de las columnas que no caben según el tamaño de la pantalla y añadiendo un botón para mostrar u ocultar esa información. La figura 77 muestra un ejemplo del uso de DataTables en este proyecto.

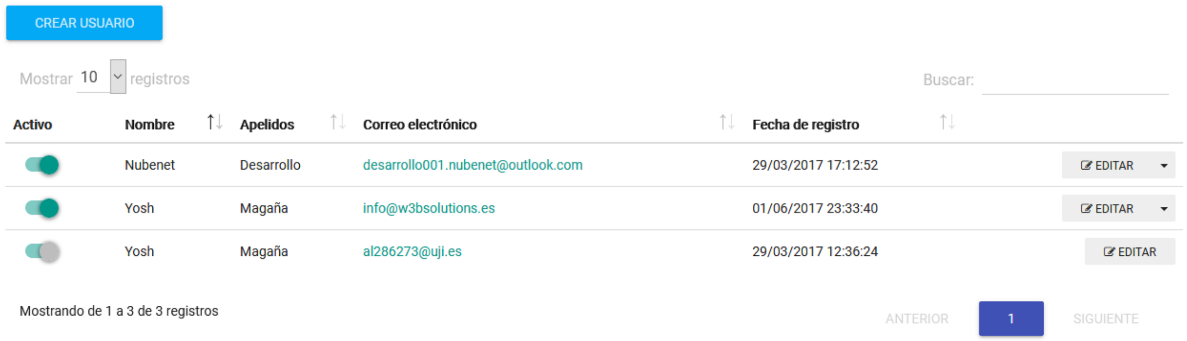



*Figura 77. Tabla de datos que utiliza el plugin de jQuery DataTables*

**YADCF (Yet Another DataTables Columns Filter)**

Para ofrecer filtros avanzados en el formulario de creación y edición de remesa se ha utilizado este *plugin* de jQuery para DataTables que permite crear campos para filtrar columnas con tipos de datos que no vienen por defecto, como calendarios para columnas con fechas para filtrar por intervalos o desplegables con valores predefinidos. La figura 78 muestra los filtros que utilizan esta extensión de DataTables con un calendario desplegado.


*Figura 78. Yet Another DataTables Column Filter*

**AJAX (Asynchronous Javascript And XML)**

AJAX es una técnica de desarrollo web para crear aplicaciones interactivas. Consiste en que el navegador ejecute las acciones mediante una comunicación asíncrona con el servidor en segundo plano, así es posible realizar cambios sin necesidad de recargar la página. Las

ventajas principales de AJAX son que permite mejor interactividad con el usuario, una navegación más sencilla y una aplicación más compacta.

En aplicaciones ASP.NET MVC reduce el código en el lado servidor. Sin AJAX, cuando un formulario avanzado (con listas de selección o desplegables, por ejemplo) se envía a un controlador y éste tiene datos inválidos y se debe mostrar de nuevo el formulario, el controlador debe buscar todos esos datos de nuevo en la base de datos y preparar la vista para mostrarla de nuevo. Con AJAX, basta con devolver el listado de errores de validación y mostrarlos en el formulario; como la página no se refresca todos los datos se mantienen igual. Esto reduce mucho el código en el lado del servidor y las peticiones que debe realizar a la fuente de datos.

La implementación de AJAX es muy sencilla. Para el lado cliente, se ha creado un pequeño *plugin* de jQuery para los formularios que lo utilizan que responde al botón de "enviar formulario" evitando que se recargue la página y enviando los campos del formulario a la URL del atributo *action* del formulario, que indica el controlador que debe procesarlos. El el controlador devuelve un código http de error (4xx o 5xx) se asocian los errores de la respuesta a sus campos correspondientes. Si la respuesta es correcta, se muestra el mensaje de éxito encima del formulario.

En el lado servidor, en una acción que recibe una petición por AJAX, en lugar de devolver la vista completa en caso de error o redirigir a otra acción en caso de éxito, se devuelve una cadena de texto JSON con el resultado, bien con los errores del formulario o un mensaje de éxito.

## 5.4 Verificación y validación

La verificación de un sistema de software consiste en determinar si el producto se está desarrollando correctamente según su especificación inicial. Por otro lado, la validación del software consiste en revisar si se está construyendo el producto correcto, satisfaciendo los requisitos del usuario. En este apartado se explican las técnicas de verificación y validación que se han empleado en este proyecto, tanto para el mantenimiento del código como para la validación de las interfaces gráficas.

**Pruebas unitarias**
Las pruebas unitarias pertenecen a la verificación del sistema, y sirven para comprobar que un método concreto del código de la aplicación funciona correctamente. Realizar pruebas unitarias tiene grandes ventajas, siendo la más destacada asegurar que el código sigue funcionando acorde a los requisitos iniciales tras introducir cambios o ampliar la aplicación con nuevos módulos que utilizan código existente.

En esta aplicación se ha creado un proyecto aparte dentro de la solución llamado Unit Tests que incluye pruebas unitarias de los métodos más importantes de los controladores. Para el controlador de cada entidad, es decir: usuarios, bancos, cuentas bancarias y remesas, se ha creado una clase de pruebas unitarias que verifica sus métodos de listado, inserción, modificación y eliminación para comprobar que efectivamente los cambios se realizan en la fuente de datos.

En el apartado de inyección de dependencias se mencionó que las interfaces también son muy útiles a la hora de implementar pruebas unitarias en un sistema de software. En ese sentido, para ejecutar las pruebas unitarias se ha creado una nueva implementación de la clase repositorio. El fin de una prueba unitaria es asegurar que un método hace lo que debe hacer, por lo que debe ser independiente de la implementación del método y de la fuente de datos. Las pruebas unitarias no deben introducir o modificar datos en el origen de datos de la aplicación. Esto lo conseguimos creando una implementación de la interfaz *IRepository* con una fuente de datos en memoria. Esta fuente de datos puede utilizar una estructura completamente distinta a la real, en el caso de este proyecto utiliza listas de objetos. Con esto conseguimos no alterar la base de datos real y al crear la base de datos en memoria conseguimos que las pruebas se ejecuten más rápido, puesto que se ahorra el tiempo de ejecución de cada consulta sobre una base de datos real.

Resumiendo, las pruebas unitarias implementadas comprueban que se listan, crean, modifican y eliminan objetos correctamente utilizando los controladores reales de la aplicación, pero inyectando un repositorio en memoria con datos ficticios. Este repositorio ejecuta las acciones directamente sin utilizar la cola de balanceo de carga de Azure. En una aplicación de más envergadura también se deberían implementar pruebas unitarias para asegurar que la cola funciona correctamente y no se pierden mensajes por el camino.

**Validación de requisitos**

La validación de los requisitos consiste en asegurar que el producto cumple con las expectativas del cliente y todos los requisitos acordados inicialmente se han implementado. Una técnica de validación de requisitos consiste en realizar una lista de pruebas *checklist* para cada caso de uso, asegurando que la prueba tiene éxito, o en caso contrario realizando los cambios necesarios para que la prueba pase. Las listas de validación para cada caso de uso que se han comprobado con éxito se presentan a continuación.

1. **Iniciar sesión / registrarse**
   - 1.1. Un usuario puede registrarse en la aplicación introduciendo sus datos manualmente.
   - 1.2. Al registrarse manualmente, se comprueba que el usuario introduce todos los datos obligatorios.
   - 1.3. Al registrarse manualmente, se comprueba que el email introducido por el usuario no está ya registrado.
   - 1.4. Un usuario puede registrarse en la aplicación a través de Facebook.
   - 1.5. Un usuario puede registrarse en la aplicación a través de Twitter.
   - 1.6. Un usuario puede registrarse en la aplicación a través de LinkedIn.
   - 1.7. Un usuario puede registrarse en la aplicación con su cuenta de Microsoft.
   - 1.8. Un usuario puede registrarse en la aplicación con su cuenta de Google.
   - 1.9. Tras registrarse, un usuario puede iniciar sesión en la aplicación.
   - 1.10. Tras iniciar sesión, un usuario puede modificar los datos de su perfil y su contraseña.
   - 1.11. Un usuario registrado puede asociar cuentas externas a su cuenta.
   - 1.12. Un usuario registrado con una cuenta externa puede eliminar la asociación con la plataforma externa.

**2. Mantener usuarios**

2.1. Un usuario administrador puede dar de alta nuevos usuarios en la aplicación, eligiendo el rol que desea asociar al usuario.
2.2. Un administrador puede modificar los datos de usuarios.
2.3. Un administrador puede bloquear el acceso de usuarios a la aplicación.
2.4. Un administrador puede eliminar usuarios de la aplicación.
2.5. Durante el alta y modificación de un usuario se exige que se rellenen los campos obligatorios.
2.6. Durante el alta y modificación de un usuario se valida que los datos cumplen con el formato especificado.
2.7. Durante el alta de un nuevo usuario se valida que el usuario no está ya registrado.
2.8. Un usuario no administrador no puede listar, añadir, modificar ni eliminar usuarios.

**3. Mantener entidades bancarias**

3.1. Un usuario con rol de invitado puede listar entidades bancarias, pero no puede crear nuevas ni editar o eliminar las existentes.
3.2. Un usuario con rol de usuario puede, además de listar entidades bancarias, crear nuevas.
3.3. Un usuario con rol de usuario puede modificar bancos.
3.4. Un usuario con rol de usuario no puede eliminar bancos.
3.5. Un usuario administrador puede, además de listar, crear y editar entidades bancarias, borrar las existentes.
3.6. Durante el alta y modificación de una entidad bancaria se valida que campos obligatorios se completan y los datos cumplen con el formato especificado.

**4. Mantener cuentas bancarias**

4.1. Un usuario con rol de invitado puede listar cuentas bancarias, pero no puede crear nuevas ni editar o eliminar las existentes.
4.2. Un usuario con rol de usuario puede, además de listar cuentas bancarias, crear nuevas.
4.3. Un usuario con rol de usuario puede modificar cuentas bancarias.
4.4. Un usuario con rol de usuario no puede eliminar cuentas bancarias.
4.5. Un usuario administrador puede, además de listar, crear y editar cuentas bancarias, borrar las existentes.
4.6. Durante el alta y modificación de una cuenta bancaria se valida que campos obligatorios se completan.
4.7. Durante el alta y modificación de una cuenta bancaria se comprueba que el IBAN cumple el formato especificado.
4.8. Durante el alta y modificación de una cuenta bancaria se comprueba que el IBAN introducido no está ya registrado.

**5. Mantener remesas**

5.1. Un usuario con rol de invitado puede listar las remesas ya creadas, pero no puede crear nuevas, ni editar o eliminar las existentes.
5.2. Un usuario con rol de invitado puede descargar los ficheros XML Core y B2B para las remesas ya creadas.

5.3. Un usuario con rol de usuario puede listar, crear, editar y descargar los ficheros XML de remesas, pero no eliminar.
5.4. Un usuario administrador puede realizar lo mismo que un usuario con rol de registrado, pero además puede eliminar remesas.
5.5. Durante el alta y modificación de una remesa se comprueba que los datos obligatorios se cumplimentan.
5.6. Durante el alta y modificación de una remesa se comprueba que la remesa contiene al menos una factura.
5.7. Si una remesa tiene activada la opción "Agrupar facturas por cliente" efectivamente se agrupan en el XML generado y los totales se calculan correctamente.
5.8. El fichero XML Core que se descarga se valida correctamente según el XML schema.
5.9. El fichero XML B2B que se descarga se valida correctamente según el XML schema.

El requisito de que la aplicación debe ser adaptable a dispositivos móviles se valida en el apartado a continuación de validación de interfaces.

**Validación y evaluación de interfaces**

**Validación de interfaces**
Todos los formularios de las interfaces tienen dos niveles de validación: el primero está relacionado las reglas que debe cumplir cada campo y utiliza el *plugin* de jQuery Validate. En los *View Models* de cada formulario se especifican las condiciones que debe cumplir cada campo utilizando anotaciones. Por ejemplo, la anotación [Required] encima de un campo indica que el campo es obligatorio y el formulario no puede enviarse si el campo está en blanco. Al generarse la vista, se incluye metadata con cada campo que contiene el mensaje de error en caso de que no cumpla alguna de las reglas definidas. Al enviarse los datos, si el *View Model* contiene errores, el controlador devuelve un estado de error y los mensajes de error de los campos afectados se muestran.

El segundo nivel de validación de los campos pertenece a la lógica de negocio. Por poner un ejemplo, el campo email de un formulario de alta de usuario tiene estos dos niveles de validación, aunque muchos otros campos no requieren del segundo: el primero requiere que el texto introducido tenga el formato de un email, el segundo requiere que el email no exista ya en la base de datos. Cuando un controlador recibe los datos de un formulario que ha pasado el primer nivel de validación, realiza el segundo nivel en los campos que lo requieren, y si algún dato está mal, asocia un mensaje de error personalizado con el campo en cuestión y devuelve el código de error a la vista nuevamente, junto con la lista de campos no válidos y sus mensajes de error. En este momento se activa de nuevo el plugin de jQuery Validate y muestra los nuevos mensajes de error junto a cada campo.

En el proyecto se ha realizado el segundo nivel de validación con los campos de email en el alta y modificación de usuarios y con el campo IBAN en el alta y modificación de cuentas bancarias, puesto que también debe cumplir un patrón y también debe ser único en la base de datos. La figura 79 muestra una interfaz con el mensaje de error. Todos los formularios se han probado para asegurar que la validación se realiza correctamente.

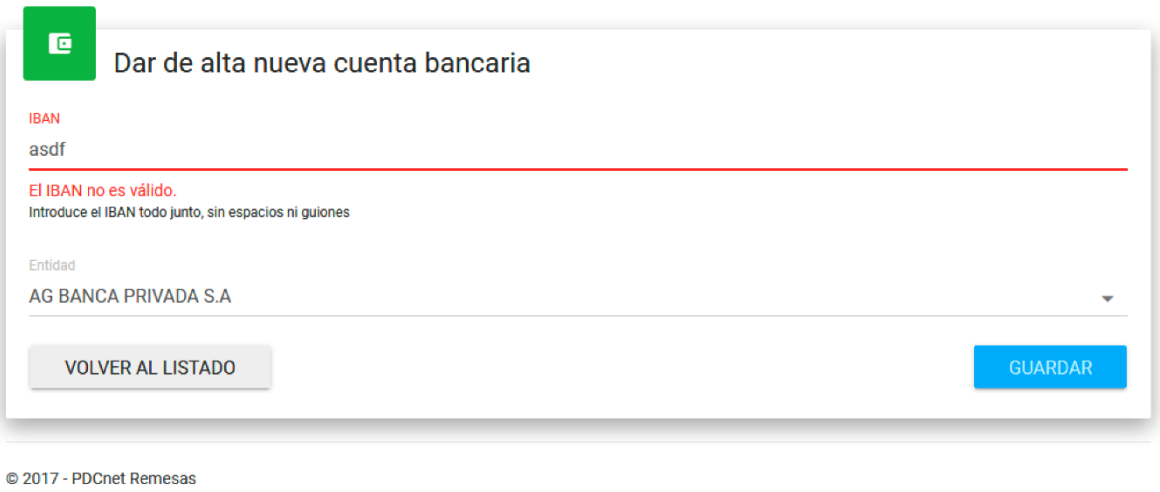



*Figura 79. Validación de campo IBAN en formulario de alta de cuenta bancaria*

En cuanto al *responsiveness* de la aplicación, se ha verificado que todas las páginas del sistema se adaptan correctamente al tamaño de la pantalla sin quitar información (en algunos casos se oculta porque no cabe de manera bonita y se añade un botón para mostrar los datos ocultos) y el menú desplegable responde a la interacción con el usuario.

**Evaluación de interfaces**

Para este apartado se ha realizado una evaluación de las interfaces de la aplicación. Primero se ha evaluado el sistema de manera global y después, de una forma más concreta, la vista de creación de una remesa, puesto que es el fin principal de la aplicación. En cada apartado se ha realizado un test heurístico y un test de aceptación de usuarios, el cual se ha cumplimentado por otro estudiante, para obtener el resultado de una tercera persona ajena al desarrollo de la aplicación y del ámbito de facturación y remesas.

**Test de aceptación global de usuarios sobre el sitio web**

La tabla 23 muestra el test de aceptación global, teniendo en cuenta que 1 significa totalmente en desacuerdo y 5 totalmente de acuerdo.

| Cuestionario | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1. El sistema ha sido fácil de usar. | | | | x | |
| 2. Puedo usar el sistema por mi cuenta, sin que un técnico me ayude. | | | | x | |
| 3. He observado varias funcionalidades del sistema bien integradas. | | | | | x |
| 4. Pienso que otras personas pueden aprender a usar este sistema | | | | x | |
| 5. El sistema responde de manera fluida ante mis acciones. | | | | x | |
| 6. El uso del sistema no requiere de conocimientos técnicos previos. | | | | x | |
| 7. Me gustaría utilizar sistemas como este con más frecuencia. | | | x | | |
| 8. El sistema muestra mensajes de ayuda adecuados. | | | x | | |
| 9. En general, estoy satisfecho/a con el uso del sistema. | | | | x | |

*Tabla 23. Test de aceptación global sobre el sitio web*

**Test heurístico global sobre el sitio web**

| **Generales** | **Puntos** |
|---|---|
| ¿Cuáles son los objetivos del sitio web? ¿Son concretos y bien definidos? | x |
| ¿Los contenidos y servicios que ofrece se corresponden con estos objetivos? | x |
| ¿Tiene una URL correcta, clara y fácil de recordar? ¿Y las URL de sus páginas internas? | x |
| ¿La estructura general del sitio web está orientada al usuario? | x |
| ¿Es coherente el diseño general del sitio web? | x |
| ¿Es reconocible el diseño general del sitio web? | x |
| **TOTAL** | **6 / 6** |
| **Identidad e información** | **Puntos** |
| ¿Se muestra claramente la identidad de la empresa a través de todas las páginas? | x |
| El logotipo, ¿es significativo, identificable y suficientemente visible? | |
| ¿La empresa posee un eslogan? ¿Se expresa realmente qué es la empresa y qué | |
| ¿Se ofrece algún enlace con información sobre la empresa? | |
| ¿Se proporciona información sobre la protección de datos de carácter personal de los | |
| **TOTAL** | **1 / 5** |
| **Lenguaje y redacción** | **Puntos** |
| ¿El sitio web habla el mismo lenguaje que sus usuarios? | x |
| ¿Emplea un lenguaje claro y conciso? | x |
| ¿Es amigable, familiar y cercano? | x |
| ¿1 párrafo = 1 idea? | x |
| **TOTAL** | **4 / 4** |
| **Rotulado** | **Puntos** |
| Los rótulos, ¿son significativos? | x |
| ¿Usa rótulos estándar? | x |
| ¿Usa un único sistema de organización, bien definido y claro? | x |
| El título de las páginas, ¿es correcto?, ¿ha sido planificado? | x |
| **TOTAL** | **4 / 4** |
| **Estructura y navegación** | **Puntos** |
| La estructura de organización y navegación, ¿es adecuada? | x |
| ¿Los enlaces son fácilmente reconocibles como tales? | x |

| | |
|---|---|
| ¿Es predecible la respuesta del sistema antes de hacer clic sobre un enlace? | x |
| ¿Se ha controlado que no haya enlaces que no llevan a ningún sitio? | x |
| ¿Se ha evitado la redundancia de enlaces? | x |
| ¿El logotipo de la página enlaza con la página de inicio? | |
| **TOTAL** | **5 / 6** |
| **PUNTUACIÓN GLOBAL** | **20 / 25** |

*Tabla 24. Test heurístico global de las interfaces*

En el apartado de Identidad e información vemos que varios puntos no están cumplimentados, como la presencia de un logotipo o información de la empresa. En este caso la empresa no lo exigió, por ello no se incluye.

**Test de aceptación de usuarios sobre la interfaz "Crear una remesa"**
La tabla 25 muestra el cuestionario, teniendo en cuenta que 1 significa totalmente en desacuerdo y 5 totalmente de acuerdo.

| Cuestionario | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1. Ha sido sencillo crear la remesa. | | | | x | |
| 2. He podido generar la remesa sin que un técnico me ayude. | | | | | x |
| 3. Los calendarios y los desplegables están bien integrados con la estética de | | | | x | |
| 4. Pienso que otras personas pueden aprender a crear remesas rápidamente. | | | | x | |
| 5. La interfaz responde de manera fluida ante mis acciones. | | | | x | |
| 6. El uso de la interfaz no requiere de conocimientos técnicos previos. | | | | x | |
| 7. Me gustaría encontrar interfaces como esta con más frecuencia. | | | | | x |
| 8. La interfaz muestra mensajes de ayuda adecuados. | | | x | | |
| 9. En general, estoy satisfecho/a con la forma en la cual está diseñado el | | | | x | |

*Tabla 25. Test de aceptación de la interfaz "Crear una remesa"*

**Test heurístico sobre la interfaz "Crear una remesa"**

| Generales | Puntos |
|---|---|
| ¿El objetivo de la interfaz es concreto y bien definido? | x |
| ¿Los contenidos y servicios que ofrece se corresponden con estos objetivos? | x |
| ¿La URL de la página es correcta, clara y fácil de recordar? | x |
| ¿El título de la página es adecuado? | x |
| ¿La estructura de la página está orientada al usuario? | x |
| ¿Es coherente el diseño de la página con el del resto del sitio web? | x |

| | |
|---|---|
| **TOTAL** | **6 / 6** |
| **Lenguaje y redacción** | **Puntos** |
| ¿La página está escrita en el mismo lenguaje que sus usuarios? | x |
| ¿Emplea un lenguaje claro y conciso? | x |
| ¿Es amigable, familiar y cercano? | x |
| **TOTAL** | **3 / 3** |
| **Estructura y navegación** | **Puntos** |
| La estructura de organización y navegación, ¿Es adecuada? | x |
| ¿Los enlaces son fácilmente reconocibles como tales? | x |
| ¿Es predecible la respuesta del sistema antes de hacer clic sobre un enlace? | x |
| ¿Se ha controlado que no haya enlaces que no llevan a ningún sitio? | x |
| ¿Se ha evitado la redundancia de enlaces? | x |
| **TOTAL** | **5 / 5** |
| **PUNTUACIÓN GLOBAL** | **14 / 14** |

*Tabla 26. Test heurístico de la interfaz "Crear una remesa"*

### Validación de los ficheros XML generados

Para validar los ficheros XML generados se ha utilizado el validador gratuito de MobileFish, eligiendo el esquema XML implementado, es decir, pain.008.001.002 y subiendo el fichero generado a su web, como se muestra en la figura 80, donde también se puede observar el resultado de la validación.

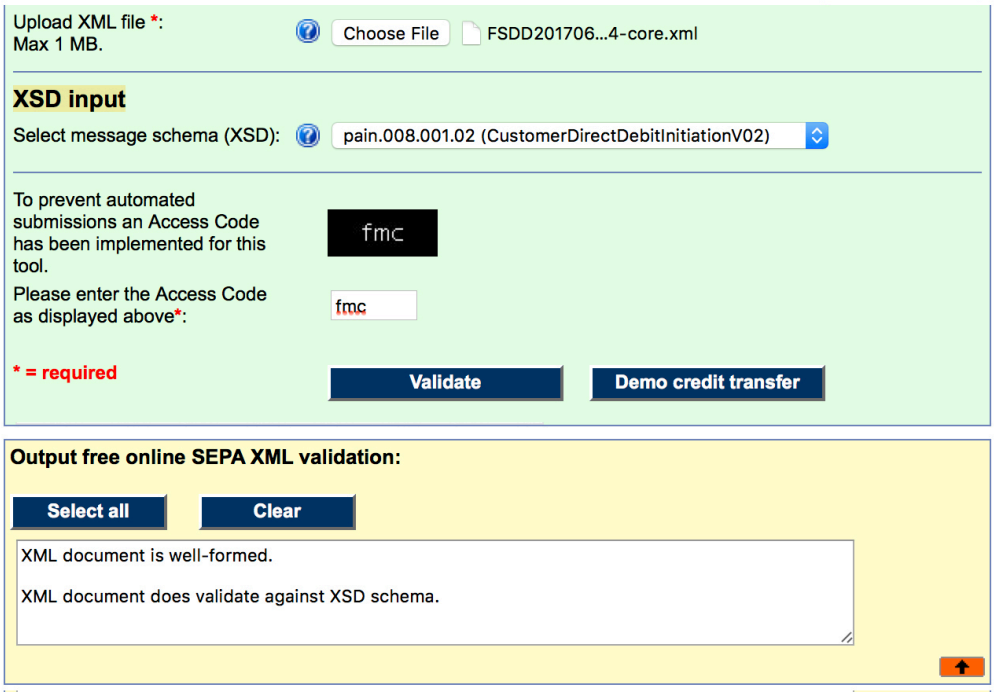



*Figura 80. Resultado de comprobación de validación de un fichero XML Core generado a través de la aplicación [16]*

Ha quedado pendiente realizar las pruebas con algún validador específico de un banco (no se ha encontrado ninguno *online*) o bien generar una remesa con datos reales y enviar el fichero XML al banco para asegurar que cumple los requisitos y no falta ningún campo, pero pasando este esta prueba con éxito también se debería validar correctamente en cualquier entidad bancaria. Debido a que, a fecha de la redacción de esta memoria el proyecto no se ha lanzado aún en producción, no se ha podido realizar una prueba con datos reales.

## 5.5 Documentación

En este apartado se explica la documentación del proyecto, tanto durante el desarrollo como al finalizarlo.

**Manual de desarrollo**
Para el desarrollo y formación del estudiante se han utilizado principalmente recursos *online* como la documentación oficial de Microsoft para ASP.NET y otros tutoriales básicos y gratuitos ofrecidos por páginas web de formación en desarrollo de aplicaciones web. Para conocer e implementar algunos de los patrones de diseño se ha utilizado un libro de Microsoft proporcionado por la empresa llamado Patrones de Diseño para la Computación en la Nube.

En cuanto al manual de desarrollo del propio proyecto, no se ha considerado necesario realizar uno ya que esta memoria servirá de referencia como documentación del proyecto, puesto que contiene toda la información necesaria que pueda necesitar un futuro desarrollador para realizar cambios en la aplicación.

**Manual de usuario**
Aunque la aplicación es sencilla e intuitiva, se ha decidido escribir un manual de usuario básico para usuarios nuevos con todos los pasos a seguir para poder generar una remesa, que es el objetivo final. Este manual se incluye en el anexo A.2.

# Capítulo 6. Resultados y conclusiones

En este capítulo se exponen los resultados del desarrollo del proyecto y las conclusiones finales.

## 6.1 Resultados del proyecto

El resultado del proyecto ha sido muy positivo. Se ha de decir que se han cumplido todos los objetivos de la aplicación, incluso se han superado los requisitos iniciales puesto que no incluía la opción de generar también el fichero XML B2B, que finalmente también se ha implementado.

Un detalle que retrasó un poco la planificación inicial fue que el supervisor solicitó varios cambios en la implementación que no se esperaban y no figuraban en los requisitos técnicos iniciales como utilizar algunos patrones de diseño. Esto está explicado en el apartado de seguimiento del proyecto en el capítulo 3.

Durante la estancia en la empresa donde se han desarrollado las prácticas no se ha experimentado ningún problema destacable y la colaboración del supervisor fue constante. También facilitó un libro en papel de patrones de diseño para la computación en la nube que fue muy útil para implementar ciertos patrones (referenciado en el apartado de Bibliografía).

En cuanto a la puesta en marcha de la aplicación, la empresa debe realizar algunos cambios para integrarla con otras de sus aplicaciones antes de poder ofrecerla a sus clientes, por lo que queda en sus manos.

## 6.2 Evaluación tecnológica

Dado que la empresa ya trabajaba con Azure para sus otras aplicaciones, utilizarlo para esta aplicación desarrollándola en ASP.NET me ha parecido una decisión acertada. Se puede decir que el resultado es una aplicación muy robusta que está diseñada para poder alojarse en la nube con servidores distribuidos. La aplicación se podría haber implementado más fácilmente con otros lenguajes de lado servidor como PHP, pero el hecho de utilizar ASP.NET MVC aporta beneficios como un código orientado a objetos que es más fácil de mantener y extender. También utilizar Bootstrap como *framework responsive* ha facilitado mucho el diseño de las vistas para dispositivos pequeños.

La envergadura del proyecto no es suficientemente grande como para que las tecnologías empleadas pudieran presentar limitaciones o deficiencias. No se ha experimentado ningún problema con respecto a los lenguajes o herramientas utilizadas.

## 6.3 Conclusiones personales

Cuando tuve que afrontar el periodo de estancia en prácticas por un lado pensé “por fin llegó el momento, ya sólo me queda esto y acabo la carrera, ¡qué ganas!”, pero por otro lado llegaba el temido momento: “a ver cómo hago para vivir 3 meses, teniendo que dejar

mi trabajo actual para trabajar gratis 8 horas al día y pagar el alquiler, facturas …". Llevo viviendo independiente y trabajando como informático y desarrollador web en una empresa de Castellón a media jornada desde los 18 años, compaginándolo con los estudios, primero un ciclo de formación profesional de grado superior y luego el grado en la UJI, y dejarlo para realizar las prácticas suponía un poco de trastorno personal, pero a la vez un desafío.

Llegado el momento, decidí que al menos el proyecto a desarrollar fuera en un ámbito nuevo para mí para el que tuviera que aprender cuanto más mejor. Cuando vi este proyecto en el listado de propuestas lo tuve claro, puesto que incluía varios desafíos. El principal es que, aunque la idea del proyecto es relativamente sencilla, abarca muchas tecnologías y lenguajes distintos. A lo largo del grado no se enseña C# ni se utiliza ASP.net, ni siquiera el IDE Visual Studio, con lo cual era un reto aprender un nuevo lenguaje y utilizar un entorno de desarrollo para programar una aplicación completa en tan poco tiempo. Otro tema nuevo era descubrir cómo funciona Azure y cómo se integra en aplicaciones, puesto que tampoco se enseña durante el grado, así como aprender sobre nuevos patrones de diseño y conectar una aplicación con una Web API dentro del mismo proyecto. Y todo esto adaptándose a un nuevo entorno de trabajo con nuevos compañeros y en otra ubicación.

Ahora, una vez terminado, he de decir que elegir este proyecto fue una de las mejores decisiones de mi vida. Además de aprender, al menos lo básico, de ASP.NET y C# para poder desarrollar aplicaciones web completas, eso fue justamente lo que necesitaba para convencer a los entrevistadores de mi nuevo puesto en BMW en Múnich, donde parte de mi trabajo consiste en desarrollar aplicaciones en ASP.NET. En tan solo dos meses aquí he ampliado mis conocimientos en gran medida, y todo esto es gracias a este proyecto donde inicié mi andadura con estos lenguajes y librerías.

El resultado del proyecto y la estancia en prácticas es muy gratificante, estoy satisfecho con la aplicación y mi supervisor también, que es aún más importante. Como desarrollador lo que más me satisface es ver a un cliente contento al terminar, y creo que con este proyecto se ha conseguido.

Por último, muy satisfecho de terminar la carrera. Obtener un título universitario fue un reto que me planteé al terminar el FP y a día de hoy me cuesta creer que lo tengo ya tan cerca, pero a la vez me anima a seguir estudiando para formarme más. En ese sentido quiero agradecer a todos mis profesores a lo largo de la carrera por formarme, y especialmente a mi novia junto a mi familia, destacando a mis tíos, por apoyarme desde el primer día y permitir hacer de mi sueño una realidad.

# 7. Referencias y Bibliografía

En el texto de esta memoria se han referenciado las siguientes páginas web todas ellas fácilmente localizables con un buscador web:

[1] «netPrevencion. Software de prevención (PRL) de Nubenet.es». [En línea]. Disponible en: http://netprevencion-lp.azurewebsites.net/. [Accedido: 04-jul-2017].
[2] «Diseño web adaptable - Wikipedia, la enciclopedia libre». [En línea]. Disponible en: https://es.wikipedia.org/wiki/Dise%C3%B1o_web_adaptable. [Accedido: 24-jul-2017].
[3] «ASP.NET MVC Framework - Wikipedia, la enciclopedia libre». [En línea]. Disponible en: https://es.wikipedia.org/wiki/ASP.NET_MVC_Framework. [Accedido: 24-jul-2017].
[4] «Modelo–vista–controlador - Wikipedia, la enciclopedia libre». [En línea]. Disponible en: https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador. [Accedido: 24-jul-2017].
[5] «Programming in Java using the MVC architecture - CodeProject». [En línea]. Disponible en: https://www.codeproject.com/Articles/879896/Programming-in-Java-using-the-MVC-architecture. [Accedido: 22-jul-2017].
[6] «c# - How set up the Entity Framework model for Identity Framework to work against an existing database? - Stack Overflow». [En línea]. Disponible en: https://stackoverflow.com/questions/25593979/how-set-up-the-entity-framework-model-for-identity-framework-to-work-against-an. [Accedido: 04-jul-2017].
[7] «Google Fonts». [En línea]. Disponible en: https://fonts.google.com/specimen/Roboto. [Accedido: 22-jul-2017].
[8] «Patrón de diseño - Wikipedia, la enciclopedia libre». [En línea]. Disponible en: https://es.wikipedia.org/wiki/Patr%C3%B3n_de_dise%C3%B1o. [Accedido: 24-jul-2017].
[9] «Inversión de control - Wikipedia, la enciclopedia libre». [En línea]. Disponible en: https://es.wikipedia.org/wiki/Inversi%C3%B3n_de_control. [Accedido: 24-jul-2017].
[10] «The Repository Pattern». [En línea]. Disponible en: https://msdn.microsoft.com/es-es/library/ff649690.aspx. [Accedido: 04-jul-2017].
[11] «Queue-Based Load Leveling | Microsoft Docs». [En línea]. Disponible en: https://docs.microsoft.com/en-us/azure/architecture/patterns/queue-based-load-leveling. [Accedido: 22-jul-2017].
[12] «An Introduction to OAuth 2 | DigitalOcean». [En línea]. Disponible en: https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2. [Accedido: 22-jul-2017].
[13] «Material design - Wikipedia, la enciclopedia libre». [En línea]. Disponible en: https://es.wikipedia.org/wiki/Material_design. [Accedido: 24-jul-2017].
[14] «Material Design for Bootstrap». [En línea]. Disponible en: http://fezvrasta.github.io/bootstrap-material-design/. [Accedido: 05-jul-2017].
[15] «GitHub - ebidel/material-playground: Polymer material design playground». [En línea]. Disponible en: https://github.com/ebidel/material-playground. [Accedido: 22-jul-2017].
[16] «Mobilefish.com - Free online SEPA XML validation». [En línea]. Disponible en: http://www.mobilefish.com/services/sepa_xml_validation/sepa_xml_validation.php. [Accedido: 22-jul-2017].

La bibliografía consultada incluye los siguientes libros y material de asignaturas:


- Cloud Design Patterns, de Homer Alex, Sharp John, Brader Larry, Narumoto Masashi, Swanson Trent. ISBN: 978-1-62114-036-8. Editorial: Microsoft Developer Guidance. Publicado en 2014.
- Transparencias de la asignatura EI1027 Diseño e Implementación de Sistemas de Información.
- Transparencias de la asignatura EI1032 Análisis de Software

# Anexo A.1. Estructura del fichero XML Sepa Direct Debits Core

La estructura del fichero XML más reciente que se detalla a continuación se ha obtenido de http://www.ceca.es/ y https://www.febelfin.be/.

**1.1 Descripción**
El fichero de presentación contiene los registros de los adeudos directos SEPA que se presenten al cobro correspondientes a uno o más acreedores. Este fichero sigue el esquema *pain.008.001.02*, que es la última versión publicada para la validación del XML.

**1.2 Organización de los contenidos**
El fichero de presentación de adeudos XML se compone de 4 partes o bloques principales:

1. **Raíz del mensaje (message root):** este bloque es obligatorio y sólo aparece una vez. Contiene una única etiqueta que identifica la naturaleza del mensaje.

2. **Cabecera (group header):** este bloque es obligatorio y sólo aparece una vez. Contiene elementos comunes a todo el mensaje.

3. **Información del pago (Payment information):** es obligatorio y puede repetirse. Contiene elementos del ámbito del acreedor de la operación. Todas las operaciones dentro de este bloque deben tener la misma cuenta del acreedor y entidad del acreedor, la misma fecha de cobro solicitada y el mismo medio de pago. Por lo tanto, si la parte iniciadora o el acreedor tienen que emitir adeudos en un mismo mensaje, por ejemplo, para su abono en diferentes cuentas o con distintas fechas de ejecución, deberán utilizar bloques de Información del pago diferentes para cada cuenta y fecha.

4. **Información del adeudo directo individual (Direct Debit transaction information**): este bloque de información de adeudo directo individual forma parte del bloque de información del pago. Es obligatorio y puede repetirse *n* veces. Contiene elementos del ámbito del deudor de la operación. Se denomina *instrucción de adeudo directo* a la combinación del bloque de información del pago más el bloque de información de adeudo directo básico.

La estructura de los datos del fichero es la siguiente:

| | **pain.008.001.02** |
|---|---|
| | **A.RAÍZ DEL MENSAJE** |
| | Raíz del mensaje |
| | **B.CABECERA** |
| | Identificación de mensaje |
| | Fecha y hora de creación |
| | Número de operaciones |

| | |
|---|---|
| | Control de suma |
| + | Presentador |
| | **C.INFORMACIÓN DEL PAGO** |
| | Identificación del pago |
| | Método de pago |
| | Indicador de apunte en cuenta |
| | Número de operaciones |
| | Control de suma |
| + | Información del tipo de pago |
| | Fecha de cobro |
| + | Acreedor |
| + | Cuenta del acreedor |
| + | Entidad del acreedor |
| + | Último acreedor |
| | Cláusula de gastos |
| + | Identificación del acreedor |
| | **D.INFORMACIÓN DE ADEUDO DIRECTO INDIVIDUAL** |
| + | Identificación del pago |
| | Importe |
| | Cláusula de gastos |
| + | Operación de adeudo directo |
| + | Último acreedor |
| + | Entidad del deudor |
| + | Deudor |
| + | Cuenta del deudor |
| + | Último deudor |
| + | Propósito |
| + | Concepto |

*Tabla 27. Estructura del fichero pain.008.001.02*

A continuación se detalla la información de cada campo dentro de estos bloques así como la etiqueta XML correspondiente.

### 1.3 Diseño y descripción de los registros

El número de + en el nombre de los campos indica la profundidad del campo en la estructura del XML.

#### 1.3.1 Raíz del mensaje

| Índice | Ocurrencias | Nombre | Etiqueta XML | Longitud |
|---|---|---|---|---|
| | [1...1] | +Raíz del mensaje | <CstmrDrctDbtInitn> | |

*Tabla 28. Raíz del mensaje*

#### 1.3.2 Cabecera

Los siguientes elementos tienen que estar siempre presentes:

- Identificación de mensaje / Message Identification
- Fecha y hora de creación / Creation Date Time
- Número de operaciones / Number Of Transactions ("1" si el mensaje contiene una única operación, "n" en el caso de que contenga múltiples)
- Presentador/ Initiating Party (parte que envía el mensaje)

El listado completo de la estructura de campos de esta sección es la siguiente:

| Índice | Ocurrencias | Nombre | Etiqueta XML | Longitud |
|---|---|---|---|---|
| 1.0 | [1...1] | + Cabecera | <GrpHdr> | |
| 1.1 | [1...1] | ++ Identificación del mensaje | <MsgId> | 35 |
| 1.2 | [1...1] | ++ Fecha y hora de creación | <CreDtTm> | 19 |
| 1.6 | [1...1] | ++ Número de operaciones | <NbOfTxs> | 15 |
| 1.7 | [0...1] | ++ Control de suma | <CtrlSum> | 18 |
| 1.8 | [1...1] | ++ Parte iniciadora | <InitgPty> | |
| 1.8 | [0...1] | +++ Nombre | <Nm> | 70 |
| 1.8 | [0...1] | +++ Identificación | <Id> | |
| 1.8 | [1...1]{Or | ++++ Persona jurídica | <OrgId> | |
| | [0...1] | +++++ BIC o BEI | <BICOrBEI> | 11 |
| | [0...n] | +++++ Otra | <Othr> | |
| | [1...1] | ++++++ Identificación | <Id> | 35 |
| | [0...1] | ++++++ Nombre del esquema | <SchmeNm> | |
| | [1...1]{{Or | +++++++ Código | <Cd> | 4 |
| | [1...1]Or}} | +++++++ Propietario | <Prtry> | 35 |
| | [0...1] | ++++++ Emisor | <Issr> | 35 |
| 1.8 | [1...1]Or} | ++++ Persona física | <PrvtId> | |
| | [0...1] | +++++ Fecha y lugar de nacimiento | <DtAndPlcOfBirth> | |

| | | | | |
|---|---|---|---|---|
| | [1...1] | ++++++ Fecha de nacimiento | <BirthDt> | 10 |
| | [0...1] | ++++++ Provincia de nacimiento | <PrvcOfBirth> | 35 |
| | [1...1] | ++++++ Ciudad de nacimiento | <CityOfBirth> | 35 |
| | [1...1] | ++++++ País de nacimiento | <CtryOfBirth> | 2 |
| | [0...1] | +++++ Otra | <Othr> | |
| | [1...1] | ++++++ Identificación | <Id> | 35 |
| | [0...1] | ++++++ Nombre del esquema | <SchmeNm> | |
| | [1...1]{{Or | +++++++ Código | <Cd> | 4 |
| | [1...1]Or}} | +++++++ Propietario | <Prtry> | 35 |
| | [0...1] | ++++++ Emisor | <Issr> | 35 |

*Tabla 29. Estructura de campos de la cabecera*

### 1.3.4 Información del pago

Los elementos que siempre aparecen en este bloque son:

- Identificación de la información del pago / Payment Information Identification
- Método de pago / Payment Method
- Fecha de cobro / Requested Collection Date
- Acreedor/ Creditor
- Cuenta del acreedor/ Creditor Account
- Entidad del acreedor/ Creditor Agent
- Deudor
- Cuenta del deudor

El listado completo de campos de esta sección para facilitar la posible extensión de la aplicación en un futuro son:

| Índice | Ocurrencias | Nombre | Etiqueta XML | Longitud |
|---|---|---|---|---|
| 2.0 | [1...n] | +Información del pago | <PmtInf> | |
| 2.1 | [1...1] | ++ Identificación de la información del pago | <PmtInfId> | 35 |
| 2.2 | [1...1] | ++ Método de pago | <PmtMtd> | 2 |
| 2.3 | [0...1] | ++ Indicador de apunte en cuenta | <BtchBookg> | 5 |
| 2.4 | [0...1] | ++ Número de operaciones | <NbOfTxs> | 15 |
| 2.5 | [0...1] | ++ Control de suma | <CtrlSum> | 18 |
| 2.6 | [0...1] | ++ Información del tipo de pago | <PmtTpInf> | |
| 2.8 | [0...1] | ++ Nivel de servicio | <SvcLvl> | |
| 2.9 | [0...1] | +++ Código | <Cd> | 4 |
| 2.11 | [0...1] | +++ Instrumento local | <LclInstrm> | |
| 2.12 | [0...1] | ++++ Código | <Cd> | 35 |
| 2.14 | [0...1] | +++ Secuencia del adeudo | <SeqTp> | 4 |
| 2.15 | [0...1] | +++ Categoría del propósito | <CtgyPurp> | |
| 2.16 | [1...1 {Or] | ++++ Código | <Cd> | 4 |

| | | | | |
|---|---|---|---|---|
| 2.17 | [1..1] Or} | ++++ Propietario | <Prtry> | 35 |
| 2.18 | [1...1] | ++ Fecha de cobro | <ReqdColltnDt> | 10 |
| 2.19 | [1...1] | ++ Acreedor | <Cdtr> | |
| 2.19 | [0...1] | +++ Nombre | <Nm> | 70 |
| 2.19 | [0...1] | +++ Dirección postal | <PstlAdr> | |
| 2.19 | [0...1] | ++++ País | <Ctry> | 2 |
| 2.19 | [0...2] | ++++ Dirección en texto libre | <AdrLine> | 70 |
| 2.20 | [1...1] | ++ Cuenta del acreedor | <CdtrAcct> | |
| 2.20 | [1...1] | +++ Identificación | <Id> | |
| | [1...1] | ++++ IBAN | <IBAN> | 34 |
| 2.20 | [0...1] | +++ Moneda | <Ccy> | 3 |
| 2.21 | [1...1] | ++ Entidad del acreedor | <CdtrAgt> | |
| | [1...1] | +++ Identificación de la entidad del acreedor | <FinInstnId> | |
| | [0...1] | ++++ BIC | <BIC> | 11 |
| | [0...1] | ++++ Otra | <Othr> | |
| | [1...1] | +++++ Identificación | <Id> | 35 |
| 2.23 | [0...1] | ++ Último acreedor | <UltmtCdtr> | |
| 2.23 | [0...1] | +++ Nombre | <Nm> | 70 |
| 2.23 | [0...1] | +++ Identificación | <Id> | |
| 2.23 | [1...1]{Or | ++++ Persona jurídica | <OrgId> | |
| | [0...1] | +++++ BIC o BEI | <BICOrBEI> | 11 |
| | [0...1] | +++++ Otra | <Othr> | |
| | [1...1] | ++++++ Identificación | <Id> | 35 |
| | [0...1] | ++++++ Nombre del esquema | <SchmeNm> | |
| | [1...1]{{Or | +++++++ Código | <Cd> | 4 |
| | [1...1]Or}} | +++++++ Propietario | <Prtry> | 35 |
| | [0...1] | ++++++ Emisor | <Issr> | 35 |
| 2.23 | [1...1]Or} | ++++ Persona física | <PrvtId> | |
| | [0...1] | +++++ Fecha y lugar de nacimiento | <DtAndPlcOfBirth> | |
| | [1...1] | ++++++ Fecha de nacimiento | <BirthDt> | 10 |
| | [0...1] | ++++++ Provincia de nacimiento | <PrvcOfBirth> | 35 |
| | [1...1] | ++++++ Ciudad de nacimiento | <CityOfBirth> | 35 |
| | [1...1] | ++++++ País de nacimiento | <CtryOfBirth> | 2 |
| | [0...1] | +++++ Otra | <Othr> | |
| | [1...1] | ++++++ Identificación | <Id> | 35 |
| | [0...1] | ++++++ Nombre del esquema | <SchmeNm> | |

| | | | | |
|---|---|---|---|---|
| | [1...1]{{Or | +++++++ Código | <Cd> | 4 |
| | [1...1]Or}} | +++++++ Propietario | <Prtry> | 35 |
| | [0...1] | ++++++ Emisor | <Issr> | 35 |
| 2.24 | [0...1] | ++ Cláusula de gastos | <ChrgBr> | 4 |
| 2.27 | [0...1] | ++ Identificación del acreedor | <CdtrSchmeId> | |
| 2.27 | [0...1] | +++ Identificación | <Id> | |
| 2.27 | [1...1] | ++++ Identificación privada | <PrvtId> | |
| 2.27 | [0...n] | +++++ Otra | <Othr> | |
| | [1...1] | ++++++ Identificación | <Id> | 35 |
| | [0...1] | ++++++ Nombre del esquema | <ScmeNm> | |
| | [1...1] | +++++++ Propietario | <Prtry> | 35 |
| 2.28 | [1...n] | ++ Información de la operación de adeudo directo | <DrctDbtTxInf> | |
| 2.29 | [1...1] | +++ Identificación del pago | <PmtId> | |
| 2.30 | [0...1] | ++++ Identificación de la instrucción | <InstrId> | 35 |
| 2.31 | [1...1] | ++++ Identificación de extremo a extremo | <EndToEndId> | 35 |
| 2.44 | [1...1] | +++ Importe ordenado | <InstdAmt> | 18 |
| 2.45 | [0...1] | +++ Cláusula de gastos | <ChrgBr> | 4 |
| 2.46 | [0...1] | +++ Operación de adeudo directo | <DrctDbtTx> | |
| 2.47 | [0...1] | ++++ Información del mandato | <MndtRltdInf> | |
| 2.48 | [0...1] | +++++ Identificación del mandato | <MndtId> | 35 |
| 2.49 | [0...1] | +++++ Fecha de firma | <DtOfSgntr> | 10 |
| 2.50 | [0...1] | +++++ Indicador de modificación | <AmdmntInd> | 5 |
| 2.51 | [0...1] | +++++ Detalles de la modificación | <AmdmntInfDtls> | |
| 2.52 | [0...1] | ++++++ Identificación del mandato original | <OrgnlMndtId> | 35 |
| 2.53 | [0...1] | ++++++ Identificación del acreedor original | <OrgnlCdtrSchmeId> | |
| 2.53 | [0...1] | +++++++ Nombre | <Nm> | 70 |
| 2.53 | [0...1] | +++++++ Identificación | <Id> | |
| 2.53 | [1...1] | ++++++++ Identificación privada | <PrvtId> | |
| 2.53 | [0...n] | +++++++++ Otra | <Othr> | |
| | [1...1] | ++++++++++ Identificación | <Id> | 35 |
| | [0...1] | ++++++++++ Nombre del esquema | <SchmeNm> | |
| | [1...1] | +++++++++++ Propietario | <Prtry> | 35 |
| 2.57 | [0...1] | ++++++ Cuenta del deudor original | <OrgnlDbtAcct> | |
| | [1...1] | +++++++ Identificación | <Id> | |
| | [1...1]{Or | ++++++++ IBAN | <IBAN> | 34 |
| | [1...1]Or} | ++++++++ Otra | <Othr> | |

| | | | | |
|---|---|---|---|---|
| | [1...1] | ++++++++ Identificación | <Id> | 35 |
| 2.58 | [0...1] | ++++++ Entidad del deudor original | <OrgnlDbtrAgt> | |
| | [1...1] | +++++++ Identificación de la entidad del deudor | <FinInstnId> | |
| | [1...1] | ++++++++ BIC | <BIC> | 11 |
| 2.62 | [0...1] | +++++ Firma electrónica | <ElctrncSgntr> | 1025 |
| 2.66 | [0...1] | ++++ Identificación del acreedor | <CdtrSchmeId> | |
| 2.66 | [0...1] | +++++ Identificación | <Id> | |
| 2.66 | [1...1] | ++++++ Identificación privada | <PrvtId> | |
| 2.66 | [0...n] | +++++++ Otra | <Othr> | |
| | [1...1] | ++++++++ Identificación | <Id> | 35 |
| | [0...1] | ++++++++ Nombre del esquema | <SchmeNm> | |
| | [1...1] | +++++++++ Propietario | <Prtry> | 35 |
| 2.69 | [0...1] | +++ Último acreedor | <UltmtCdtr> | |
| 2.69 | [0...1] | ++++ Nombre | <Nm> | 70 |
| 2.69 | [0...1] | ++++ Identificación | <Id> | |
| 2.69 | [1...1]{Or | +++++ Persona jurídica | <OrgId> | |
| | [0...1] | ++++++ BIC o BEI | <BICOrBEI> | 11 |
| | [0...1] | ++++++ Otra | <Othr> | |
| | [1...1] | +++++++ Identificación | <Id> | 35 |
| | [0...1] | +++++++ Nombre del esquema | <SchmeNm> | |
| | [1...1]{{Or | ++++++++ Código | <Cd> | 4 |
| | [1...1]Or}} | ++++++++ Propietario | <Prtry> | 35 |
| | [0...1] | +++++++ Emisor | <Issr> | 35 |
| 2.69 | [1...1]Or} | +++++ Persona física | <PrvtId> | |
| | [0...1] | ++++++ Fecha y lugar de nacimiento | <DtAndPlcOfBirth> | |
| | [1...1] | +++++++ Fecha de nacimiento | <BirthDt> | 10 |
| | [0...1] | +++++++ Provincia de nacimiento | <PrvcOfBirth> | 35 |
| | [1...1] | +++++++ Ciudad de nacimiento | <CityOfBirth> | 35 |
| | [1...1] | +++++++ País de nacimiento | <CtryOfBirth> | 2 |
| | [0...1] | ++++++ Otra | <Othr> | |
| | [1...1] | +++++++ Identificación | <Id> | 35 |
| | [0...1] | +++++++ Nombre del esquema | <SchmeNm> | |
| | [1...1]{{Or | ++++++++ Código | <Cd> | 4 |
| | [1...1]Or}} | ++++++++ Propietario | <Prtry> | 35 |
| | [0...1] | +++++++ Emisor | <Issr> | 35 |
| 2.70 | [1...1] | +++ Entidad del deudor | <DbtrAgt> | |

| | | | | |
|---|---|---|---|---|
| | [1...1] | ++++ Identificación de la entidad del deudor | <FinInstnId> | |
| | [0...1] | +++++ BIC | <BIC> | 11 |
| | [0...1] | +++++ Otra | <Othr> | |
| | [1...1] | ++++++ Identificación | <Id> | 35 |
| 2.72 | [1...1] | +++ Deudor | <Dbtr> | |
| 2.72 | [0...1] | ++++ Nombre | <Nm> | 70 |
| 2.72 | [0...1] | ++++ Dirección postal | <PstlAdr> | |
| 2.72 | [0...1] | +++++ País | <Ctry> | 2 |
| 2.72 | [0...2] | +++++ Dirección en texto libre | <AdrLine> | 70 |
| 2.72 | [0...1] | ++++ Identificación | <Id> | |
| 2.72 | [1...1]{Or | +++++ Persona jurídica | <OrgId> | |
| | [0...1] | ++++++ BIC o BEI | <BICOrBEI> | 11 |
| | [0...1] | ++++++ Otra | <Othr> | |
| | [1...1] | +++++++ Identificación | <Id> | 35 |
| | [0...1] | +++++++ Nombre del esquema | <SchmeNm> | |
| | [1...1]{{Or | ++++++++ Código | <Cd> | 4 |
| | [1...1]Or}} | ++++++++ Propietario | <Prtry> | 35 |
| | [0...1] | +++++++ Emisor | <Issr> | 35 |
| 2.72 | [1...1]Or} | +++++ Persona física | <PrvtId> | |
| | [0...1] | ++++++ Fecha y lugar de nacimiento | <DtAndPlcOfBirth> | |
| | [1...1] | +++++++ Fecha de nacimiento | <BirthDt> | 10 |
| | [0...1] | +++++++ Provincia de nacimiento | <PrvcOfBirth> | 35 |
| | [1...1] | +++++++ Ciudad de nacimiento | <CityOfBirth> | 35 |
| | [1...1] | +++++++ País de nacimiento | <CtryOfBirth> | 2 |
| | [0...1] | ++++++ Otra | <Othr> | |
| | [1...1] | +++++++ Identificación | <Id> | 35 |
| | [0...1] | +++++++ Nombre del esquema | <SchmeNm> | |
| | [1...1]{{Or | ++++++++ Código | <Cd> | 4 |
| | [1...1]Or}} | ++++++++ Propietario | <Prtry> | 35 |
| | [0...1] | +++++++ Emisor | <Issr> | 35 |
| 2.73 | [1...1] | +++ Cuenta del deudor | <DbtrAcct> | |
| | [1...1] | ++++ Identificación | <Id> | |
| | [1...1] | +++++ IBAN | <IBAN> | 34 |
| 2.74 | [0...1] | +++ Último deudor | <UltmtDbtr> | |
| 2.74 | [0...1] | ++++ Nombre | <Nm> | 70 |
| 2.74 | [0...1] | ++++ Identificación | <Id> | |

| | | | | |
|---|---|---|---|---|
| 2.74 | [1...1]{Or | +++++ Persona jurídica | <OrgId> | |
| | [0...1] | ++++++ BIC o BEI | <BICOrBEI> | 11 |
| | [0...1] | ++++++ Otra | <Othr> | |
| | [1...1] | +++++++ Identificación | <Id> | 35 |
| | [0...1] | +++++++ Nombre del esquema | <SchmeNm> | |
| | [1...1]{{Or | ++++++++ Código | <Cd> | 4 |
| | [1...1]Or}} | ++++++++ Propietario | <Prtry> | 35 |
| | [0...1] | +++++++ Emisor | <Issr> | 35 |
| 2.74 | [1...1]Or} | +++++ Persona física | <PrvtId> | |
| | [0...1] | ++++++ Fecha y lugar de nacimiento | <DtAndPlcOfBirth> | |
| | [1...1] | +++++++ Fecha de nacimiento | <BirthDt> | 10 |
| | [0...1] | +++++++ Provincia de nacimiento | <PrvcOfBirth> | 35 |
| | [1...1] | +++++++ Ciudad de nacimiento | <CityOfBirth> | 35 |
| | [1...1] | +++++++ País de nacimiento | <CtryOfBirth> | 2 |
| | [0...1] | ++++++ Otra | <Othr> | |
| | [1...1] | +++++++ Identificación | <Id> | 35 |
| | [0...1] | +++++++ Nombre del esquema | <SchmeNm> | |
| | [1...1]{{Or | ++++++++ Código | <Cd> | 4 |
| | [1...1]Or}} | ++++++++ Propietario | <Prtry> | 35 |
| | [0...1] | +++++++ Emisor | <Issr> | 35 |
| 2.76 | [0...1] | +++ Propósito | <Purp> | |
| 2.77 | [1...1] | ++++ Código | <Cd> | 4 |
| 2.88 | [0...1] | +++ Concepto | <RmtInf> | |
| 2.89 | [0...n] | ++++ No estructurado | <Ustrd> | 140 |
| 2.90 | [0...n] | ++++ Estructurado | <Strd> | |
| 2.110 | [1...1] | +++++ Referencia facilitada por el acreedor | <CdtrRefInf> | |
| 2.111 | [1...1] | ++++++ Tipo de referencia | <Tp> | |
| 2.112 | [1...1] | +++++++ Código o propietario | <CdOrPrtry> | |
| 2.113 | [1...1] | ++++++++ Código | <Cd> | 4 |
| 2.115 | [0...1] | +++++++ Emisor | <Issr> | 35 |
| 2.116 | [1...1] | ++++++ Referencia | <Ref> | 35 |

*Tabla 30. Estructura de campos de la información del pago*

**Formatos específicos y reglas de uso especiales a tener en cuenta:**

| | |
|---|---|
| 1.1 | Identificación del mensaje |

| | |
|---|---|
| | Regla de uso: El acreedor/presentador debe asegurarse que esta referencia es única para cada entidad destino del mensaje de presentación para un período de tiempo previamente acordado:<br>Regla de uso para financiación de remesas: el acreedor compondrá esta referencia incluyendo en las cuatro primeras posiciones de la identificación del mensaje el prefijo FSSD |
| 1.2 | Fecha y hora de creación<br>Formato: ISODateTime YYYY-MM-DDThh:mm:ss<br>Ejemplo: 2010-11-05T09:25:15 |
| | Control de suma<br>Formato: Tiene 18 dígitos máximos, 2 de ellos son decimales. Los dígitos correspondientes a los decimales irán separados por un punto. |
| 1.8 | Parte iniciadora<br>Regla de uso: para el sistema de adeudos SEPA se utilizará exclusivamente la etiqueta "Otra" estructurada. |
| | BIC o BEI<br>Regla de uso: sólo se permite un BIC o BEI válido. El BIC o BEI válido está registrado por la Autoridad de Registro ISO 9362, y consta de ocho (8) u once (11) caracteres. |
| | Propietario<br>Regla de uso: debe consignarse el literal "SEPA" |
| | Fecha de nacimiento<br>Formato: ISODate YYYY-MM-DD<br>Ejemplo: 1990-03-08 |
| | País de nacimiento<br>Formato: el código de país de 2 letras se obtiene de la ISO 3166, Alpha 2 |
| 2.2 | Método de pago<br>Regla de uso: Solamente se admite el código "DD" (Direct Debit) |
| 2.3 | Indicador de apunte en cuenta<br>Formato:<br>1. "true" indica que se solicita un apunte en cuenta por la suma de los importes de todas las operaciones en el mensaje<br>2. "false" indica que se solicita un apunte en cuenta por cada una de las operaciones incluidas en el mensaje<br>Regla de uso: Cuando no se utilice este indicador, se aplicarán las condiciones que el acreedor haya acordado previamente con su entidad. |
| 2.9 | Código<br>Regla de uso: solamente se admite el código "SEPA" |
| 2.14 | Secuencia del adeudo<br>Regla de uso: se admiten los valores:<br>❑ FNAL: último adeudo de una serie de adeudos recurrentes<br>❑ FRST: primer adeudo de una serie de adeudos recurrentes<br>❑ OOFF: adeudo correspondiente a una operación con un único pago<br>❑ RCUR: adeudo de una serie de adeudos recurrentes, cuando no se trata ni del primero ni del último |
| 2.23 | Último acreedor<br>Reglas de uso: |

| | |
|---|---|
| | • este elemento sólo aparecerá cuando el último acreedor sea diferente del acreedor.<br>• puede aparecer bien en el nodo 'Información del pago' (2.0) bien en el nodo 'Información de la operación de adeudo directo' (2.28), pero solamente en uno de ellos. Se recomienda en el 2.0. |
| 2.24 | Cláusula de gastos<br>Regla de uso: solamente se admite el código 'SLEV' |
| 2.44 | Importe ordenado<br>Reglas de uso:<br>1. máximo 11 dígitos, de los cuales 2 son decimales<br>2. separador decimal: "." (punto)<br>3. en la etiqueta debe incluirse el código 'EUR'<br>Ejemplo: <InstdAmt Ccy="EUR">537.00</InstdAmt> |
| 2.113 | Código<br>Formato: solamente se admite el código 'SCOR' |
| 2.116 | Referencia<br>Este campo se cumplimentará de acuerdo a lo definido en el estándar ISO11649, de la siguiente manera (25 posiciones):<br>1. las dos primeras posiciones serán RF, que identifica la referencia<br>2. las posiciones 3 y 4 se cumplimentarán con dos dígitos de control<br>3. de las posiciones 5 a 25 (21 caracteres en total) se cumplimentará la referencia como tal, de forma que confirmen que la referencia es completamente correcta. |

*Tabla 31. Formatos específicos de campos en el XML*

# Anexo A.2 Manual de usuario de la aplicación

Para poder crear una remesa en la aplicación PDCnet Remeseas se deben seguir los pasos que se detallan a continuación.

1) Si aún no tenemos cuenta en la aplicación, debemos registrarnos. Podemos utilizar el formulario de registro habitual (ver figura 1) o bien registrarnos utilizando una cuenta de alguna plataforma externa como Facebook, Twitter, Microsoft o LinkedIn.

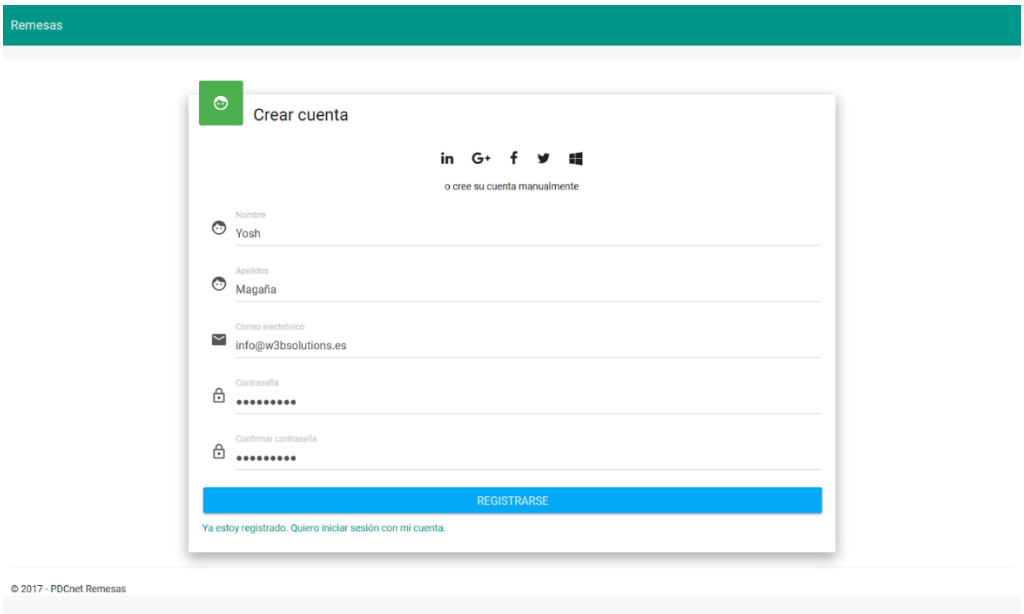



*Figura 1. Formulario de registro manual*

2) Una vez registrados tendremos el perfil de usuario invitado asignado por defecto, por lo que solamente podremos ver los datos, pero no podemos crear nuevas remesas ni modificar las actuales. Para ello, otro usuario administrador deberá cambiar nuestro rol accediendo a la aplicación al apartado de gestión de usuarios y modificar nuestra cuenta (ver figura 2):

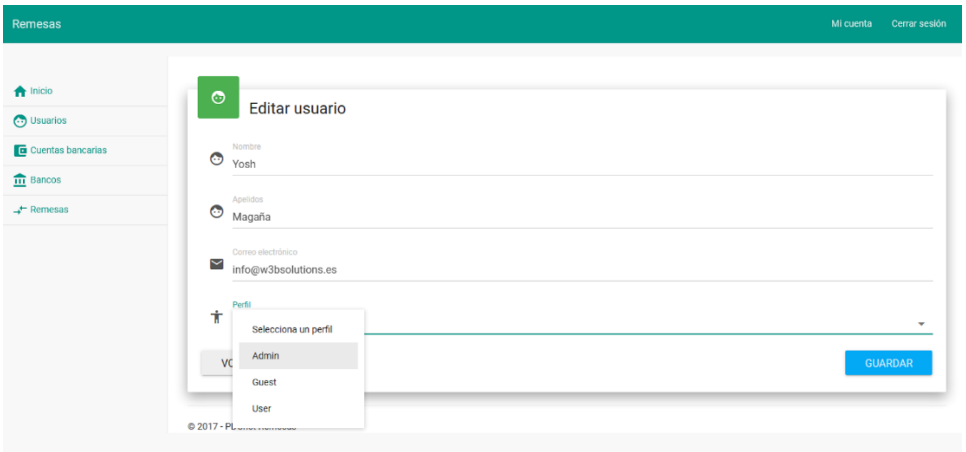



*Figura 2. Cambio de rol de usuario*

3) Una vez tenemos el perfil de Usuario o Administrador podremos crear todo lo necesario para generar una remesa. Lo primero es dar de alta la entidad bancaria donde tenemos la cuenta en la que queremos recibir los cobros. Para ello accedemos al apartado de Bancos (ver figura 3) y damos de alta la entidad introduciendo el nombre y el código BIC de la misma (ver figura 4).

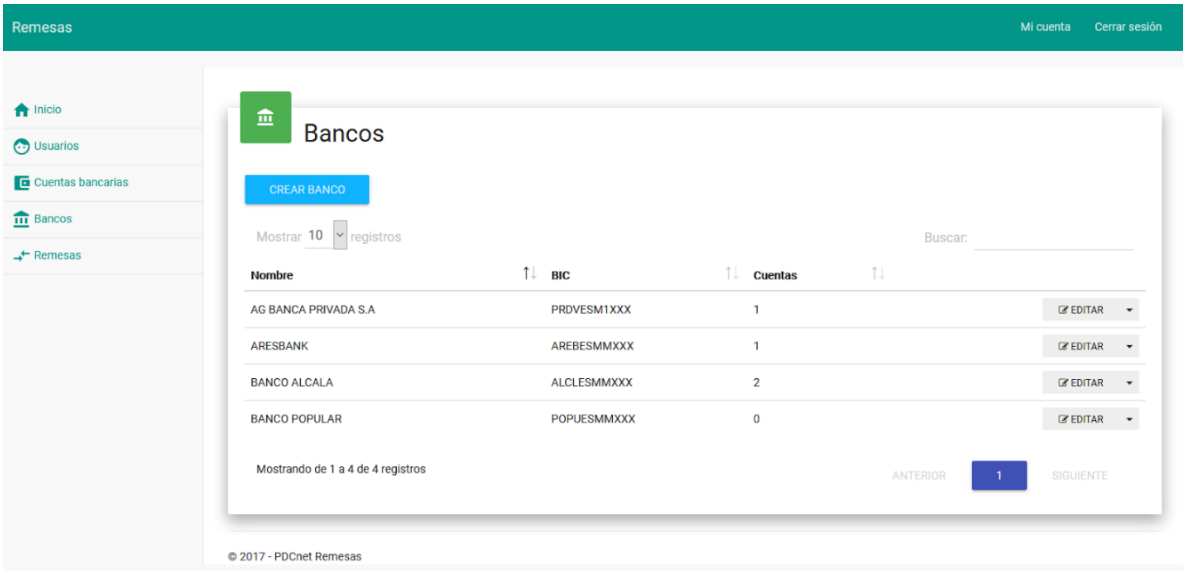



*Figura 3. Apartado de bancos*

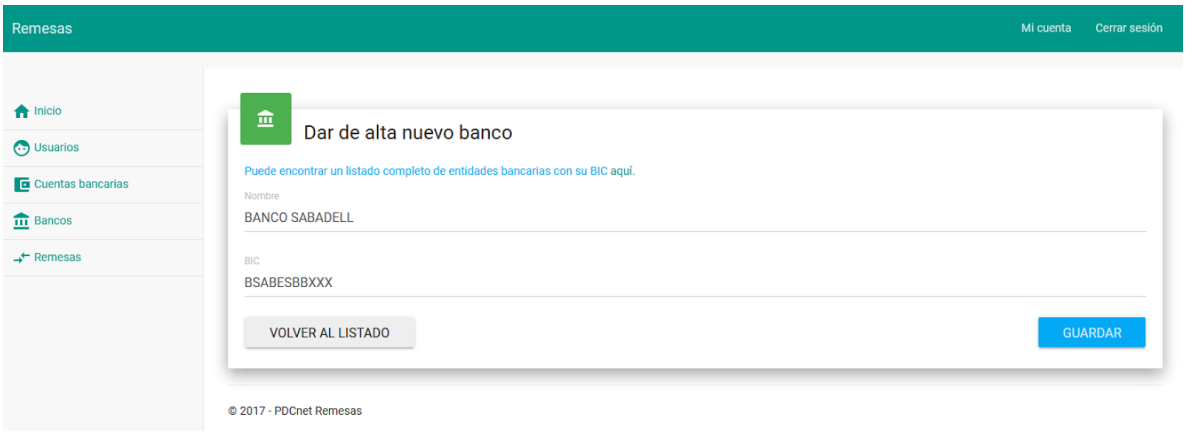



*Figura 4. Formulario de alta de nuevo banco*

4) Una vez creada la entidad, procedemos a dar de alta la cuenta bancaria en el apartado Cuentas bancarias (ver figura 5) pinchando en el botón “Crear cuenta bancaria” (ver figura 6) e introduciendo el IBAN, todo junto y sin espacios ni guiones, y eligiendo la entidad bancaria registrada previamente.

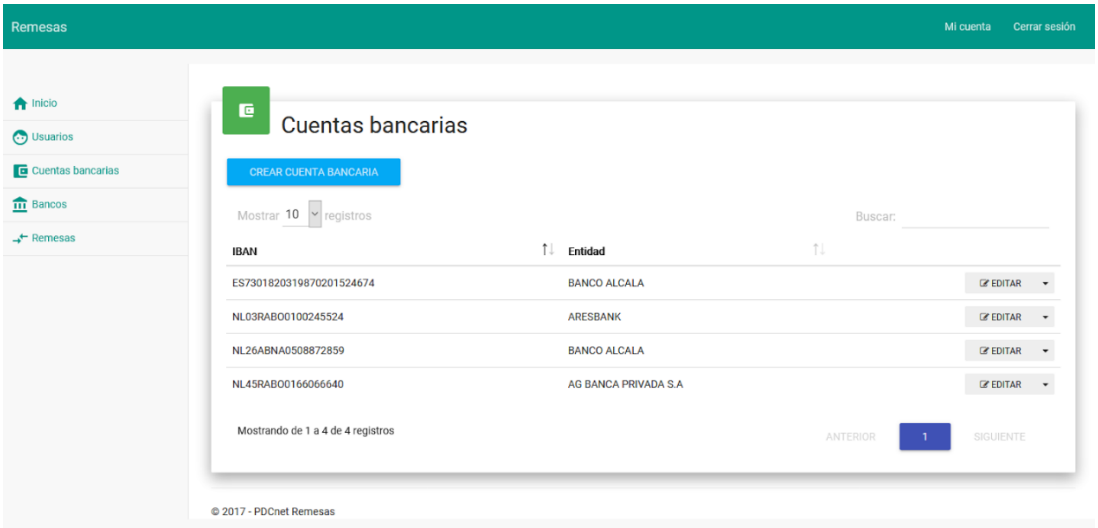



*Figura 5. Apartado de cuentas bancarias*

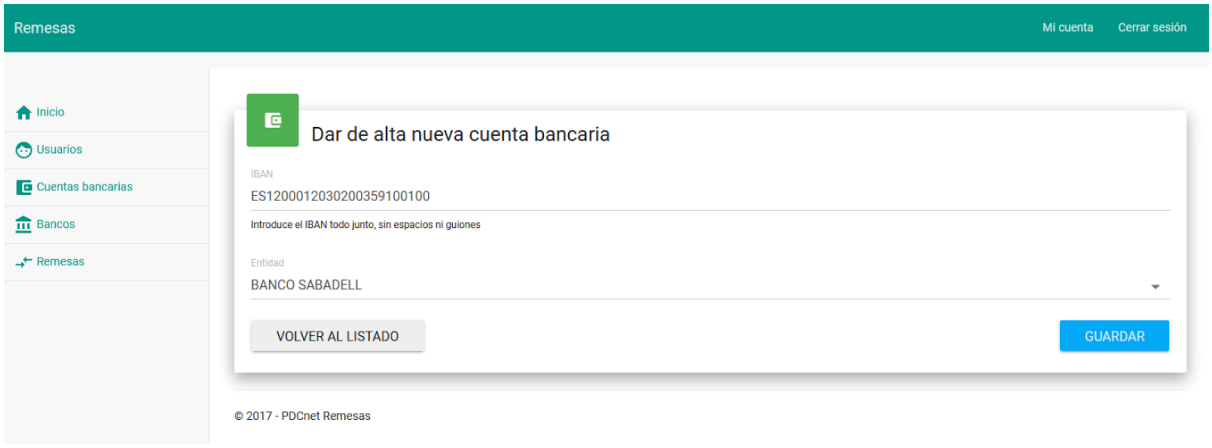



*Figura 6. Formulario de alta de cuenta bancaria*

5) Llegados a este punto podemos proceder a crear la remesa, para ello vamos al apartado Remesas (ver figura 7) y pulsamos el botón “Generar remesa” para que se nos muestre el formulario de creación de remesa (ver figura 8).

En este formulario debemos rellenar todos los campos, así como determinar si queremos que las facturas se agrupen por cliente (en el caso de que un cliente tenga múltiples facturas en la misma remesa). Debemos seleccionar las facturas que queremos incluir en la remesa y hacer clic en el botón “Incluir seleccionadas”. Podemos filtrar las facturas por número, cliente, rango de fechas, ver si alguna factura tiene problemas que impidan que la podamos incluir en la remesa o también elegir si queremos o no incluir facturas que ya han sido remesadas previamente. Podemos acceder a más detalles de cada factura pinchando en el botón “Ver detalles” (ver figura 9).

Una vez incluidas todas las facturas deseadas pinchamos en el botón “Crear remesa”. Si todo es correcto veremos un mensaje en la parte superior que indica que la remesa se ha generado con éxito y se activa el botón de “Descargar XML”. Pinchando en este botón podemos descargarnos el fichero XML que deberemos enviar al banco para procesar su cobro (figura 10).

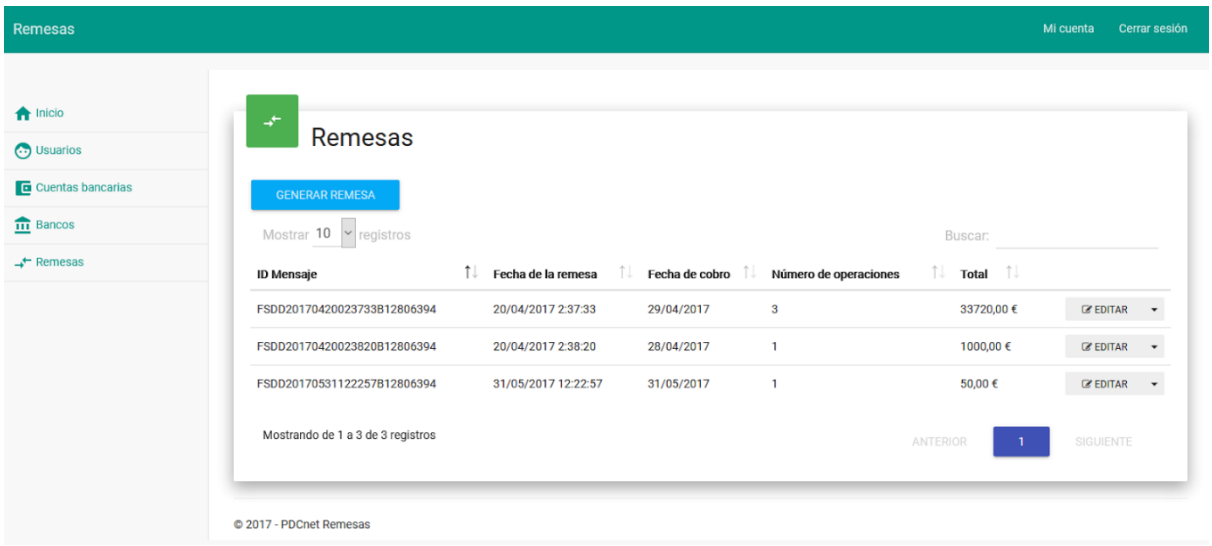



*Figura 7. Apartado de Remesas*

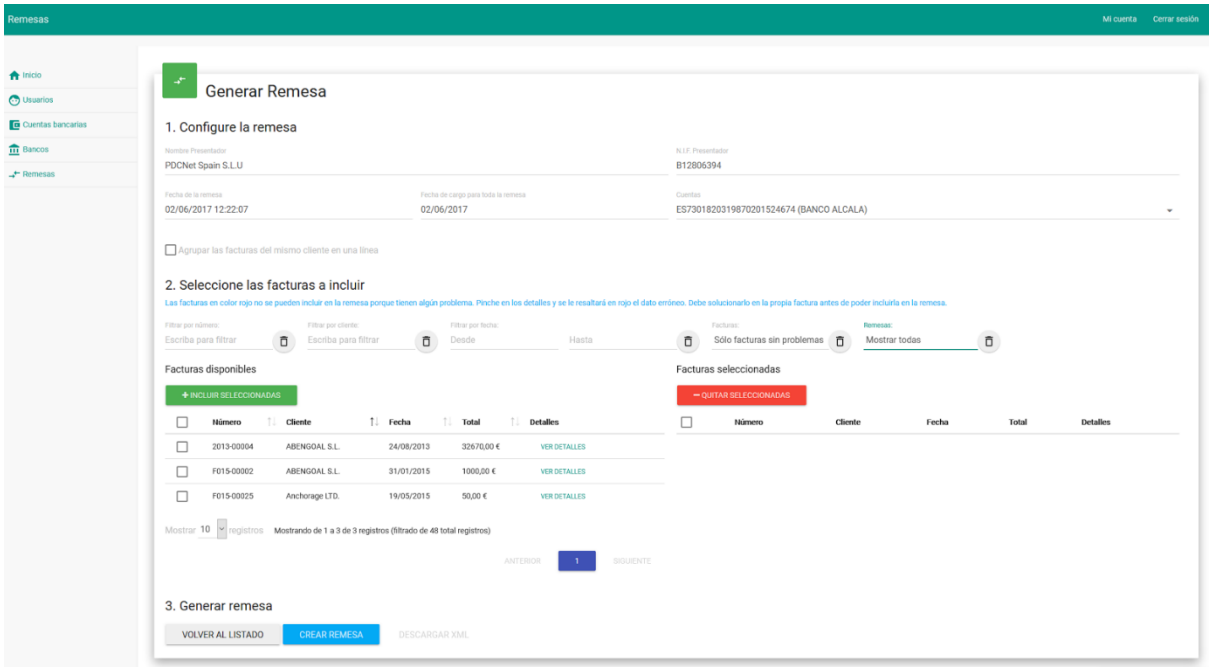



*Figura 8. Formulario de creación de remesa*

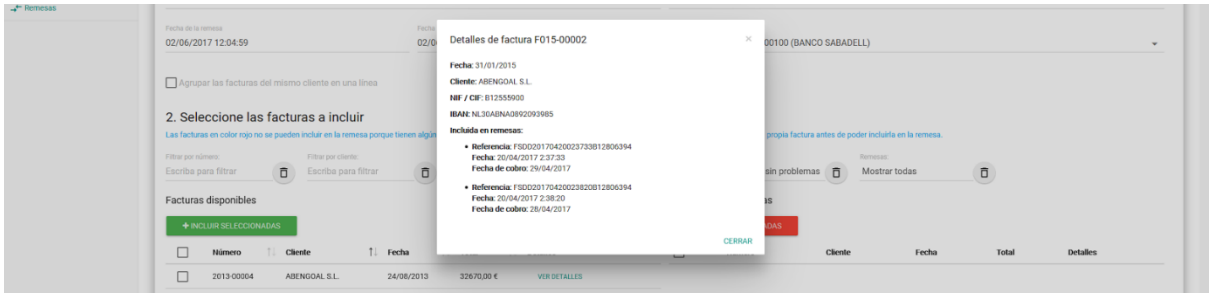



*Figura 9. Vista de detalles de factura*

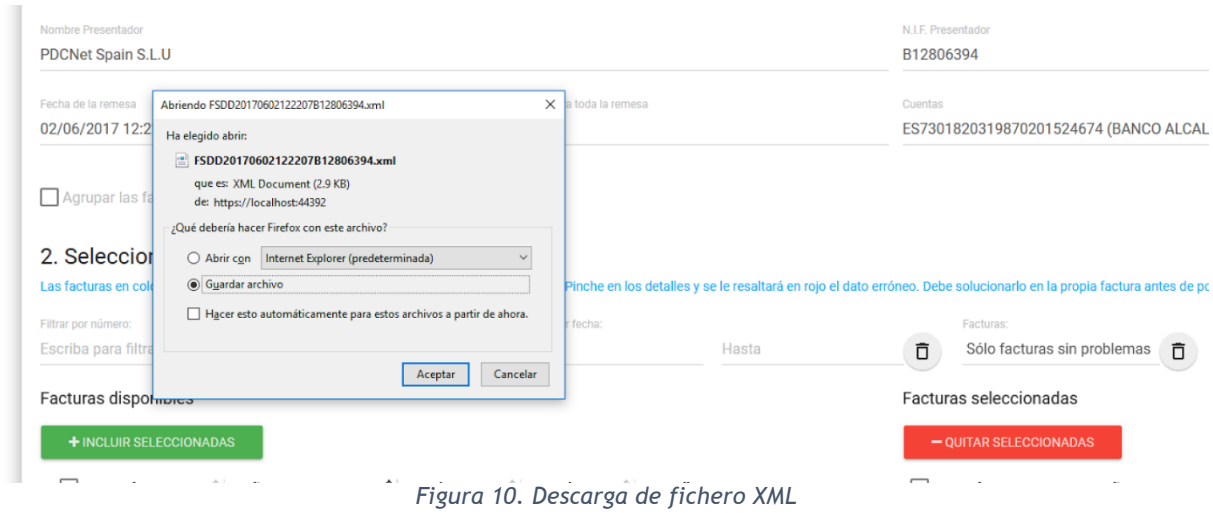



*Figura 10. Descarga de fichero XML*