Prototipo de un sitio web para la gestión económica de una comunidad de vecinos en Ruby on Rails

Mack Yao Fu

Grado en Ingeniería Informática

Tutores: D. Carmelo Cuenca Hernández
D.ª Francisca Quintana Domínguez

13 de julio de 2018

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

TFT04

SOLICITUD DE DEFENSA DE TRABAJO DE FIN DE TÍTULO

D/Dª Mack Yao Fu, autor del Trabajo de Fin de Título Prototipo de un sitio web para la gestión económica de una comunidad de vecinos en Ruby on Rails, correspondiente a la titulación Grado en Ingeniería Informática, en colaboración con la empresa/proyecto (indicar en su caso) _____

S O L I C I T A

que se inicie el procedimiento de defensa del mismo, para lo que se adjunta la documentación requerida.

Asimismo, con respecto al registro de la propiedad intelectual/industrial del TFT, declara que:

[ ] Se ha iniciado o hay intención de iniciarlo (defensa no pública).
[X] No está previsto.

Y para que así conste firma la presente.

Las Palmas de Gran Canaria, a 12 de julio de 2018.

El estudiante

Fdo.: _____

A rellenar y firmar **obligatoriamente** por el/los tutor/es

En relación a la presente solicitud, se informa:

[X] Positivamente

[ ] Negativamente
(la justificación en caso de informe negativo deberá incluirse en el TFT05)

QUINTANA DOMINGUEZ FRANCISCA CANDELARIA - 43755165B

Firmado digitalmente por QUINTANA DOMINGUEZ FRANCISCA CANDELARIA - 43755165B
Fecha: 2018.07.13 14:07:39 +02'00'

Fdo.: Carmelo Cuenca Francisca Quintana

DIRECTOR DE LA ESCUELA DE INGENIERÍA INFORMÁTICA

Quiero agradecer en primer lugar a mi pareja, Idania, por la paciencia y ánimos que me ha dado durante mi último año académico. Gracias a ti, he podido ir creciendo como persona, y has sido la que me daba el empujón cuando lo necesitaba y la que estaba ahí cuando estaba en lo bajo.

A mis amigos y familia, entre los que cabe mencionar Esther y Pedro, por estar ahí cuando los he necesitado. Mi hermano Víctor, que sé que siempre me apoyará en todo lo que haga, y una mención a todos aquellos amigos que me han apoyado durante el trayecto.

Un último agradecimiento a mis tutores, por orientar este proyecto a algo tangible y real.

Índice

1. Introducción

El proyecto a realizar trata de «Un prototipo de sitio web destinado a la gestión económica de una comunidad de vecinos». La aplicación web está destinada al uso en comunidades de vecinos donde haya un presidente o encargado en la gestión de pagos pendientes de los comuneros. Esta aplicación permitirá visualizar información como la contabilidad de pagos de cada comunero, las viviendas existentes en la comunidad y los comuneros en sí, de manera rápida y efectiva, permitiendo una gestión más eficiente en la comunidad.

El proyecto nace a partir de la idea del cliente y tutor del trabajo, D. Carmelo Cuenca, basándonos en problemas reales que tiene siendo el presidente de su comunidad, y los problemas que acarrea no tener un sistema automatizado con las herramientas adecuadas para la gestión económica de la comunidad.

1.1. Objetivos

Partiendo de este problema, el objetivo del proyecto es automatizar la gestión, facilitando el cumplimiento de tareas que tiene el presidente a la hora de realizar un seguimiento de los recibos de cada comunero, y que estos sepan de manera clara y concisa cuánto deben a la comunidad. La misión primaria es, por tanto, conseguir que los usuarios sean capaces de visualizar los movimientos de la cuenta corriente de la comunidad, y que la aplicación web tenga las herramientas necesarias para poder filtrar los movimientos por nombres, fechas, viviendas..., de manera que se pueda identificar rápidamente a quién le pertenece cada pago y saber cuánto debe cada contribuyente.

Con relación a esto, podemos describir algunas de las funciones iniciales que podrían hacer los comuneros siendo usuarios de esta aplicación:

- Poder ver todos los movimientos del extracto bancario.
- Filtrar estos movimientos por diversos criterios, para facilitar la búsqueda y ver la información de manera más eficaz.
- Viendo los movimientos, solicitar que un registro particular que haya se asocie a una vivienda, para que pueda liberar pagos pendientes.
- Ver todos los pagos pendientes que debe el usuario por medio de sus viviendas, empezando desde el más antiguo, plasmando cuánto debe y el motivo, que podría ser por la cuota de la vivienda o una derrama de la comunidad.
- Ver todos los movimientos que estén asociados a las viviendas, que en esencia es un historial de todos los movimientos de la vivienda.

Además, se dispondrá de un usuario administrador para la comunidad que será el presidente o encargado de la gestión de los movimientos. Solo debería haber un administrador, el cual gozará de más funcionalidades y tendrá una mayor responsabilidad, siendo principalmente:

- Acceso a las funciones básicas de crear, leer, actualizar y borrar (CRUD) extractos bancarios.
- CRUD de los movimientos de los extractos bancarios.
- Asociar a los usuarios las viviendas que corresponden.
- Poner los pagos pendientes que debe cada vivienda, que pueden ser distintas para cada vivienda.
- Ver las solicitudes de los usuarios con respecto a los movimientos y asignarlo de manera acorde.
- Poder subir los ficheros de los extractos bancarios a la aplicación web.

De cara al estudiante, los objetivos que se pretenden alcanzar al desarrollar esta aplicación son principalmente:

- Una introducción al desarrollo de aplicaciones web utilizando un framework actual y relevante.
- Poner en práctica y reforzar conocimientos que se han adquirido a lo largo de la formación académica sobre las metodologías de desarrollo del software.
- Una formación básica en el uso de servicios de la nube, tanto para el desarrollo de estos como en el despliegue de aplicaciones.

1.2. Planificación inicial

El documento de la propuesta del Trabajo de Fin de Título (TFT), incluido junto a la entrega de este documento, contiene la planificación inicial completa sobre las divisiones de las tareas y las competencias que han de cumplirse a lo largo del proyecto.
A modo de resumen del documento, podemos dividir la planificación inicial del proyecto en varios apartados:

1. Una fase de análisis y preparación inicial, donde el estudiante se familiarizará con las herramientas que se utilizarán y que no han aparecido durante la formación académica. Estas herramientas son principalmente **Ruby on Rails** (**RoR**), **Heroku**, y un entorno de desarrollo local preparada para el desarrollo con **RoR** en una máquina virtual.
   Durante esta fase de análisis, también se estudiarán los casos de usos de aplicaciones similares, como pueden ser aplicaciones de gestión de fincas o de gestión de viviendas para realizar una comparativa con la aplicación que se quiere desarrollar.
2. Una vez realizado el análisis procederemos al diseño, desarrollo e implementación del proyecto. La implementación se realizará utilizando una metodología ágil basada en un desarrollo iterativa e incremental, utilizando algunas ideas

y filosofías de metodologías como Scrum o la metodología test-driven development (TDD), que se explicará con detenimiento en el siguiente apartado. Al realizarse con un proceso iterativo e incremental, la siguiente fase, en la que constan las pruebas, se vería integrada en el desarrollo de la aplicación en cierta medida.

3. Finalizado el desarrollo e implementación, procederemos a evaluar la aplicación comprobando la funcionalidad correcta de la misma. La fase de pruebas estará integrada en medida de lo posible dentro de la implementación de la aplicación, con el uso de test para cubrir las clases y acciones de esta.

4. Finalmente, se elabora este documento cuya estructura se podrá ver en apartados posteriores.

1.3. Metodología

Como se ha mencionado en la planificación inicial, utilizaremos elementos de varias metodologías a la hora de desarrollar el sitio web, entre las que se encuentran principalmente un desarrollo iterativo e incremental, Scrum y en la medida de lo posible, "Test-Drive Development" (Desarrollo orientado a pruebas o TDD).

El desarrollo iterativo e incremental fue una metodología creada para combatir las ineficiencias que se encontraban en el modelo tradicional en cascada. La idea principal consiste en desarrollar un sistema mediante el uso de ciclos o iteraciones, en una franja menor de tiempo, permitiendo aprender de lo desarrollado durante el ciclo para utilizarlo en el siguiente. Este proceso nos permite ver los requerimientos del proyecto desde una edad temprana, haciendo que evolucione desde la implementación de un conjunto de requisitos simple inicial hasta llegar al producto final, ya que tras cada iteración se añaden modificaciones del diseño y nuevas funcionalidades. La idea de utilizar iteraciones es lo que aplicaremos a nuestro proyecto, centrándonos en objetivos fundamentales en cada ciclo para poder ir creando la aplicación gradualmente y aprendiendo de los errores de la iteración anterior.

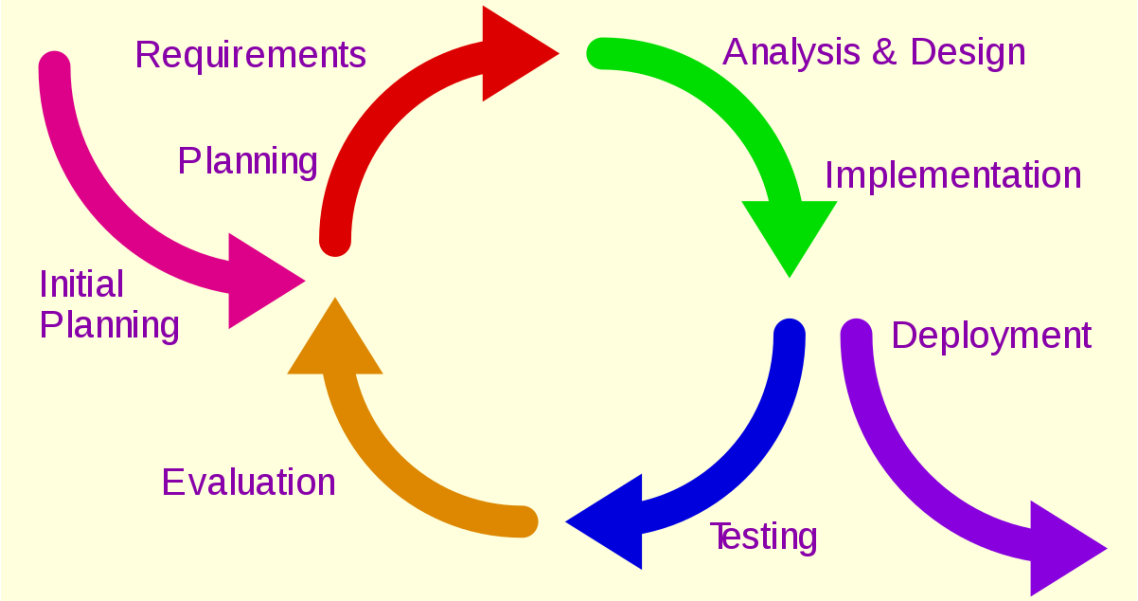

Figura 1: Ciclo de un desarrollo iterativo-incremental

1.3.1. Scrum

Hemos mencionado también la metodología Scrum, que es un subconjunto de las metodologías de desarrollo ágil y del desarrollo iterativo e incremental. Las ideas claves sobre las que se basa Scrum son el reconocimiento de que los clientes cambiarán de idea de lo que quieren o necesitan y que siempre habrá desafíos desconocidos, algo para lo que un enfoque predictivo o planeado no funciona muy bien. En consecuencia, Scrum adopta un enfoque empírico, aceptando que el problema no siempre se podrá entender del todo ni se podrá definir desde el principio, y dirigiendo los recursos a mejorar las habilidades de un equipo autoorganizado para realizar entregas tempranas, además de responder a requisitos que aparezcan y poder adaptarse a un mercado con tecnologías cambiantes. Scrum por tanto se puede resumir en los siguientes postulados:

- Metodología (reduciendo carga de gestión)
- Sistema iterativo e incremental
- Requisitos indefinidos y cambiantes
- Orientado a las personas
- Trabajo priorizado por el cliente
- Entrega continua y temprana
- Autogestión y autoorganización de los equipos
- Visibilidad del progreso

Scrum define tres principales roles, **Product Owner**, o propietario del producto; **Development Team**, o equipo de desarrollo y el **Scrum master** o facilitador de Scrum.

El **product owner** representa los intereses de los **stakeholders** del proyecto y es la voz del cliente. Asimismo, es el responsable de que el equipo de desarrollo entregue valor al negocio. Define el producto en un conjunto de historias de usuario y después los añade al **product backlog**, priorizándolos según su nivel de importancia en el proyecto.

Por otro lado, el equipo de desarrollo es el responsable de entregar un producto potencialmente terminado tras cada iteración o sprint realizado, siendo este la meta de la iteración. Un equipo ideal de Scrum debería componerse de entre tres y nueve miembros, encargados de realizar todas las tareas para producir el incremento del producto, siendo autoorganizados con respecto a las tareas de cada uno.

Finalmente, tenemos el **scrum master**, responsable de eliminar cualquier impedimento hacia el equipo de desarrollo para que pueda entregar el producto. Actúa como intermediario entre el equipo y todas las posibles distracciones a este, y se encarga de que el framework de scrum se esté cumpliendo.

Los elementos y artefactos principales que componen Scrum se pueden ver en la figura siguiente:

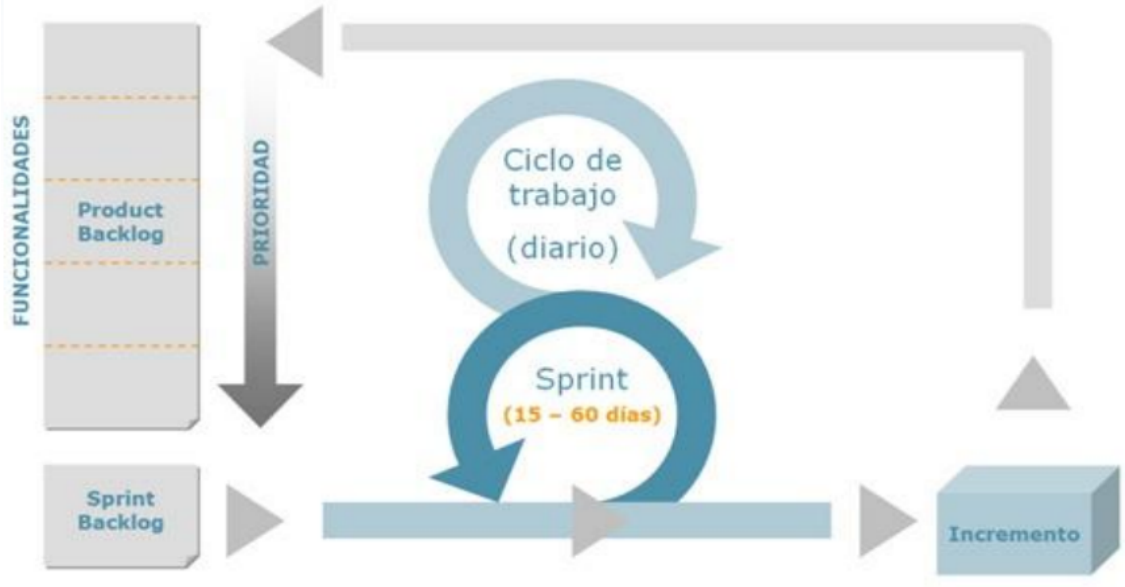

Figura 2: Artefactos utilizados en Scrum

Que se pueden dividir en:

- **Sprint**: la iteración en sí, la duración de este se decide al comienzo del proyecto y debería ser fijo para el resto del proyecto. Cada sprint empieza con una reunión para decidir las historias de usuario que compondran el sprint backlog para la iteración, y acaba con una revisión y retroinspección del sprint. Durante el día a día, el sprint empieza con un daily meeting, que no debería durar más

de 15 minutos, donde el equipo habla para decidir las prioridades del día y el progreso que hicieron en el día anterior.

- **Product Backlog:** es una lista de historias de usuario del proyecto, ordenadas según una prioridad que establece el propietario del producto y con un nivel de esfuerzo aproximado según la magnitud de la historia de usuario.
- **Sprint Backlog:** son aquellas historias de usuario que se tratarán en la siguiente iteración. Tiene un límite de la cantidad de esfuerzo que puede contener. Tras cada sprint, el equipo puede aprender el esfuerzo real que han realizado para poder reasesorar el esfuerzo necesario para las historias de usuarios futuros, refinando con más precisión la cantidad de trabajo que se debe poner en cada sprint.
- **Incremento:** tras terminar un sprint, se realiza un incremento del producto, que debería ser un producto funcional con las características previas y las nuevas que hubiera en el sprint backlog de la iteración.

De cara a nuestro proyecto, es difícil aplicar Scrum siendo una sola persona. No obstante, podemos coger elementos de Scrum para organizar nuestro proyecto, como es el uso de un product backlog, donde iremos estimando el esfuerzo de las historias de usuario en base a **Story points**, que son los puntos de valor de cada historia de usuario. En la Subsección 3.5 de desarrollo, explicaremos mejor el sistema que hemos utilizado para el desarrollo de la aplicación. Además, utilizaremos iteraciones de una semana de duración donde iremos introduciendo las historias de usuarios principales a desarrollar durante la iteración.

Las historias de usuario que se definen inicialmente en el product backlog son provenientes de los requisitos que se definen en la Sección 2, además de los que surgen durante el desarrollo de la aplicación relacionados a nivel de programación.

1.3.2. Test-Driven development

El TDD es una técnica o metodología ágil cuyas primeras formas fueron descritas en las declaraciones de Extreme Programming (XP) por Kent Beck. Dentro de XP, la fase de pruebas utiliza un concepto basado en la creación de los test primero y posteriormente el código (Test-first development o TFD), que es sobre la que se basa TDD.

La creación de los test primero permite al desarrollador tener claro cuáles son los requerimientos de la funcionalidad que se quiere implementar, ya que son necesarias para poder crear test unitarios que vayan probando pequeños segmentos de la característica a implementar, además del hecho de ir asentando una red de seguridad en el código. Estos test, al ser unitarios, permiten un feedback inmediato, teniendo claro cuándo hemos acabado, que es cuando todos los test que hemos escritos pasen.

Basándose en esto, TDD resulta en un ciclo de desarrollo con cinco partes:

1. **Añadir un test:** el desarrollador tiene que tener claro qué es lo que se quiere implementar para poder añadir un test particular, ya sea mirando los casos

de usos o las historias de usuario, creando una obligación a centrarse en los requerimientos de la funcionalidad antes de escribir código.

2. **Correr los test y ver que el nuevo test falla:** el nuevo test debe fallar, ya que aún no hemos implementado código alguno. Este paso verifica que el framework de testeo utilizado funciona, además de confirmar que el test no pasa sin motivo alguno.

3. **Escribir el código:** en esta fase escribimos código que pueda pasar el test, independientemente de si está escrito con la mejor calidad posible, ya que eso se refina en el último paso.

4. **Correr los test:** tras escribir el código, todos los tests, incluyendo los que estaban anteriormente, deben pasar. Si no pasa, o se rompe alguna otra parte del código, reescribiremos el código hasta solucionarlo.

5. **Refactorización:** por último, refactorizar el código base, que irá creciendo con cada test que validemos. Esto puede implicar tareas como limpiar el código, mover el código a lugares más relevantes dentro del proyecto, renombrar clases y variables a unos más apropiados, eliminar código duplicado... dejando un código final más legible y mantenible.

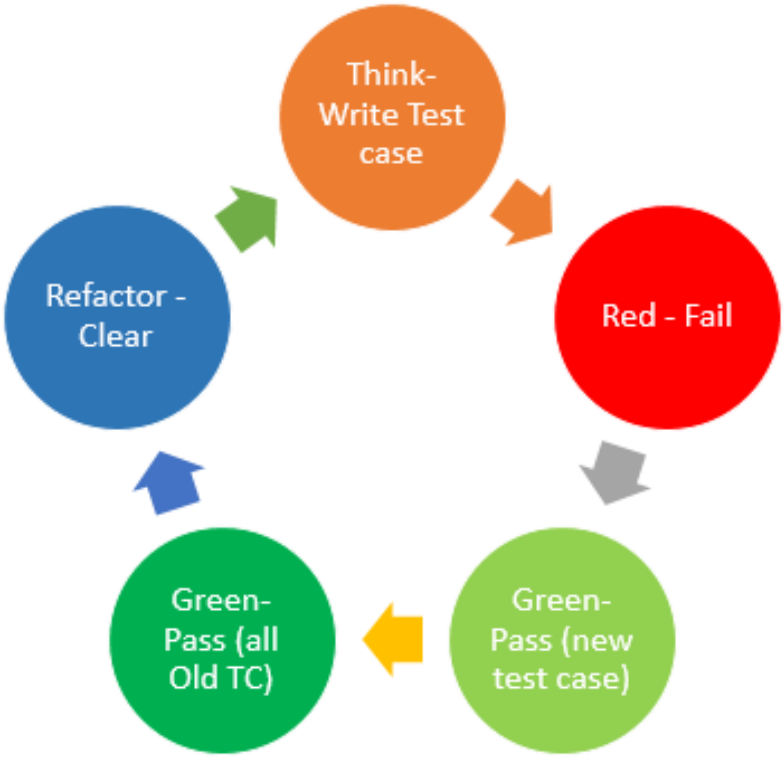

Figura 3: Ciclo del Test-Driven development

En nuestro proyecto, lo ideal sería incorporar en la medida de lo posible esta metodología durante el desarrollo. De esa manera, podríamos garantizar al terminar un sprint que toda la funcionalidad implementada está probada mediante test automatizados, disminuyendo los recursos necesarios para la prueba de la aplicación. No obstante, dado a la falta de experiencia del desarrollador a la hora de aplicar con rigurosidad la metodología y la experiencia necesaria para poder definir con precisión y antelación los test necesarios, no se ha podido seguir una definición de TDD en la aplicación, aunque si se han descrito algunos test automatizados que se verán en la sección de Subsección 3.6.

1.4. Contenido de la memoria

Con el espectro del problema comentado en la Sección 1 de esta memoria, aquí escribiremos lo que se puede encontrar en las siguientes páginas del documento.

En primer lugar empezaremos con Sección 2, análisis, donde explicaremos la idea del proyecto en su totalidad, teniendo en cuenta que se acabará limitando el alcance de este a un nivel apropiado para la realización de un trabajo de fin de grado. Para ello, en la Subsección 2.1 realizaremos un estudio inicial de aplicaciones similares, donde podremos ver sus puntos fuertes y débiles, además de características críticas que estos puedan tener en común. Compararemos nuestra aplicación con la competencia para ver las diferencias en la Subsubsección 2.1.4. Al ver algunas de las características que la competencia tenga en común, definiremos nuestros requisitos y usuarios potenciales en Subsección 2.2, teniendo en cuenta qué es lo que hace la competencia para los usuarios a los que quieren apelar. Con los requisitos, pasaremos a enseñar los diagramas de casos de uso (Subsección 2.3) que se utilizarán en el proyecto, haciendo especial hincapié en aquellos que estén dentro del alcance de la aplicación web. Con los casos de usos también crearemos los mockups de la aplicación y mostraremos como es la idea inicial de las páginas principales que hay en nuestra aplicación. Finalmente, para acabar el bloque de análisis se describirá el modelo de negocio que utilizaremos para la aplicación, además de tener en cuenta los aspectos legales que deben ser considerados en el desarrollo del sitio web(Subsección 2.5, Subsección 2.6).

En la Sección 3, desarrollo, se describirá el alcance del proyecto al que hemos ido haciendo referencia durante el análisis, que es lo que permitirá al estudiante adquirir conocimientos básicos de las tecnologías implicadas. Se enumerarán todas las características que se han implementado para la aplicación en la Subsección 3.1, basado en las ideas principales del análisis y diseño. Después, describiremos el diseño arquitectónico que se ha implementado y las principales clases que se han implementado, así como los principales módulos, responsabilidades y las relaciones entre sí (Subsección 3.2). También se describirá los componentes principales del modelo de la base de datos en la Subsección 3.3, haciendo uso de diagramas entidad-relación. Por otro lado, recogemos todas las tecnologías (Subsección 3.4) que están implicadas en el desarrollo de la aplicación y comentaremos brevemente su función dentro del proyecto. Mostraremos también las pruebas (Subsección 3.6) que se han realizado para la aplicación. Los test realizados reflejarán la mayoría de los requisitos que hemos definido y que hemos ido desarrollando. Por último, comentaremos el despliegue de

la aplicación en la Subsección 3.7, que se encuentra alojada en **Heroku**, tecnología que se comentará más adelante en el documento.

Una vez hecho el análisis y el desarrollo, en la Sección 4 sacaremos las conclusiones del proyecto tras haber cumplido todos los objetivos propuestos en el desarrollo. Se podrán ver las conclusiones del alumno para cada fase del recorrido, además de posibles mejoras que pudiera tener la aplicación. El alumno también comentará lo que ha aprendido y lo que le ha supuesto realizar el trabajo.

Dentro de la documentación (Sección 5) se podrán encontrar todos los recursos que se han utilizado para el desarrollo del proyecto, al igual que las referencias utilizadas.

Para finalizar, tendremos los anexos en la Sección 6, donde están adjuntados los documentos pertinentes, como pudiera ser el código fuente, los flujos de diagramas de casos de uso u otros elementos que fueran demasiados pesados para el documento principal.

2. Análisis

La idea es crear una aplicación de gestión económica de una comunidad de vecinos, dando especial atención a la parte de gestión económica, con la idea de hacer un proyecto que pudiera ser escalable a más comunidades y con gestiones de otros tipos. De primeras, los requisitos y características fundamentales que queremos tener son la vista de los movimientos bancarios y la gestión de cuotas de cada vivienda a la que pertenezca el usuario.

Para ello, haremos un análisis de aplicaciones similares, entre las cuales hemos escogido un total de 3 aplicaciones: **PropietariosOnline**, **PortalFincas**, **Fynkus**. Las primeras dos aplicaciones se han elegido utilizando palabras claves en Google y escogido aquellas con relevancia a lo que buscamos, y el último programa viene de un ranking de aplicaciones para la gestión de comunidades, utilizando como motivo de selección la popularidad entre los usuarios.

A continuación, veremos las características principales que ofrece cada una según se pueda ver en la página principal y cualquier tipo de documentación que se pueda acceder, además de algunas de las capturas de las aplicaciones para poder ver la interfaz y su usabilidad. Se comentará los precios o planes de pago que tengan las aplicaciones, y también veremos si disponen de información detallada acerca de todas las características que tengan disponibles. Las características que compararemos con nuestro proyecto serán algunas que consideramos más importantes para la implementación del prototipo, como:

- Poder importar extractos bancarios y visualizarlos en su totalidad.
- Crear pagos pendientes para cada vivienda.
- Liberar pagos pendientes de cada vivienda utilizando los movimientos de los extractos bancarios apropiados.

- Cada usuario puede visualizar información acerca de su vivienda y los pagos que tenga pendiente.
- Herramientas para asignar movimientos los movimientos a las viviendas de los usuarios.
- Poder filtrar los movimientos bancarios por determinados filtros, para poder visualizarlo de mejor manera.

Veremos si algunas de las aplicaciones que hemos escogido poseen esas características, además de comentar si hay algunas otras características que tengan en común con las demás aplicaciones sin haberlo tenido en cuenta nosotros.

2.1. Estudio de aplicaciones similares

En esta sección haremos un análisis de **PropietariosOnline**, **PortalFincas** y **Fynkus**, comentando algunos de los aspectos generales de cada aplicación y que puedan aportar a nuestro proyecto.

2.1.1. PropietariosOnline

PropietariosOnline	
Nombre de la aplicación	propietariosonline
Desarrollador	Advanced Programming Solutions
Versión	2.0
Uso	Sitio web y aplicación móvil
Fecha de visita	2018
Popularidad/Relevancia	Primeros resultados usando la búsqueda gestión online comunidades

Cuadro 1: Ficha técnica básica de **PropietariosOnline**

PropietariosOnline es una aplicación web para la gestión de comunidades de propietarios y su mantenimiento, ofreciendo también una versión móvil. La documentación que ofrece la aplicación es muy extensa y explica claramente qué es lo que se puede hacer con cada característica y las limitaciones que trae, por ejemplo, para poder crear un recibo, debes de haber generado una remesa de antemano.

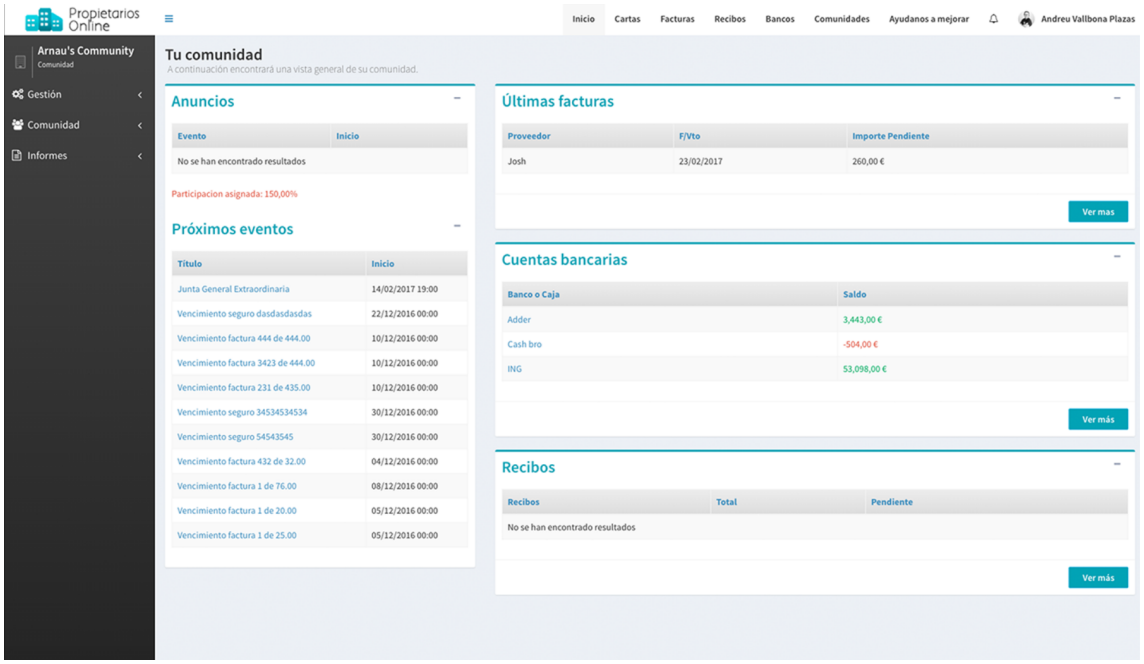

Figura 4: Pantalla principal de la aplicación **PropietariosOnline**

Las características básicas que tiene la aplicación que nos son de relevancia son:

- Facturas: añadir y pagar facturas.
- Remesas: generar, modificar remesas.
- Recibos: cobrar recibos.
- Presupuestos: crear presupuestos y control presupuestario.
- Informes: listar morosos, listar consumo, estado de cuentas, informes mensuales, anuales y trimestrales.
- Gestionar más de una comunidad: añadir comunidad, cambiar de comunidades.

En la documentación disponible de **PropietariosOnline** se puede ver claramente y bien agrupadas el resto de características y distinto tipos de gestiones de manera más detallada.

En cuanto a fortalezas de la aplicación, lo que más me ha agradado ha sido la existencia de una documentación o tutorial explicativo donde se ve claramente lo que hay que hacer para empezar y cómo hay que hacerlo. Además, cuenta con vídeotutoriales para ayudar de manera más visual a realizar las configuraciones iniciales y cómo utilizar determinados aspectos de la aplicación. Tiene capacidad de ser utilizado en web y móvil. Considerando que los administradores tienen más información visual disponible que los usuarios, es bastante bueno tener acceso a las dos vías.

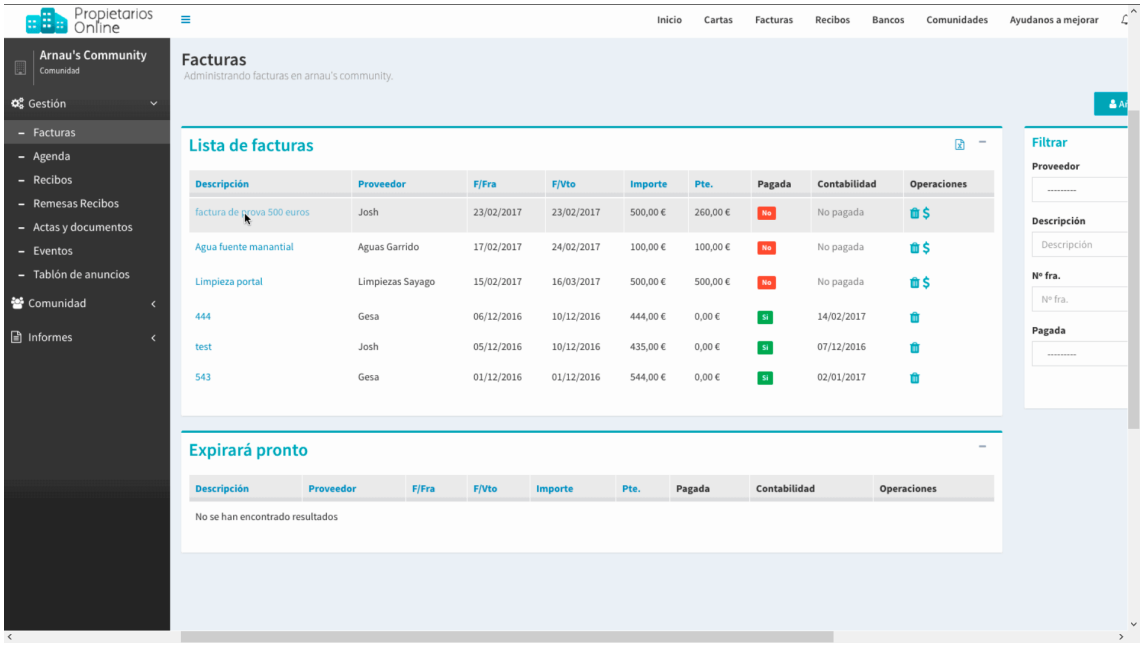

Figura 5: Página de recibos de **PropietariosOnline**

Con respecto a nuestro problema, permite generar informes de morosos por las viviendas cuyos recibos han vencido y aún no se han pagado, lo que puede ser algo bueno a tener en cuenta para implementar en nuestras funcionalidades.

PropietariosOnline tiene un plan de pago mensual, empezando con una prueba gratuita. El pago empieza a partir de 12 euros y va aumentando en función del número de propiedades se vayan incluyendo. Un punto negativo en este aspecto es la falta de transparencia, ya que no te dice con claridad cuánto va aumentando el precio mensual ni qué factores exactamente son los que hacen aumentar los precios.

Una de las características interesantes y a tener en cuenta para nuestra aplicación es la capacidad de que una propiedad tenga varios propietarios, que es el caso contrario a lo que nos ocurre, teniendo un comunero que tiene varias viviendas. En el caso de **PropietariosOnline** se podrá suponer que un mismo usuario podría ponerse con varias viviendas y se generaría sin problema un recibo por cada uno.

2.1.2. PortalFincas

PortalFincas	
Nombre de la aplicación	portalfincas
Desarrollador	Aner
Versión	No se encuentra publicado en las páginas
Uso	Sitio web, aplicación online
Fecha de visita	2018
Popularidad/Relevancia	Primeros resultados usando la búsqueda gestión fincas online y variantes

Cuadro 2: Ficha técnica básica de **PortalFincas**

A continuación tenemos **PortalFincas**, sitio web que va dirigido a los administradores para la gestión de cuotas, propiedades y propietarios. A diferencia de **PropietariosOnline**, la aplicación parece que tiene una visión más amplia al nivel de gestión, ya que está más orientada a la facilitación de gestión de varias comunidades. No dispone de documentación abierta, pero tiene cursos para la formación en el uso de la aplicación, ya que tiene como clientes objetivos a Autónomos/particulares que se encarguen de la gestión de comunidades y fincas, empresas y administradores colegiados.

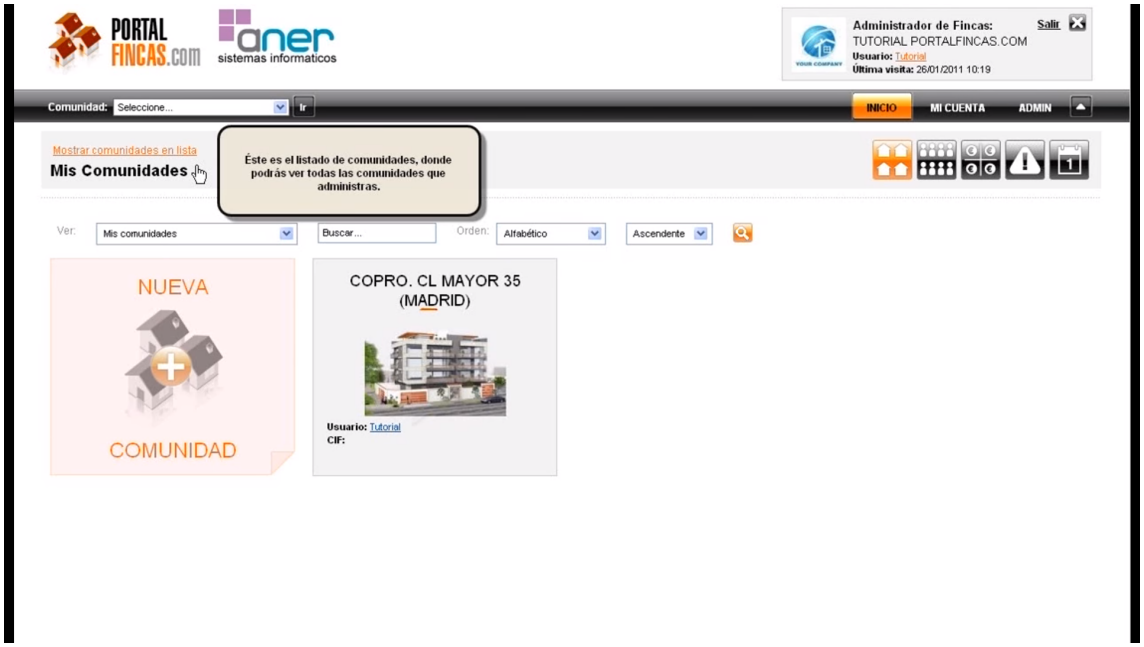

Figura 6: Pantalla inicial de la aplicación **PortalFincas**

Entre las características que nos interesan que ofrece, dispone de:

- Gestión de cuotas de la comunidad:
  - Cuotas periódicas basadas en un presupuesto anual.
  - Recibos de liquidación basándose en los movimientos de los propietarios.
  - Cuotas por adelanto por obras o específicas a una propiedad.
- Gestión de propietarios de comunidades para el administrador de fincas:
  - Datos personales del propietario.
  - Dirección postal.
  - Asignación de propiedad, que puede tener más de uno si es preciso.
  - Porcentaje de participación.
  - Cuenta corriente.
  - Cargo en la junta.

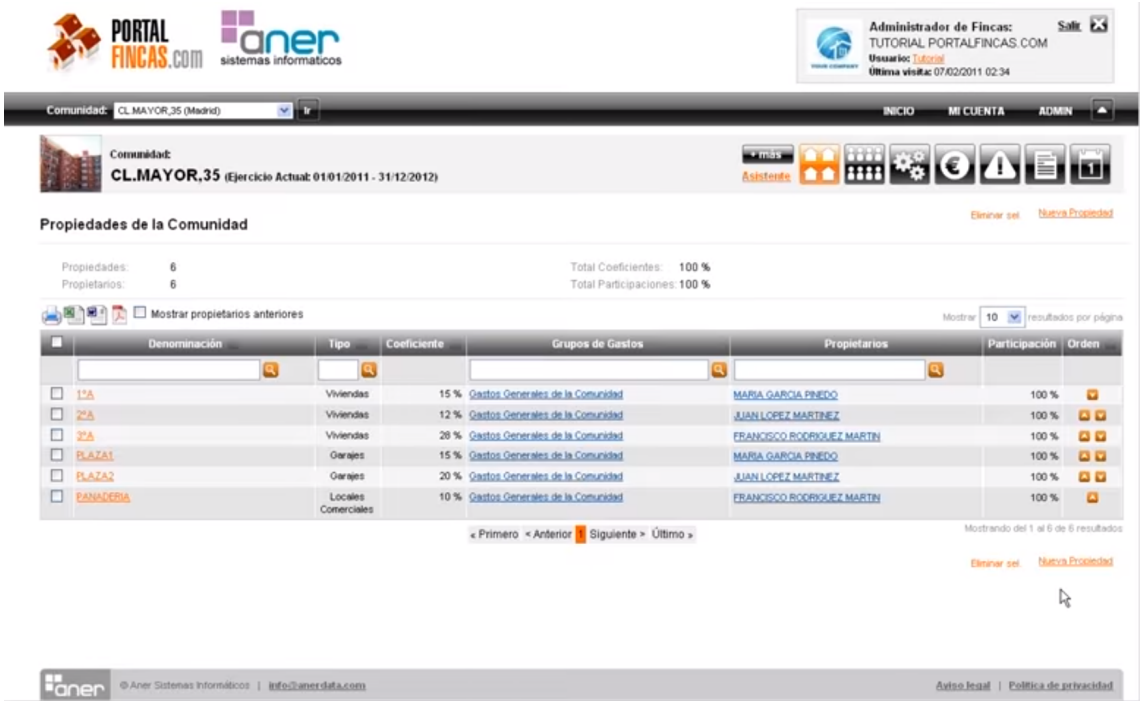

Figura 7: Pantalla de gestión de propietarios o usuarios en **PortalFincas**

La principal debilidad que se podría percibir es el simple hecho de que sea necesario ofrecer formación para el uso de la aplicación, pudiendo implicar que no es tan fácil de usar y se trata de un programa orientado a un uso más profesional, como puede ser una empresa, más que a un particular de una comunidad de vecinos. Por otro lado, parece que las características que ofrece esta son más completas que las de **PropietariosOnline**, siendo más versátil y con una mayor capacidad de personalización de los campos dentro de las funcionalidades que el administrador desee tener. También dispone de la capacidad de gestionar juntas, lo cual puede ser interesante para comunidades más grandes.

El plan de pago está claramente estructurado, además de contar con una prueba inicial de 30 días para probar la aplicación en cualquiera de las tarifas. La única diferencia entre los planes de pago son el nº máximo de propiedades y usuarios que tiene el plan, ya que las características a las que tiene acceso cada plan son los mismos, lo cual es bastante flexible según las necesidades de cada comunidad.

Una de las características del que aún no nos habíamos percatado ni pensado era los pagos por adelantado, ya que en nuestra situación, algunos de nuestros comuneros pagan varios meses juntos del tirón. Esta funcionalidad se podría tener en cuenta para poder llevar cuentas independientemente de las formas de realizar pagos que pudieran tener los usuarios.

2.1.3. Fynkus

Fynkus	
Nombre de la aplicación	Fynkus
Desarrollador	Fintech Driver
Versión	2.3
Uso	Aplicación móvil
SO	Android 4.1+ / IOS 8.0+
Fecha de visita	2018
Popularidad/Relevancia	Escogido de una página de ranking de aplicaciones de gestión de comunidades
NºDescargas	Más de 500

Cuadro 3: Ficha técnica básica de **Fynkus**

Por último, tenemos una aplicación que encontramos en un "ranking de apps" para gestión de comunidades. Tiene una división clara entre lo que puede hacer los vecinos y el administrador de las fincas y está más orientado a incluir ambos tipos, a diferencia de **PortalFincas**. La página tiene una simulación de cómo serían las pantallas para cada usuario, algo novedoso que ninguna de las otras dos aplicaciones que hemos comentado ha proporcionado. Pese a que no sea una característica directamente de la aplicación, permite al usuario potencial ver y probar la aplicación sin tener que descargar y solicitar una demostración.

Entre las características principales de la aplicación tenemos el saldo de la comunidad y de la propiedad, pudiendo consultar:

- Saldo y próximo recibo de la comunidad
- Ingresos y detalle de los gastos actualizados.
- Evolución del saldo.
- Morosidad.
- Contratos y póliza del seguro.
- Actas de las juntas.

Figura 8: Pantalla del gestor **Fynkus**

A diferencia de **PortalFincas**, la aplicación es muy gráfica y visualmente estimulante, dándote información relevante de la manera menos tediosa posible. No obstante, al priorizar este aspecto, quita muchos niveles de personalización y no permite realizar muchas de las acciones que se pudieran hacer con los otros dos programas. En este aspecto, veo que es una aplicación buena para un miembro de la comunidad cuando todo esté previamente configurado, pero quizás no tan útil para el administrador o presidente a la hora de realizar la gestión. Además, no tiene claro cómo gestionan los usuarios los movimientos que les correspondan ni las capacidades que tiene el administrador con características relacionadas a esta.

Figura 9: Pantalla de movimientos en **Fynkus**

Uno de los puntos más fuertes de la aplicación es que es gratis, teniendo como fuentes de ingresos publicidad. Esto es muy importante ya que no supone un gasto adicional a la comunidad, y probar una aplicación sabiendo que será siempre gratuita es muy atractivo, sobre todo cuando muchos comuneros no quieren tener que estar constantemente detrás de las ocurrencias en la comunidad con un gasto adicional.

Con respecto a nuestro proyecto, se podría tener en cuenta la visibilidad que tiene la aplicación para mostrar información, sobre todo para usuarios peculiares que de por sí hacen lo que quieren a la hora de realizar el pago. Tener la información de manera gráfica puede ayudar al usuario a tener claro qué es lo

que tiene que hacer siguiente o asegurarnos de que la información le llegue adecuadamente.

2.1.4. Comparativa con nuestro prototipo

Una vez estudiadas las características que tienen algunos de nuestros competidores, pasaremos a ver si tienen algunas de las características que habíamos mencionado al inicio del análisis. Algo a tener en cuenta antes de la comparativa general, es que para las importaciones de datos bancarios para ver los movimientos, nuestra aplicación no se conectará directamente con una cuenta corriente como hacen algunas de las aplicaciones, si no que añadiremos un extracto de los movimientos que queramos. Dicho esto, la tabla comparativa es el siguiente:

Características	**Aplicaciones**			
	Prop.onli	**port.finc**	**Fynkus**	**Prototipo**
Crear pagos pendientes para cada vivienda	Si	Si	No	Si
Liberar pagos con movimientos asociados a viviendas	Si	Si	Si	Si
Ver información acerca de la vivienda y sus pagos	Si	Si	Si	Si
Importar extractos y asociar movimientos a viviendas	No	No	No	Si
Filtrar movimientos bancarios por características	Si	Si	No	Si

Cuadro 4: Tabla comparativa de características

Podemos observar en la tabla que la funcionalidad que nos diferencia de primeras es la importación de extractos y manipulación directa de los movimientos a viviendas. Esto puede ser debido a que la escala inicial de nuestra aplicación es menor, ya que las aplicaciones que hemos visto pueden ser utilizadas para la gestión de varias comunidades y distinto tipos de gestiones. En el alcance actual de nuestra aplicación, podemos destacar, por tanto, que la habilidad de importar el extracto y poder trabajar directamente sobre los movimientos generados nos ayuda a cumplir los objetivos de proporcionar más eficacia al administrador para la gestión económica de la comunidad de vecinos.

Podemos decir que nuestra aplicación es distinta de los competidores, aunque esto sea probablemente debido a la diferencia de alcance general como acabamos de mencionar, y al tratamiento de datos, utilizando ficheros CSV de los extractos bancarios, frente a la competencia.

2.2. Requisitos

Para poder establecer los requisitos que necesitará nuestro proyecto, debemos tener en cuenta quiénes son nuestros usuarios potenciales y las características relevantes que les definan acorde a nuestra aplicación.

Plantilla para identificar usuarios potenciales	
Nombre	
Edad	
Sexo	
Ocupación	
Estado civil	
Nº Hijos	
Nº habitantes en la misma vivienda	
Nº Propiedades en la comunidad	
Cargo en la comunidad	

Cuadro 5: Plantilla para identificar usuarios potenciales de la aplicación

Utilizando esta plantilla, podemos tener una idea de qué tipo de habitante tenemos dentro de la comunidad. El razonamiento de las características elegidas para preguntar a los usuarios son los siguientes:

- La edad de los usuarios puede tener impacto sobre el diseño final de la aplicación, ya que si tenemos a usuarios de edad avanzada, sería recomendable un tamaño y fuente de letra adecuada a la lectura.
- La ocupación, junto a la edad, nos puede dar una idea del poder adquisitivo que tenga la comunidad en general, adaptando nuestros planes de pago acordemente.
- El estado civil, número de hijos y las personas con las que vive en la misma vivienda nos permite hacer una idea de si deberíamos adecuar la aplicación para que las viviendas alberguen usuarios adicionales, y dependiendo de la media, cuántos deberíamos añadir.
- El número de propiedades de cada usuario influye en la decisión de qué relación tomar entre los usuarios y viviendas. En nuestro caso, sabemos de antemano que hay un comunero que posee varias viviendas, por lo que tenemos que ser capaces de adaptar la aplicación para cubrir esa necesidad.
- El cargo de la comunidad nos deja ver cuántos cargos distintos pudiera haber en la comunidad y cuánta relevancia podría esta tener dentro de la aplicación. Considerando que nosotros solo tendremos un usuario administrador, que sería el presidente o el encargado de la gestión, no tendremos que implementar funcionalidades extras para cubrir los demás cargos.

Basándonos en esto, los requisitos que formularemos serán basados en los roles de usuarios, diferenciándose entre los usuarios no identificados, los usuarios habituales y el rol de administrador. Para entender bien los requisitos, tenemos que definir de antemano las dos clases relacionadas a los extractos bancarios que utilizaremos para definir los requisitos, para poder posteriormente ver cómo se relacionan en los casos de uso.

- Extracto bancario: el extracto bancario se utilizará como un listado donde tendremos registros o movimientos. A la hora de importar el extracto en nuestra aplicación, se debería poder dividir la cantidad de un movimiento en varios en el caso de que el usuario tenga varias viviendas. Además, se debería poder asignar el movimiento directamente en la importación a la vivienda correspondiente. Todos estas acciones las tendrá que hacer el administrador de manera manual a la hora de importar el fichero.
- Movimiento bancario o registro: es cada transacción que tiene una cuenta bancaria. Estos movimientos serán la unidad del extracto bancario.
- Saldo de la vivienda: las viviendas dispondrían de un saldo que se generaría a partir de los movimientos bancarios que se le asignen para poder pagar los pagos que estén pendientes.

2.2.1. Rol usuario no identificado

Empezaremos identificando los requisitos de los usuarios que no se han identificado aún:

- Como usuario no identificado quiero poder iniciar sesión para utilizar la aplicación.
- Como usuario no identificado quiero poder solicitar una cuenta de usuario para poder hacer uso de la aplicación.

Los usuarios no identificados inicialmente no podrán realizar ninguna otra acción más que identificarse y solicitar una cuenta a un administrador. El administrador es el que se encargará de que los datos proporcionados por el usuario son los adecuados para validar la cuenta. Nos aseguramos de esta manera que nadie que no esté autorizado pueda utilizar la aplicación y consultar datos de la comunidad.

2.2.2. Rol usuario

Definimos a continuación las historias de usuario de los usuarios, que el administrador podrá utilizar además de las funcionalidades de gestión económica:

- Como usuario quiero poder ver todos los extractos bancarios que haya subido el administrador para poder buscar mis movimientos.

- Como usuario quiero poder ver los movimientos asociados a mi vivienda para tener constancia de ellos.
- Como usuario quiero poder ver los pagos pendientes asociados a mi vivienda.
- Como usuario quiero poder solicitar que un movimiento se asigne a uno de mi(s) vivienda(s).
- Como usuario quiero poder ver los datos de mi(s) vivienda(s).

2.2.3. Rol administrador

El rol administrador tendrá unos requisitos adicionales a los mencionados por el usuario, siendo estos:

- Como administrador quiero poder filtrar los extractos bancarios por fechas, nombres, concepto, cantidad, entre otros, para manejar mejor la información.
- Como administrador quiero poder añadir usuarios a viviendas para crear una asociación entre estos.
- Como administrador quiero poder asociar un movimiento bancario a una vivienda para poder limpiar pagos pendientes.
- Como administrador quiero poder asociar pagos pendientes a viviendas para que se sepa lo que tienen que pagar.
- Como administrador quiero poder calcular cuánto dinero ha pagado cada vivienda para después ver si deben dinero.
- Como administrador quiero poder ver la cuota que tiene cada vivienda asociada para poder calcular adecuadamente lo que deben.
- Como administrador quiero poder utilizar el saldo que tiene una vivienda para pagar un pago pendiente que tenga esta.
- Como administrador quiero poder dividir la cantidad de un movimiento en la importación de extractos bancarios para poder asignar la cantidad adecuada a una vivienda.

Para el cumplimiento de estos requisitos hay que tener en cuenta la peculiaridad de los comuneros, dado que hay algunos que no pagan varios meses, pagando después varios meses a veces hasta por adelantado; un comunero en particular, que es el que tiene varias viviendas, puede pagar de todas las combinaciones posibles que se nos ocurran, y nuestro sistema deberá poder contemplar este caso mediante la implementación del último requisito descrito.

Además de estos requisitos, se tendrá en cuenta las funcionalidades CRUD para los usuarios, las cuotas que se mencionaron en la introducción del documento, además de modificaciones de los datos en las viviendas:

- Como administrador quiero poder hacer operaciones CRUD en los usuarios de la comunidad.
- Como administrador quiero poder hacer operaciones CRUD en los pagos pendientes de las viviendas.
- Como administrador quiero poder hacer operaciones CRUD con los extractos bancarios.
- Como administrador quiero poder hacer operaciones CRUD con los movimientos de los extractos bancarios.
- Como administrador quiero poder modificar los datos de una vivienda para actualizarlo a la información más reciente posible.

2.2.4. Requisitos del sistema

Por último, tenemos algunos requisitos que el sistema tiene que cumplir para el correcto funcionamiento de la aplicación:

- Como sistema quiero poder controlar la subida de archivos repetidos para que no haya datos duplicados en la base de datos.
- Como sistema quiero tener un sistema de gestión de correos para poder enviar correos a los usuarios.

Las funcionalidades del gestor de correos se podrán ir refinando más a medida que vayamos desarrollando el proyecto y veamos qué información le podría resultar útil al usuario tener en algún lugar por escrito.

2.3. Diagramas de casos de uso

En las secciones próximas podremos ver los casos de uso creados en función de los requisitos que hemos definido en el apartado anterior. Los casos de usos al completo encuentra en el anexo, y por ahora, utilizaremos el primer nivel de profundidad del administrador y describiremos los casos de usos más importantes del usuario y administrador.

Todos las especificaciones de los diagramas de casos de uso se adjuntan a la entrega del documento, aunque incluiremos algunos de los más relevantes si se menciona en los diferentes roles de casos de uso.

2.3.1. Casos de uso del usuario no identificado

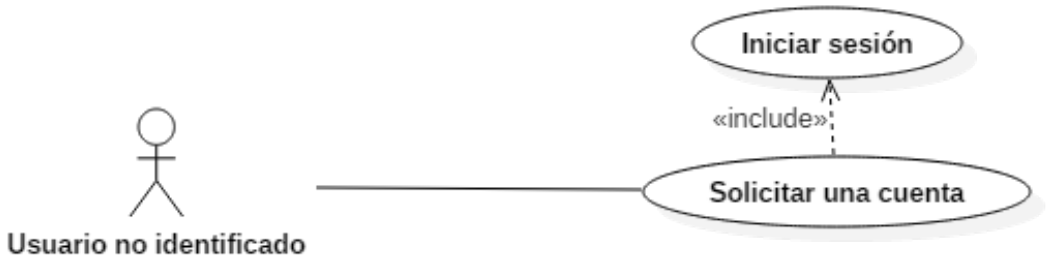

Figura 10: Diagrama de casos de uso del usuario no identificado

El principal caso de uso de los usuarios sin identificar es el que se puede ver en la Figura 10, donde el usuario podrá solicitar una cuenta al administrador para poder utilizar la aplicación. Es necesario recalcar que es una solicitud de cuenta y no una creación, ya que lo segundo implicaría que cualquier usuario podría crear una cuenta de usuario y utilizar el servicio, cuando solo queremos que los propietarios de la comunidad hagan uso de la aplicación.

CASO DE USO	**1**	**Solicitar una cuenta**		
Fecha		19-05-2018	**Fecha modif.**	
Descripción		El usuario no identificado solicita la creación de una cuenta		
Actores		Usuario no identificado		
Precondiciones		Estar en la pantalla de inicio de sesión		
Flujo normal		**Paso**	**Acción**	
		1	El usuario aprieta el botón de solicitar cuenta	
		2	Se rellena el formulario con los datos del usuario	
		3	Se envía el formulario al servidor y se espera hasta que un administrador dé de alta la cuenta	
Postcondiciones		El formulario se envía correctamente al servidor. El administrador recibe la solicitud y decide si crear la cuenta de usuario (Caso de uso 13) o rechazarla.		
Variaciones		**Paso**	**Acción**	
Extensiones		**Paso**	**Condición**	**Caso de uso**
Excepciones		Paso 3	El formulario no se envía y manda un mensaje de error al usuario y al servidor de que algo ha fallado	
Observaciones				

Figura 11: Especificación de caso de uso “solicitar cuenta”

2.3.2. Casos de uso del usuario

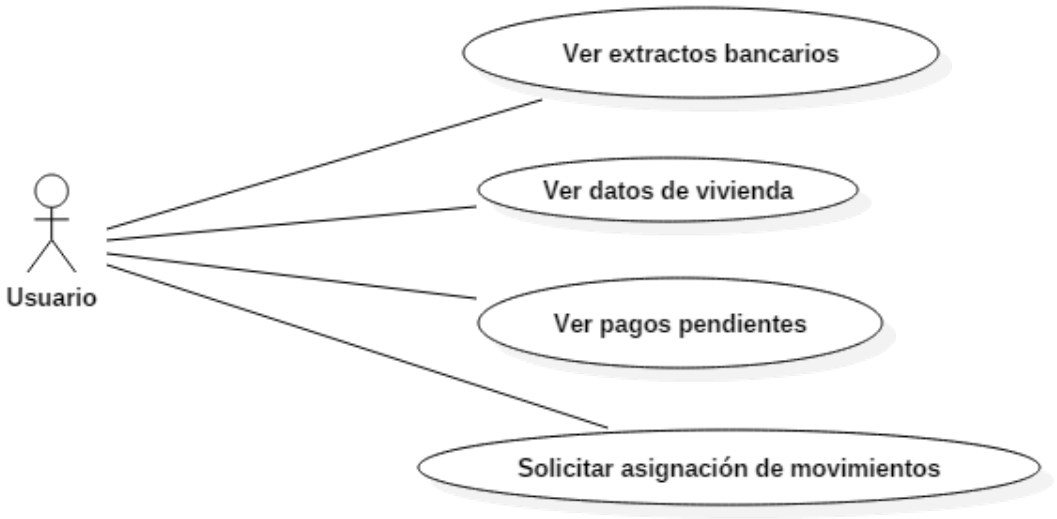

Figura 12: Diagrama de casos de uso del usuario

Los casos de usos del usuario son aquellos a los que tienen acceso los usuarios cuyas cuentas hayan sido validadas por el administrador. El administrador también tendrá acceso a las vistas de los usuarios normales. Entre los casos de uso que hay, cabe destacar la de "Solicitar asignación de movimiento", donde entre los extractos a los que tenga acceso el usuario, podrá solicitar que uno de los movimientos se le asigne a él por medio del administrador. A continuación podremos ver la especificación de este caso de uso.

CASO DE USO	6	Solicitar asignación de movimientos	
Fecha	19-05-2018	**Fecha modif.**	16-06-2018
Descripción	El usuario solicita que un movimiento se le asigne a una de sus viviendas de los extractos bancarios		
Actores	Usuario		
Precondiciones	El usuario ha iniciado sesión correctamente		
Flujo normal	**Paso**	**Acción**	
	1	El usuario entra y elige uno de los extractos bancarios, como está definido en el caso de uso 3	
	2	El usuario elige uno de los movimientos dentro del extracto y aprieta el botón de enviar solicitud de asignación, eligiendo la vivienda a la que quiere asignarlo.	
Postcondiciones	La solicitud se envía correctamente al administrador		
Variaciones	**Paso**	**Acción**	
	2	Antes de enviar la solicitud, puede añadir algún comentario que pudiera ser aclarativo del movimiento	
Extensiones	**Paso**	**Condición**	**Caso de uso**
Excepciones			
Observaciones			

Figura 13: Especificación de caso de uso "Solicitar asignación de movimientos"

2.3.3. Casos de uso del administrador

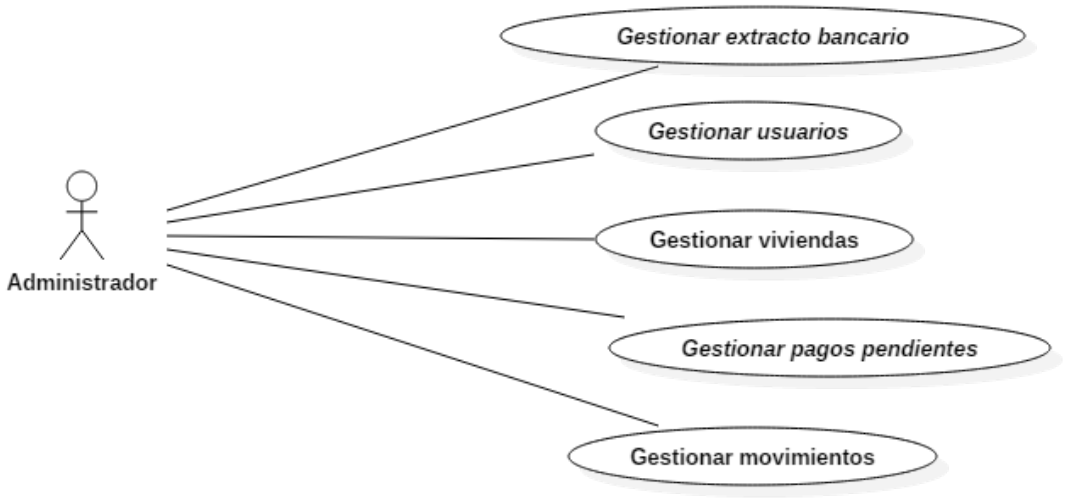

Figura 14: Primer nivel de diagrama de casos de uso del administrador

Como se puede observar en la Figura 14, los casos de usos principales de la aplicación reside en el rol de administrador. En la mayoría de las operaciones que puede realizar el administrador, las funcionalidades disponen de operaciones CRUD además de algunas otras. Estas operaciones adicionales son:

- **Gestionar extractos bancarios**
  - Filtrar extractos bancarios por nombre, fecha, búsqueda, concepto o cantidad
  - Importar extractos bancarios a la aplicación, permitiendo al administrador controlar si algún movimiento tiene mayor cantidad que una cuota normal y poder dividirlo y asignarlo a una vivienda.

CASO DE USO	**7**	**Crear extracto bancario**		
Fecha		20-05-2018	**Fecha modif.**	16-06-2018
Descripción		Crea un nuevo extracto bancario a partir de un fichero subido o uno en blanco		
Actores		Administrador		
Precondiciones		Inicio de sesión como administrador Si se crea a partir de un fichero subido, que haya un fichero subido previamente		
Flujo normal		**Paso**	**Acción**	
		1	El administrador se dirige al apartado de extractos bancarios	
		2	Aprieta el botón de importar un extracto bancario	
		3	Completa los pasos del caso de uso 12 de importar extracto bancario	
Postcondiciones		El nuevo extracto bancario tiene que figurar en la BD y los datos deben estar correctamente importados		
Variaciones		**Paso**	**Acción**	
		2	Se dirige a las opciones disponibles en la página del extracto bancario y elige “Crear nuevo extracto bancario”	
		3	Rellena los datos pertinentes al formulario que se genera para el nuevo extracto	
		4	Se envía los datos al sistema	
Extensiones		**Paso**	**Condición**	**Caso de uso**
Excepciones				
Observaciones				

Figura 15: Especificación de caso de uso “Crear extracto bancario”

CASO DE USO	12	Importar extractos bancarios		
Fecha		20-05-2018	**Fecha modif.**	16-06-2018
Descripción		Importar fichero de extracto bancario		
Actores		Administrador		
Precondiciones		Iniciado sesión como administrador Fichero a subir está en formato CSV con los mismos campos que los movimientos		
Flujo normal		**Paso**	**Acción**	
		1	El administrador se dirige a la zona de extractos bancarios y aprieta el botón de importar extractos	
		2	Se elige un fichero a subir	
Postcondiciones		El fichero se carga correctamente y se genera un extracto bancario con la información del fichero **Variación:** los nuevos movimientos que ha creado el administrador se crean de manera adecuada		
Variaciones		**Paso**	**Acción**	
		3		
Extensiones		**Paso**	**Condición**	**Caso de uso**
Excepciones		Paso 3	El fichero no tiene el formato adecuado o no tiene los mismos campos y se envía un mensaje de error al administrador para que lo solucione	
Observaciones				

Figura 16: Especificación de caso de uso "importar extracto bancario"

- **Gestionar usuarios**
  - Añadir un usuario a una vivienda, que le permitirá ver los pagos que tiene pendientes por la vivienda.

CASO DE USO	17	Añadir usuario a vivienda		
Fecha		20-05-2018	**Fecha modif.**	
Descripción		Se añade el usuario elegido a una vivienda		
Actores		Administrador		
Precondiciones		Iniciado sesión como administrador Haber elegido un usuario a añadir en el caso de uso 14 La vivienda no supere el límite de usuarios posibles establecido		
Flujo normal		**Paso**	**Acción**	
		1	Al seleccionar el botón de añadir usuario a una vivienda, se muestra un listado de las viviendas a añadir	
		2	Al elegir la vivienda, le saldrá una confirmación, que en caso de elegir "Sí", se actualiza la base de datos con la nueva asociación.	
Postcondiciones		La BD crea la nueva asociación entre usuario y vivienda		
Variaciones		**Paso**	**Acción**	
Extensiones		**Paso**	**Condición**	**Caso de uso**
Excepciones		2	Si el usuario a añadir fuera a superar el límite establecido por la vivienda, daría un mensaje de error y se redirige al administrador a donde estaba antes	
Observaciones				

Figura 17: Especificación de caso de uso "Añadir usuario a vivienda"

- **Gestionar pagos pendientes**
  - Asociar pagos pendientes a viviendas, por si tenemos que crear pagos pendientes aparte de los creados por la cuota de la vivienda y asignarlo a una vivienda.
  - Ver el historial de todos los pagos que ha tenido la vivienda, pudiendo seleccionar en la asignación del movimiento del extracto bancario qué usuario lo ha realizado para tener una constancia de los pagos realizados por el usuario.
  - Pagar un pago pendiente, utilizando el saldo de la vivienda, que se genera cuando asignamos un movimiento a la vivienda.

CASO DE USO	22	Crear pago pendiente		
Fecha		20-05-2018	**Fecha modif.**	16-06-2018
Descripción		Crea un nuevo pago pendiente		
Actores		Administrador		
Precondiciones		Iniciado sesión como administrador **Variación:** haber elegido una vivienda en el caso de uso 18		
Flujo normal		**Paso**	**Acción**	
		1	Se entra en el apartado de gestión de pagos pendientes	
		2	Se aprieta el botón de crear un nuevo pago pendiente	
		3	El administrador rellena los datos, en el cual se eligen las viviendas en las que se quiere dividir el pago pendiente	
		4	Se envían los datos al sistema	
Postcondiciones		El pago pendiente se crea en la BD con las relaciones correspondientes a las viviendas elegidas **Variación:** el pago pendiente se crea en la BD directamente con la relación con la vivienda elegida		
Variaciones		**Paso**	**Acción**	
		1	Dentro de la vivienda elegida seleccionamos crear nuevo pago pendiente	
		2	El administrador rellena los datos, en el cual la vivienda debería estar seleccionada ya	
		3	Envía los datos al sistema para que se cree el pago pendiente	
Extensiones		**Paso**	**Condición**	**Caso de uso**
Excepciones		Paso 3 **Variación** Paso 2	Los datos que envía el administrador no son válidos. En este caso se le muestra mensajes de error para que pueda corregirlos.	
Observaciones		Los datos que se crean en el pago pendiente son dependientes de información como la cuota asociada a la vivienda		

Figura 18: Especificación de caso de uso "Crear pago pendiente"

- **Gestionar movimientos**
  - Asignar movimientos a la vivienda, en el caso de que un usuario solicite que se asigne un movimiento en particular a una de sus viviendas.

CASO DE USO	32	Asignar movimiento a vivienda		
Fecha		20-05-2018	**Fecha modif.**	16-06-2018
Descripción		Añade el movimiento a una vivienda, aumentando su saldo		
Actores		Administrador		
Precondiciones		Iniciado sesión como administrador Haber elegido un movimiento a añadir en el caso de uso 29		
Flujo normal		**Paso**	**Acción**	
		1	Se aprieta el botón de asignar movimiento a una vivienda	
		2	Se envía la petición al servidor	
Postcondiciones		El movimiento está relacionado con la vivienda, y el saldo de la vivienda se actualiza **Variación:** la vivienda anterior se le descuenta el saldo adecuado, y en caso de quedarse en negativo, se le cancela el último pago pendiente que tuviera. La vivienda a la que se le asigna este movimiento se le actualiza el saldo		
Variaciones		**Paso**	**Acción**	
		1	En el caso de que el movimiento ya esté asignado a otra vivienda, se envía un mensaje de confirmación	
Extensiones		**Paso**	**Condición**	**Caso de uso**
Excepciones				
Observaciones				

Figura 19: Especificación de caso de uso "Asignar movimiento a usuario"

Una de las gestiones que no fue incluida en el listado anterior es la gestión de viviendas, ya que no proporcionaremos las operaciones CRUD al completo. Los casos de uso que tiene asociado la gestión de vivienda son:

- Ver vivienda.
- Actualizar una vivienda, aquí se podrá cambiar la cuota que tuviera o cambiar el titular de la vivienda.
- Quitar usuarios de vivienda.
- Quitar movimientos de la vivienda y asignárselos a otra vivienda. En el caso de realizar esta acción, se tendría en cuenta si al quitar el movimiento el saldo de la vivienda se quedaría en negativo. Si esto ocurre, se desmarcarían los pagos pendientes como pagados hasta que tuviera el saldo suficiente.

CASO DE USO	18	Ver vivienda		
Fecha		20-05-2018	**Fecha modif.**	
Descripción		Ver todas las viviendas que hay y la descripción detallada del que se elija		
Actores		Administrador		
Precondiciones		Iniciado sesión como administrador		
Flujo normal		**Paso**	**Acción**	
		1	El administrador entra en la sección de las viviendas y se muestra el listado de estos	
		2	Elige uno en el que entra en más detalle y puede ver la información de la vivienda elegida	
Postcondiciones		La vivienda se muestra correctamente con los datos de la BD		
Variaciones		**Paso**	**Acción**	
Extensiones		**Paso**	**Condición**	**Caso de uso**
Excepciones				
Observaciones				

Figura 20: Especificación de caso de uso "Ver vivienda"

2.4. Mockups y diseño de la interfaz

Con los casos de uso descritos, pasamos a la visualización de estos con unos esbozos de cómo sería la interfaz de nuestra aplicación. Hay que tener en cuenta que estos mockups no será la versión final, pero si se utilizará como guía para la versión final del prototipo. Todas las imágenes de mockups se añadirán en el anexo del documento, y aquí mostraremos algunas de las pantallas de los casos de uso más significativos del proyecto.

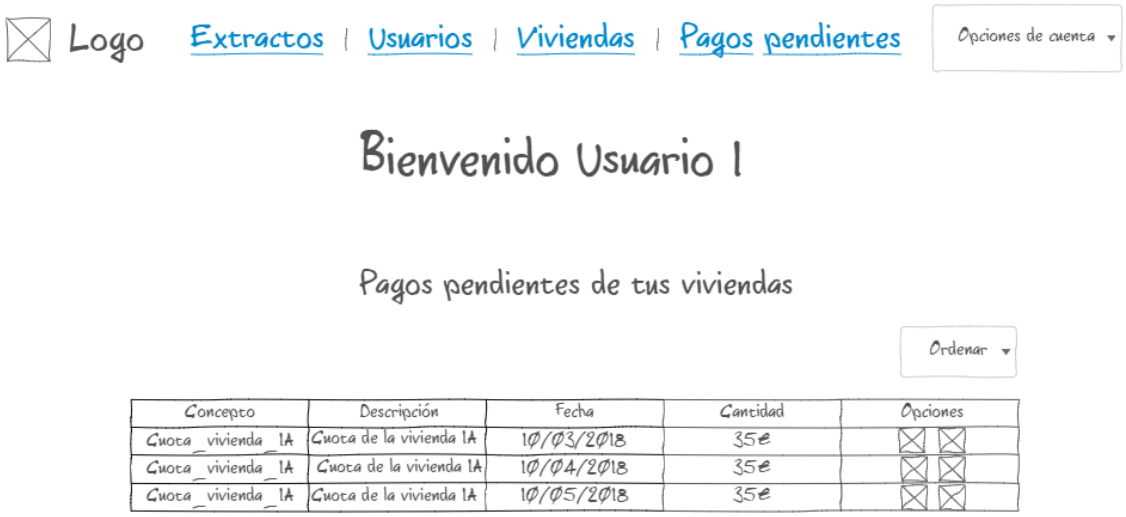

Figura 21: Pantalla principal de la aplicación

Tras pasar una pantalla de login inicial, la pantalla por defecto que verían los usuarios sería la pantalla de la Figura 21. En esta pantalla se le mostraría al usuario lo principal de la aplicación, ver que pagos tiene pendiente para poder pagarlas. Hay tener en cuenta que aunque el usuario pueda realizar estas acciones en el caso de que tenga el saldo suficiente, el administrador podría realizar todas las acciones necesarias para mantener los pagos pendientes actualizados en función de los ingresos en la cuenta corriente de la comunidad, pudiendo ser meramente informativo la aplicación para el usuario en el caso de que esto ocurriera.

Algunos elementos de la aplicación cambiarán según el rol con el que se inicie sesión, como por ejemplo, las opciones disponibles en la barra de navegación. Los menús disponibles para el administrador tendrán una finalidad más global, mientras que los usuarios tendrán disponibles en la navegación enlaces relacionados con ellos, como por ejemplo el enlace a usuarios, en vez de ver los usuarios de la comunidad, vería su perfil, mientras que el administrador podría ver todos los usuarios de la aplicación.

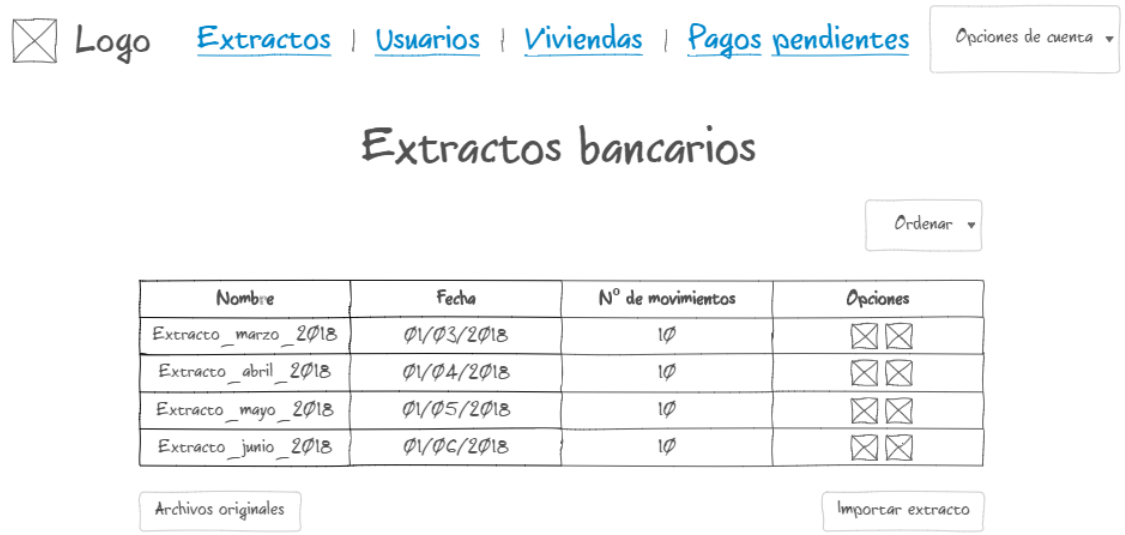

Figura 22: Pantalla de los extractos bancarios

En la siguiente pantalla tenemos los extractos bancarios, en este caso con la vista del administrador. Podemos ver todos los extractos bancarios, y tenemos disponible dos enlaces, uno a los archivos originales que ha subido el administrador, y otro para importar nuevos extractos bancarios. Además, podremos ordenar los extractos por fecha, nombre o número de movimientos. En las opciones tendremos las funcionalidades disponibles descritas en los casos de uso, como puede ser borrar el extracto bancario elegido.

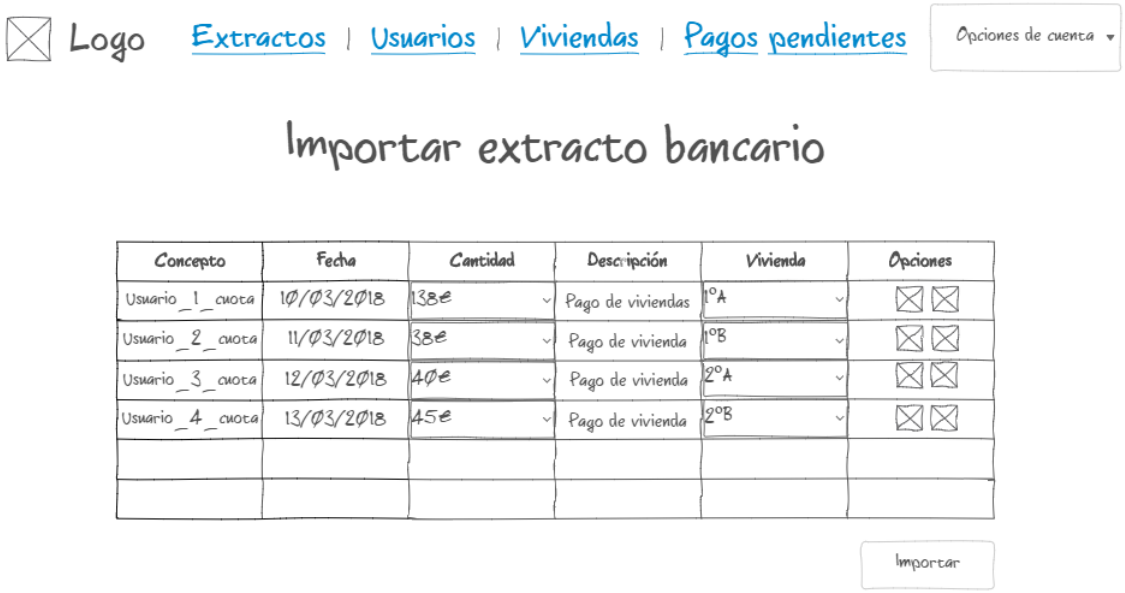

Figura 23: Pantalla de importar extracto bancario

En la importación de extracto bancario, podremos subir un archivo CSV que se transformará en una tabla como vemos en la Figura 23. Aquí el administrador podrá

ver todos los movimientos del extracto, y en caso de ver uno que no corresponde a una cuota normal (porque el usuario ha pagado de más, o ha pagado por más de una vivienda), podrá elegir dividir la cantidad en más registros, que copiará los campos del registro del cual se copia, y eligiendo la vivienda a la que asignar el nuevo registro. Al asignarse las viviendas e importar el extracto, los saldos de las viviendas afectadas se actualizarán con la cantidad ingresada. En caso de que no se elija ninguna vivienda porque no se tiene claro de quién es el ingreso, los usuarios que sí reconozcan el movimiento podrán realizar la solicitud de asignación a la vivienda correspondiente.

Figura 24: Pantalla del perfil del usuario

Figura 25: Pantalla del perfil de la vivienda

En la Figura 24 y la Figura 25 tenemos los perfiles de los usuarios y de las viviendas. En el perfil del usuario podemos ver las viviendas que tuviera asociado, al igual que datos del usuario. Desde aquí, podemos ir al perfil de la vivienda, donde tendremos información de la vivienda, como la cuota actual de la vivienda, la cantidad que debe la vivienda, o el saldo que tenemos disponible fruto de los movimientos asociados. El administrador desde el perfil de la vivienda puede editar los usuarios que pertenecen a la vivienda, al igual que pagar los pagos que estén pendientes desde la vista de pagos pendientes, que veremos en la Figura 26.

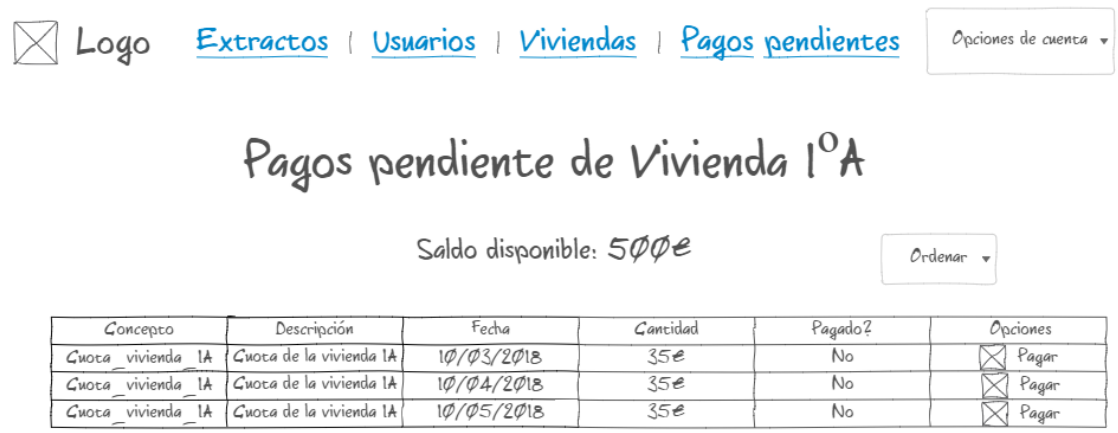

Figura 26: Pantalla de los pagos pendientes de una vivienda

Desde la vista de pagos pendientes de la vivienda podremos pagar los pagos pendientes si disponemos del saldo adecuado para ello. También se puede filtrar como en la mayoría de tablas de la aplicación, aunque solo se podrá pagar el pago pendiente más antiguo.

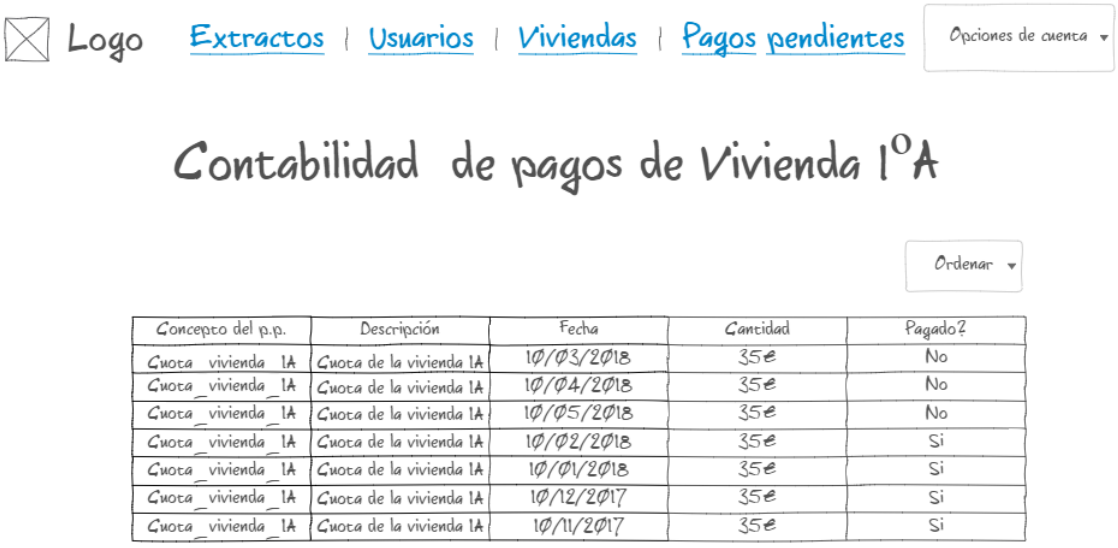

Figura 27: Pantalla de la contabilidad de pagos de una vivienda

Por último, hemos incluido una pantalla de la contabilidad de pagos de la vivienda, donde podemos ver todos los pagos asociados a la vivienda, estén pagadas o no. En el caso de incluir el cambio de titularidad de la vivienda, que trae consigo algunos requisitos, como la transferencia de deudas, podríamos modificar esta vista para acomodar esa funcionalidad, viendo qué usuario es el que ha pagado cada pago pendiente.

2.5. Modelo de negocio

A la hora de ver la competencia, comentamos brevemente los planos de pagos que tenía cada una de las aplicaciones, agrupándolas en dos planes de pago principales:

- Aplicación gratuita utilizando publicidad como medio de ingresos.
- Esquema de pagos dividido en varios niveles de pagos, ofreciendo una mayor cantidad de recursos disponible según el plan de pago.

Las aplicaciones gratuitas incentivan al usuario a utilizarlos, ya que no cuestan nada y la mayoría de publicidad puede ser ignorada sin problema. Esto podría funcionar cuando disponemos de muchos usuarios, o como mínimo si la aplicación es utilizada múltiples veces por los usuarios cada día. En nuestro caso, donde nuestro cliente principal sería el administrador para llevar la gestión y no tan dirigido a los comuneros para su uso diario, podría no ser óptimo elegir este modelo de negocio, ya que es probable que utilice la aplicación en momentos del mes que sepa que algunos comuneros vayan a pagar, o fechas concretas, y no diariamente.

Por otro lado, tenemos el esquema de pago escalable a las necesidades del usuario. Dentro del esquema, podemos tener pagos únicos o pagos por tiempo, y en ambos pagos existen varios niveles de pago en función de los servicios ofertados. Los pagos únicos tienden a tener precios a corto plazo más altos a diferencia de los pagos por tiempo, pero a diferencia de los pagos mensuales que puede acabar costando más que un pago único, en una gestión de comunidad de vecinos donde su uso podría ser casi vitalicio por poseer una propiedad, podría ser más atractivo para el cliente. No obstante, un pago único implica la necesidad de buscar nuevos clientes, cosa en la que nuestra aplicación a corto alcance no tiene en cuenta.

Por tanto, para hacer rentable nuestra aplicación en su estado inicial, aplicaríamos un modelo de negocio de pagos temporales, pudiendo cambiar de modelo si nuestra aplicación crece en algún momento.

2.6. Normativa y legislación

Considerando el hecho de que estamos tratando con datos reales de usuarios, una de las normativas que tenemos que tener en cuenta es el tratamiento de datos acorde a la ley de protección de datos. Dispondremos de algunas páginas informativas con la política de privacidad estándar y condiciones de uso a la hora de registrarse para

el uso de la aplicación. De cara al documento, añadiremos aquí tenemos un enlace para la Agencia Española de Protección de Datos (AGPD).

Respecto a la nueva legislación de protección de datos europea que entró en vigor en mayo de 2018, la legislación no se aplica al tratado de datos que tengan un uso de carácter personal. La legislación entraría en vigor en el momento en el que las actividades sobre la que se procesan los datos sean de carácter comercial o profesional. La aplicación de esta legislación por tanto dependería de si la aplicación se llega a comercializar, en cuyo caso debería aplicar los cambios necesarios para cumplir la nueva legislación.

3. Desarrollo

Una vez realizado el análisis de los requisitos necesarios, procederemos a la implementación de dichos requisitos, comentando el proceso de las distintas etapas en la creación del prototipo. Comenzaremos explicando el alcance de la aplicación propuesto y las características que se han implementado, seguido del diseño arquitectónico que sigue el proyecto además del modelo de la base de datos. Después, presentaremos brevemente toda la tecnología implicada en el desarrollo del prototipo, y entraremos en la etapa de desarrollo de iteraciones. Concluiremos la sección de desarrollo con las pruebas realizadas en el proyecto y el despliegue de la aplicación en la nube.

3.1. Alcance de la implementación

Como era de esperar, el prototipo desarrollado no lleva implementada todas las características que hemos mostrado en la sección de análisis (Sección 2), pero ha podido cubrir gran parte de las características propuestas, validando las principales ideas hechas en el análisis. Esto nos ha permitido a su vez adquirir conocimientos básicos sobre las tecnologías implicadas en el proyecto, sobre todo con **RoR**, que comentaremos con más profundidad en el diseño arquitectónico que sigue.

Entre las características principales que hemos implementado, cabe mencionar que las acciones de las funcionalidades son casi exclusivamente del administrador. Los usuarios dentro de la aplicación tienen permisos suficientes para ver determinadas partes de las funcionalidades, como los extractos generales que existen en la comunidad o datos que conciernen a sus viviendas. Las características que están disponibles en el prototipo son:

- **Gestión de extractos:** importar ficheros CSV con datos bancarios reales para crear extractos bancarios, que pueden ser actualizados y borrados. Los ficheros se alojan en cubos de S3 de Amazon Web Services (AWS), con los que podremos procesar los datos y crear en nuestra base de datos un extracto bancario con movimientos bancarios vinculados a este.
- **Gestión de movimientos:** tras realizar la importación de un extracto, se crean movimientos que pueden ser asociado a viviendas o borrados. Los mo-

vimientos no se pueden manipular. excepto cuando hay que dividir un movimiento para poder cubrir las cuotas de diversas viviendas con un solo ingreso, que es una de las funcionalidades descritas para un requisito de un vecino particular.

- **Gestión de viviendas:** el prototipo nos permite crear, ver, actualizar y borrar viviendas que se encuentren en nuestra aplicación. En función del saldo que tiene la vivienda, generada por los movimientos que tenga asociados, podemos pagar los pagos pendientes que haya a nombre de uno de los residentes de la vivienda. Pese a que las operaciones de crear y borrar viviendas no estaban incluidas en los requisitos iniciales, se han añadido en la aplicación al no saber de antemano el número de viviendas que hay en la comunidad en la que se vaya a utilizar.
- **Gestión de pagos pendientes:** podemos crear pagos pendientes para las viviendas en función de la cuota que tenga cada una, además de poder crear derramas para la comunidad con la capacidad de elegir los meses en los que se quiere dividir la cantidad total. Tenemos también la capacidad de actualizar los pagos pendientes, borrarlos, o incluso asociarlo a otra vivienda en algún caso particular que pudiera ocurrir dentro de la comunidad.
- **Gestión de usuarios:** el administrador cuenta también con la capacidad de asociar usuarios a viviendas, o nombrar determinados usuarios como propietario de una. Podrá también crear nuevos usuarios, borrar usuarios existentes y también validar las cuentas de los usuarios que soliciten una cuenta nueva en la aplicación.

Los usuarios que no sean administradores dentro de la aplicación podrán acceder a cada parte de las distintas gestiones, cuya vista está modificada para ver datos relacionados a sí mismo, y las acciones están restringidas al no ser administrador. Las vistas disponibles a los usuarios son:

- **Extractos bancarios:** el usuario podrá ver todos los extractos que haya subido el administrador a la aplicación, dando una transparencia a la comunidad al poder ver todas las transacciones.
- **Vivienda:** se listan todas las viviendas a las que está asociado el usuario, pudiendo ser la vivienda en la que reside o las viviendas que le pertenecen. Dentro de la vista de cada vivienda podrá ver información relacionada a esta, como los usuarios que residen en ella, los movimientos que están identificadas de la vivienda y una contabilidad de los pagos, mostrando pagos que estén pagados y pagos pendientes que tiene la vivienda.
- **Pagos pendientes:** se muestra un listado con los pagos pendientes de todas las viviendas a las que esté asociado un usuario, ya sea como residente o propietario.
- **Movimientos:** al igual que con los extractos bancarios, el usuario tiene la información de todos los movimientos generados por los extractos a su disposición.

- **Usuarios:** pese a que no puede realizar acciones activas en ninguna de las gestiones, el usuario podrá actualizar sus datos personales para mantenerlas al día, al igual que ver todos los usuarios que se encuentren en la vivienda con sus respectivos datos, por si tuviera que contactar con alguno.

El caso de uso de filtrar los extractos bancarios se implementó durante el desarrollo de manera general, es decir, todos los modelos tienen una implementación de filtrado en función de los atributos que se muestren en las pantallas.

Una de las funcionalidades del usuario que no se ha implementado ha sido la solicitud de movimientos a sus viviendas, que se contará con más detalle dentro de la sección de trabajo futuro a realizar.

Por otro lado, los usuarios que no hayan iniciado sesión podrán solicitar una cuenta en la aplicación. Esta cuenta que no está creada por el administrador requiere validación para poder acceder a las vistas del usuario normal.

3.2. Diseño arquitectónico

Ruby on Rails es conocido principalmente por seguir el patrón arquitectónico de Modelo-Vista-Controlador (MVC), y al utilizar este framework como vía principal de desarrollo, es el que se ha utilizado para nuestra aplicación.

El patrón MVC divide una aplicación en tres componentes:

- Modelo: encargado del manejo de datos, la lógica de negocio y reglas de la aplicación.
- Vista: manejo de la representación de los objetos y lo que ve el usuario.
- Controlador: manejo de la entrada de datos del usuario y lo transforma en acciones para el modelo o la vista.

Al realizar esta separación, las solicitudes del usuario siguen el siguiente proceso:

1. El navegador (cliente) envía una solicitud de una página al controlador del servidor.
2. El controlador coge la información que necesita del modelo para responder a la solicitud.
3. El controlador pasa los datos recibidos a la vista.
4. La vista se crea y se envía al cliente mediante la visualización del navegador.

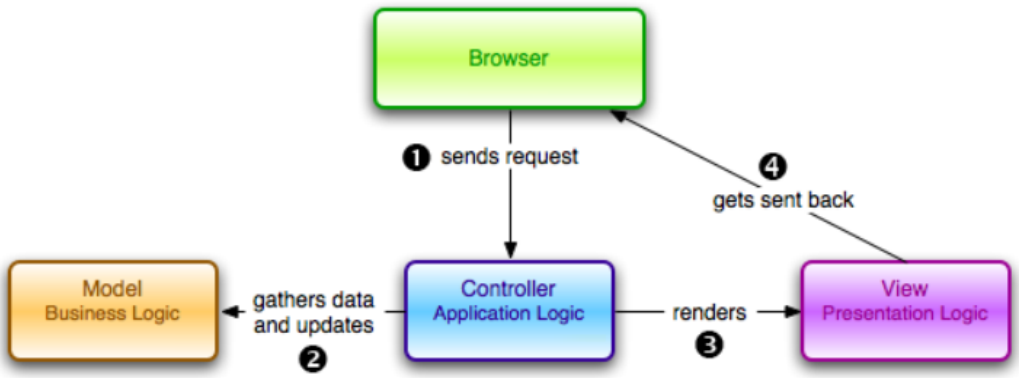

Figura 28: Proceso de una solicitud con MVC

Las principales ventajas de utilizar este patrón es que el proyecto pasa a tener poca dependencia entre los elementos, que permite desarrollar en los distintos componentes de manera paralela, y a su vez una alta cohesión, pudiendo agrupar las acciones relacionadas de un controlador en un mismo lugar. También aumenta la reusabilidad de los componentes, ya que un mismo modelo puede ser utilizado en varias vistas.

Ruby on Rails también sigue un paradigma de diseño software llamado "Convención sobre configuración" (convention over configuration) utilizado en muchos frameworks e introducido por David Heinmeier Hansson, creador de **RoR**. El objetivo principal es reducir la cantidad de decisiones que tiene que tomar el usuario debido a que hay establecido un estándar por defecto para la gran mayoría de acciones al iniciar un proyecto. Esto se puede tomar como ejemplo que al crear un modelo con nombre de "Usuario", la base de datos podrá crear automáticamente una tabla con el nombre de "usuario", sin necesidad de intervención del usuario. Dentro de **Ruby on Rails**, el diseño se aplica sobre todo a la creación del proyecto por defecto, haciendo que el desarrollador evite tener que escribir archivos de configuración determinando qué módulos del framework deberían cargar, que era algo muy común en los framework iniciales.

Rails genera una clase para cada componente del MVC de la cuál se heredan todas los modelos, vistas y controladores futuras. Estas clases son imprescindibles para que Rails pueda funcionar:

- **ActiveRecord (modelo):** es la capa del sistema encargada de representar los datos y lógica del negocio. Active Record facilita la creación y uso de los objetos del negocio cuyos datos requieren un almacenamiento persistente en la base de datos. Es a su vez una implementación del patrón de diseño "Active Record", que en sí es la descripción de un Mapeo objeto-relacional (ORM).
- **ActionController (controlador):** tras determinar desde la solicitud qué controlador se va a utilizar para cubrirla, el controlador se encarga de entender bien la solicitud y producir una respuesta adecuada. El controlador

hace la mayoría del trabajo mediante el uso de las convenciones, haciendo que sea lo más sencillo posible para el usuario.

- **ActionView (vista):** tras realizar las acciones necesarias desde el controlador, Action View se encarga de poder enviar esa respuesta al solicitante. En rails, las plantillas de Action View están escritas con Ruby incrustado en etiquetas mezclada con HTML. Para evitar el llenado de las plantillas con mucho código, se proporcionan clases auxiliares para elementos en común, como por ejemplo formularios.

Además de los tres componentes del MVC, RoR incluye otros componentes de arquitectura en su conjunto. Entre los componentes básicos podemos encontrar:

- **Servidor de Rails (Rails Server):** es un ejecutable que crea una instancia de un servidor web. Durante el desarrollo de aplicación, utilizar este componente nos permite acceder a la aplicación desde puerto 3000 de la dirección IP local (considerando que no lo hayamos modificado). El servidor se encarga de coger las solicitudes del usuario y de procesarlo desde la aplicación, además de escribir información relacionado a las solicitudes en registros.
- **Rutas (Routes):** las rutas en una manera de juntar las URLs que nuestra aplicación entiende y qué parte de la aplicación debería encargarse de manejar estas solicitudes. Para ello, el fichero "config/routes.rb" contiene las URLs que entiende a partir del URL base "/" junto al controlador y la acción necesaria para encargarse de la solicitud.
- **Bienes (Assets):** los bienes son todos los ficheros como código de JavaScript, CSS o imágenes que utilice la aplicación, complementando el contenido.

```
Rails.application.routes.draw do
  devise_for :users

  devise_scope :user do
    authenticated :user do
      root 'pending_payments#index'
    end

    unauthenticated do
      root 'devise/sessions#new', as: :unauthenticated_root
    end
  end

  # root                                                    to: 'static_pages#home'
  get '/help',                                            to: 'static_pages#help'
  get '/about',                                           to: 'static_pages#about'

  # User custom routes
  get '/profile/:id',                                     to: 'users#show', as: 'user_profile'
  get '/:id/users',                                       to: 'users#user_list', as: 'userlist'
  post '/users/create',                                   to: 'users#create', as: 'user_create'
  get '/users/:id/approve',                               to: 'users#approve', as: 'approve_user'

  # Statement custom routes
  get '/statements/bucket',                               to: 'statements#bucket', as: 'bucket'
```

Figura 29: Extracto del archivo "routes.rb" de la aplicación

Una vez comentado el aspecto teórico del MVC, continuaremos describiendo cómo aplica este patrón **Ruby on Rails** utilizando la estructura de nuestro proyecto como ejemplo para ver los elementos que se crean por defecto al utilizar "Convención sobre configuración" y MVC.

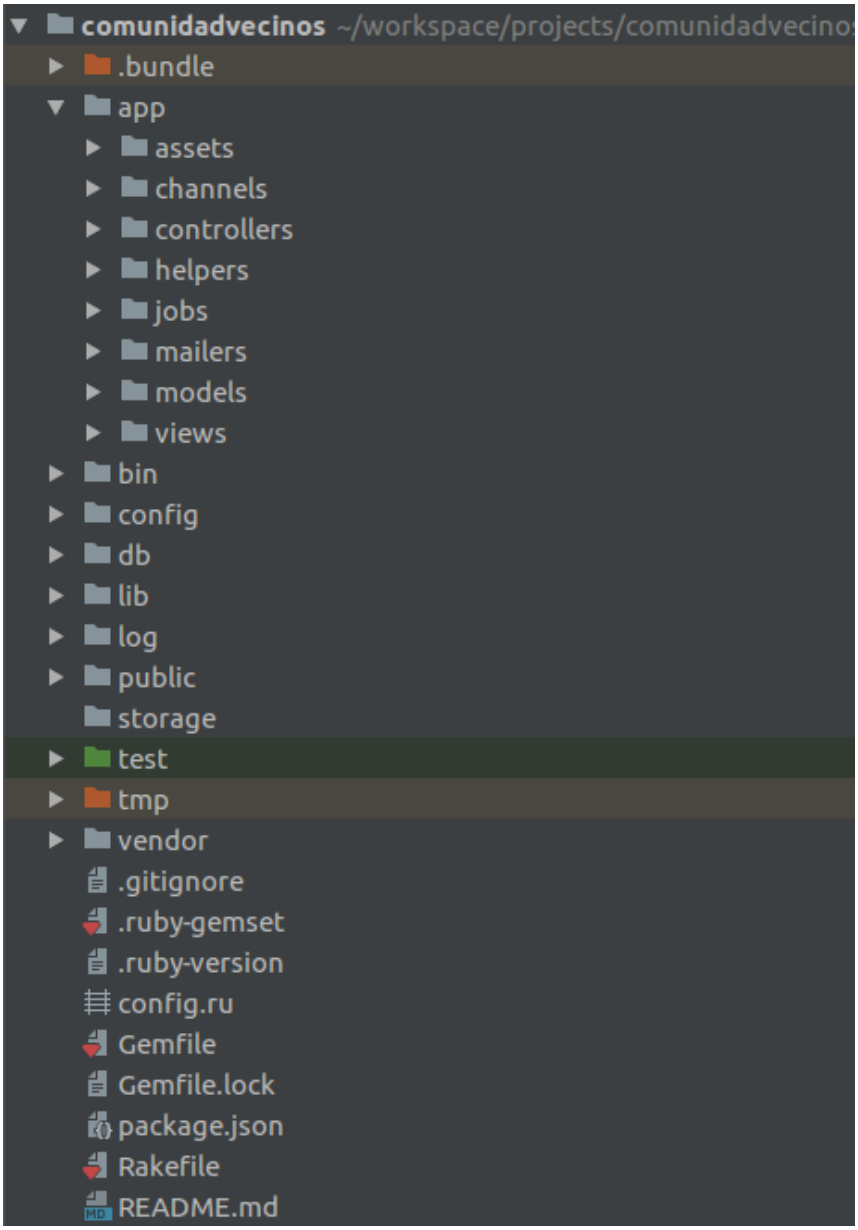

Figura 30: Estructura de nuestra aplicación en Rails

Al crear una aplicación utilizando RoR, este tendrá siempre la misma estructura por defecto, que solamente cambiará cuando el usuario quiera hacer algo que no sea convencional. La parte fundamental de las carpetas generadas, se encuentra en la carpeta de **app**, donde podemos ver las carpetas de **controllers**, **models** y **views**, respectivamente, para cada componente. En estas carpetas es donde desarrollamos la gran mayoría de la lógica de la aplicación, separando acordemente la lógica de la funcionalidad que queramos en la carpeta correspondiente.

En el caso de nuestra aplicación, las clases que se han implementado están en la carpeta de modelo, donde tenemos además sus relaciones definidas.

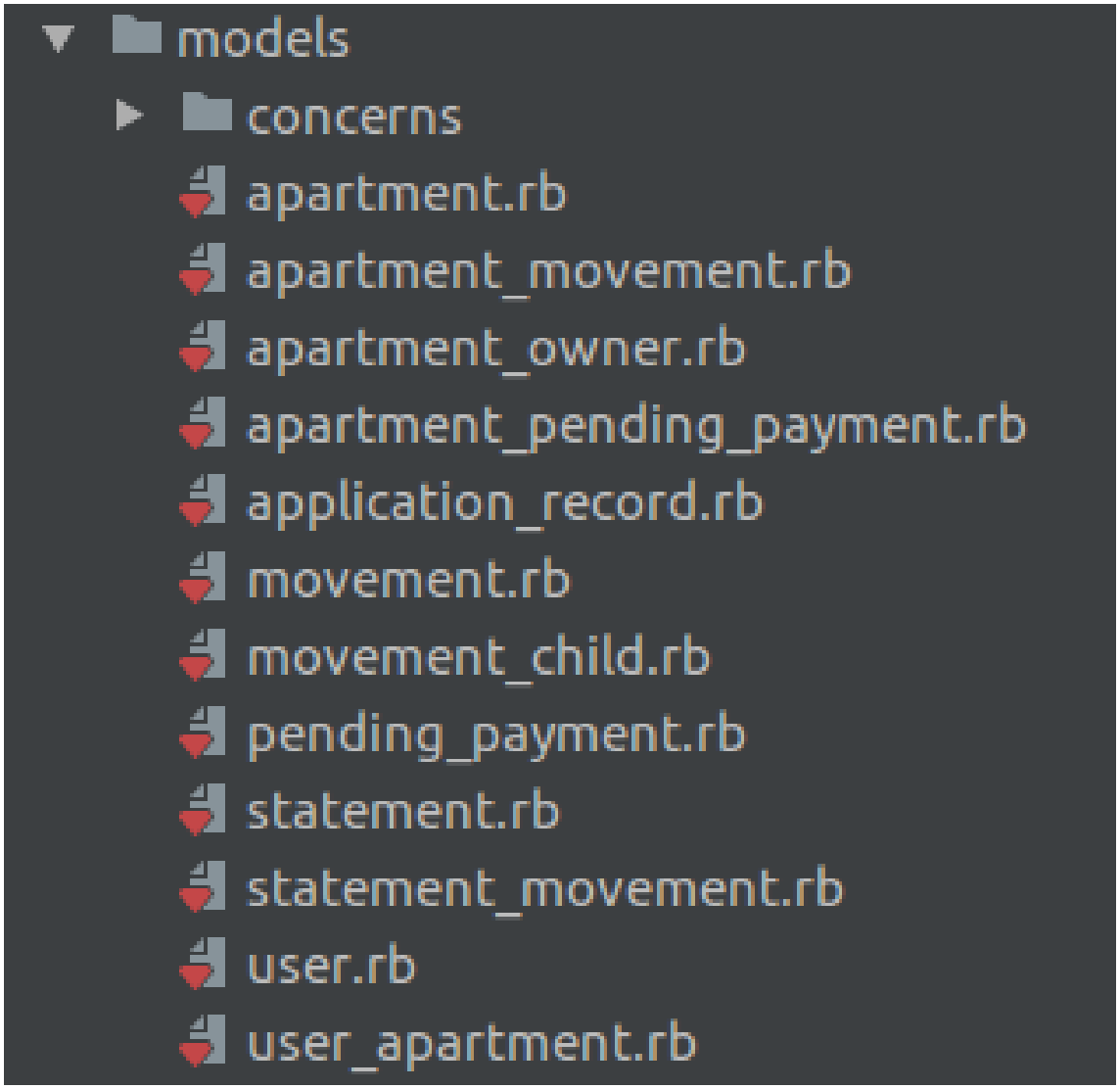

Figura 31: Modelos creados en la aplicación de rails

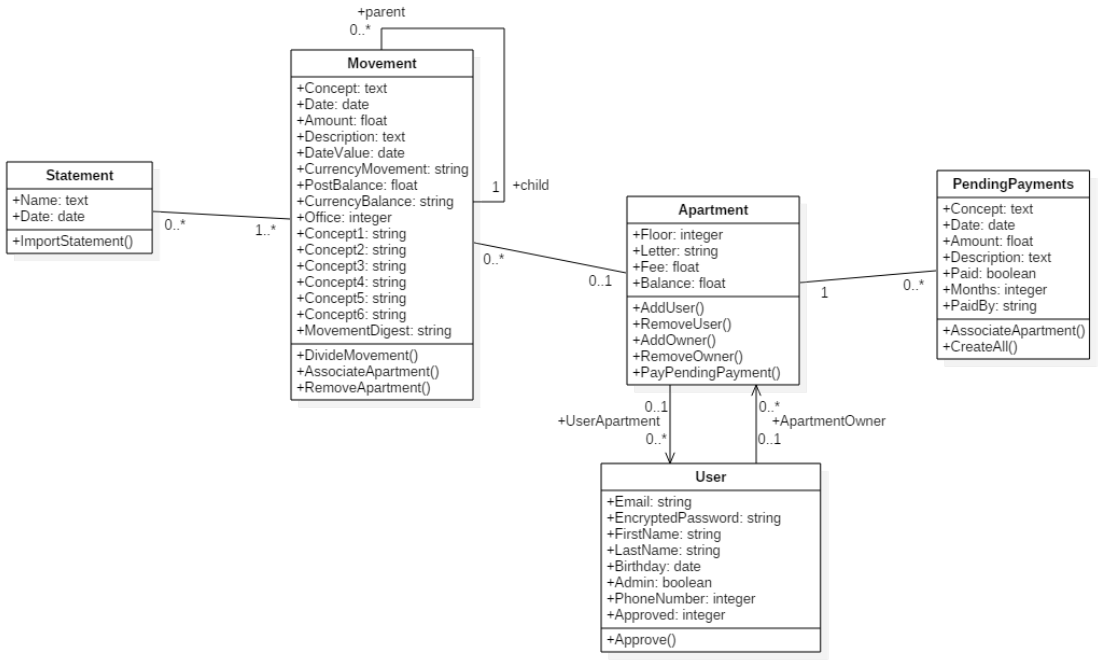

Figura 32: Diagrama de clases de la aplicación

En la Figura 31 podemos ver los modelos que existen en la aplicación, incluyendo los que se generan como relaciones entre modelos. En la Figura 32 tenemos el diagrama de clase con sus respectivos atributos. Todos los modelos de rails tienen un atributo implícito que se crea a la hora de generar un nuevo modelo: "Created_At" y "Updated_At". Estos atributos determinan cuándo se ha creado o actualizado una instancia de modelo. Dado que estos dos atributos están en todos los modelos, no lo hemos incluido en el diagrama de clases. Los modelos o clases principales que tenemos son:

- **Extracto bancario (statement)**: los extractos bancarios poseen únicamente dos atributos, el nombre del extracto y una fecha opcional para la importación del extracto. Esta fecha opcional está a disposición del administrador por si quiere importar extractos de años anteriores y que se vea reflejado de alguna manera en la aplicación.
- **Movimiento bancario (movement):** los movimientos tienen como atributos las columnas que tiene el extracto bancario de la comunidad a la que va dirigida principalmente la aplicación. Uno de los atributos a destacar es el de "Description", que es el único campo modificable del modelo, dirigido al administrador para poder escribir un comentario opcional sobre el movimiento. El atributo que nos permite combatir la duplicidad de los extractos bancarios es el de "MovementDigest", que se explicará en el apartado de relación de las entidades.
- **Vivienda (apartment):** el modelo vivienda posee atributos para identificar qué vivienda es. No puede haber dos viviendas con el mismo piso y letra en la comunidad, y para la simplificación del proyecto, la cuota de la vivienda es un

valor numérico que decide el administrador, pese a que debería ser calculado en función a variables reales como los metros cuadrados de la propiedad. Cabe destacar el atributo de saldo, que es el valor que utilizaremos para poder pagar pagos pendientes asociados a la vivienda. El saldo se calcula sumando los ingresos provenientes de los movimientos y restándolo posteriormente a los pagos que ya se han realizado en la vivienda. El saldo óptimo de una vivienda debería ser 0, que implicaría que no hay ningún pago pendiente de la vivienda por pagar y que todos los movimientos que hubiera asociado están siendo utilizados para realizar pagos.

- **Pago pendiente (pendingPayment):** los pagos pendientes se crean en función de algunos datos de las viviendas, principalmente la cuota que posean. El atributo "Months" se utiliza para la creación de pagos pendientes con prospecto de pagarse en varios plazos, como puede ser una derrama de la comunidad, que suelen ser inesperados y no todos los vecinos pueden tener el dinero que les corresponde disponible.

- **Usuario (user):** los atributos que se muestran en el diagrama de clase son las que se utilizan de manera primaria en la aplicación. El modelo user posee más atributos que se han generado mediante el uso de la gema "Devise", que se comentará e incluirá en la sección de tecnologías (Subsección 3.4). El rol de administrador de la aplicación la conseguimos gracias al atributo de "Admin", cuyo uso nos permite restringir acciones y vistas en función de si el usuario tiene el atributo activado o no.

3.3. Modelo de la base de datos (Diagrama entidad-relación)

Antes de describir nuestro diagrama de entidad-relación, hay que explicar cómo funcionan las relaciones entre las entidades porque el diagrama ha sido generada mediante la gema **Rails-ERD**.

3.3.1. Rails-ERD

Rails-ERD es una gema que crea automáticamente un diagrama entidad-relación basada en las relaciones que están establecidas en cada modelo de la aplicación, y para poder entender el diagrama del proyecto, utilizaremos los ejemplos que nos proporciona la documentación de la gema.

```ruby
class Country < ActiveRecord::Base
  # A country may or may not have a head of state.
  has_one :head_of_state
end


class HeadOfState < ActiveRecord::Base
  belongs_to :country

  # A head of state always belongs to a country.
  validates_presence_of :country
end
```

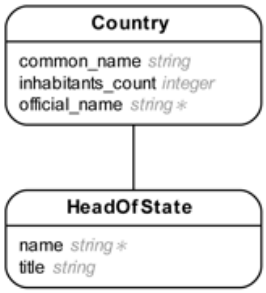

Figura 33: Ejemplo de relación "uno a uno" del ERD

La primera relación es "uno a uno", donde asociamos dos modelos entre sí, pudiendo tener o no una instancia relacionada del otro modelo. Se representa con una línea recta.

```ruby
class Galleon < ActiveRecord::Base
  # Galleons have up to 36 cannons.
  has_many :cannons

  validates_length_of :cannons, :maximum => 36
end


class Cannon < ActiveRecord::Base
  # A cannon belongs to a galleon.
  belongs_to :galleon
end
```

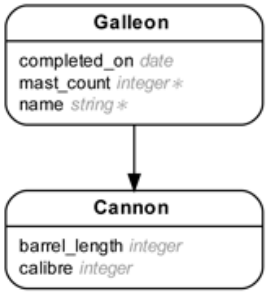

Figura 34: Ejemplo de relación "uno a muchos" del ERD

A continuación tenemos la relación "uno a muchos", con un modelo pudiendo tener más de una instancia del segundo modelo relacionado. Esta relación se representa con una flecha, donde la dirección de la flecha indica qué modelo es el que puede tener varias instancias del modelo al que apunta.

```ruby
# The two models are joined by the join table 'films_genres',
# with two foreign keys, 'film_id' and 'genre_id'.

class Film < ActiveRecord::Base
  # A film belongs to one or more genres.
  has_and_belongs_to_many :genres
end

class Genre < ActiveRecord::Base
  # Each genre may be applicable for one or more films.
  has_and_belongs_to_many :films
end
```

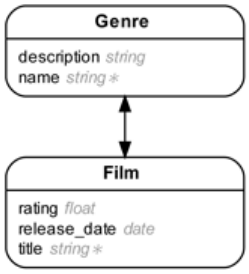

Figura 35: Ejemplo de relación "muchos a muchos" del ERD

En tercer lugar tenemos la relación “muchos a muchos”, representada por una flecha en ambas direcciones.

```ruby
class Wizard < ActiveRecord::Base
  has_many :spells, :through => :spell_masteries
  has_many :spell_masteries
end

class Spell < ActiveRecord::Base
  has_many :spell_masteries
end

class SpellMastery < ActiveRecord::Base
  belongs_to :wizard
  belongs_to :spell
end
```

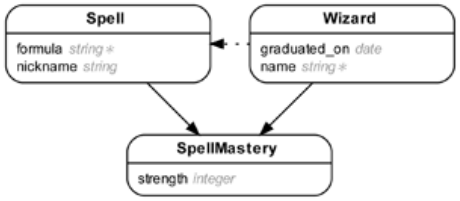

Figura 36: Ejemplo de relación “muchos a muchos” mediante una tabla intermedia

Por último, tenemos la relación “muchos a muchos” mediante el uso de una tabla conjunta, que es la representación primaria que utilizamos en nuestro diagrama para representar las relaciones “muchos a muchos”. Se representa con dos tipos de líneas, una línea discontinua entre los modelos, que sigue las mismas reglas que las figuras anteriores, y la línea continua de los modelos con la tabla intermedia.

3.3.2. Diagrama entidad-relación del proyecto

A continuación veremos los distintos apartados del diagrama de entidad-relación del proyecto. El diagrama completo se podrá encontrar en los anexos de la memoria.

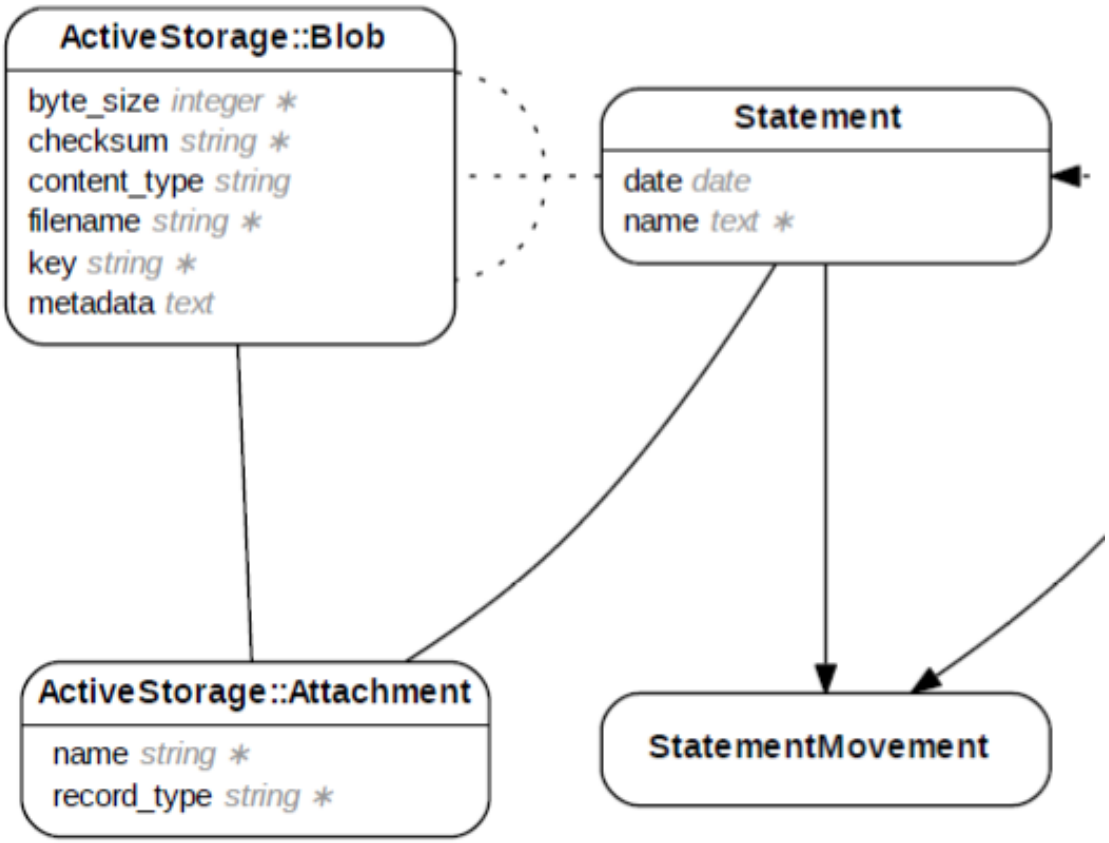

Figura 37: Relación de extractos con ficheros adjuntos

Lo que vemos en la Figura 37 es la relación que se crea entre el fichero CSV real y el extracto bancario como modelo que se crea en nuestra aplicación. Las entidades de **ActiveStorage::Blob** y **ActiveStorage::Attachment** no se han incluido en el diagrama de clases porque son clases propias de Rails, en especial de la versión 5.2, que es la última versión estable del framework. Son clases que nos permiten crear una asociación de un modelo con un objeto adjuntado, en nuestro caso el CSV, y con ello podemos subirlo directamente a la nube.

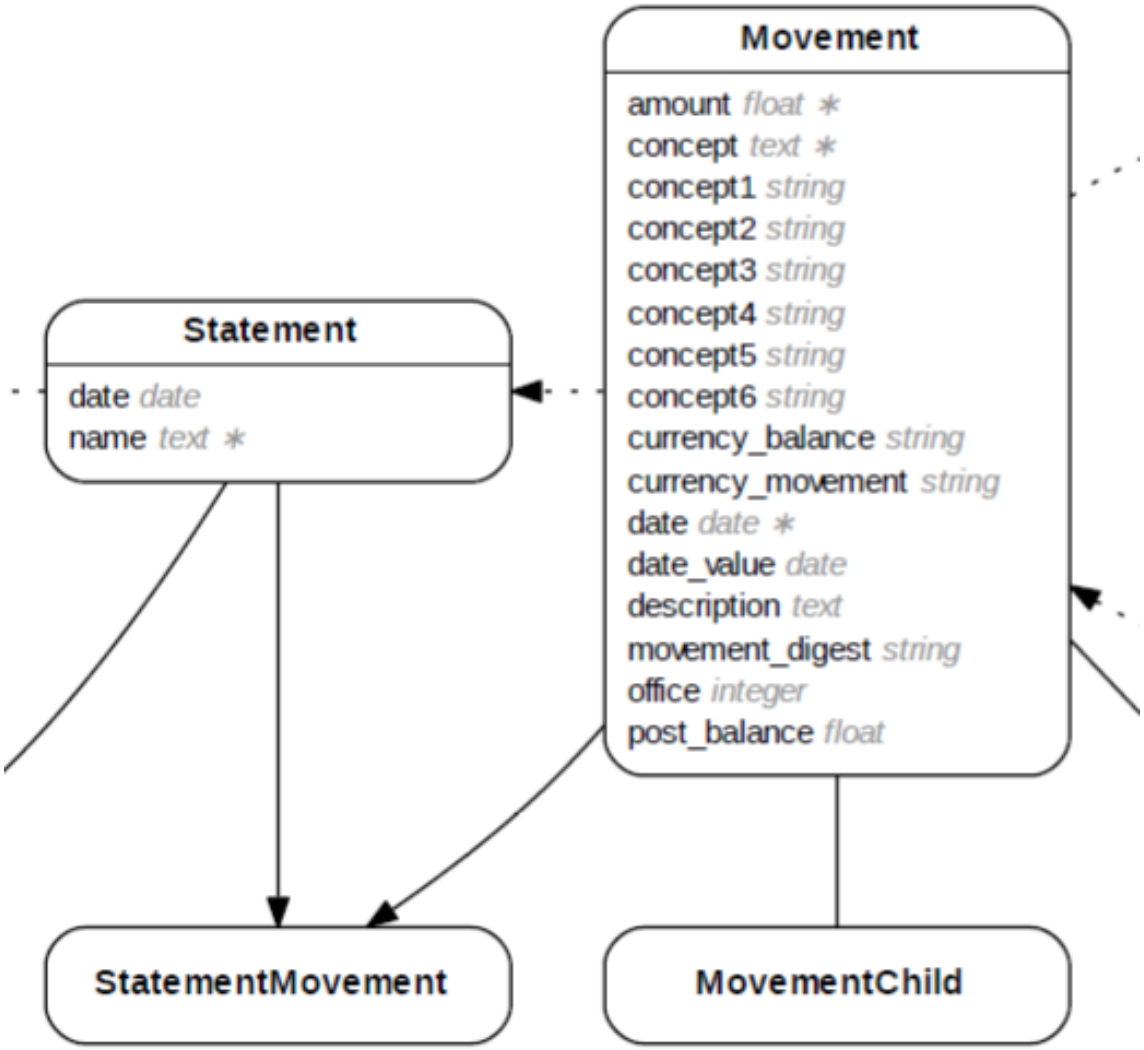

Figura 38: Relación de extractos con movimientos

La Figura 38 contiene dos relaciones de la base de datos, los extractos bancarios con los movimientos, utilizando la tabla intermedia **StatementMovement**, y la asociación reflexiva de movimiento, utilizando la tabla intermedia **MovementChild**.

La relación entre extractos y bancarios es de mucho a muchos, pero como se veía en el diagrama de clases, los movimientos bancarios al crearse necesitan como mínimo estar relacionado con un extracto bancario. La razón por la que un movimiento puede pertenecer a varios extractos bancarios es para poder solucionar el problema de duplicidad en la aplicación, mediante el atributo **MovementDigest**. **MovementDigest** se genera creado una ristra usando codificación md5 sobre todos los demás atributos del movimiento, pudiendo buscar a la hora de importar un extracto nuevo si el movimiento que queremos introducir ya existe en la base de datos.

Por otro lado, el modelo movimiento posee una asociación reflexiva, que se crea cuando dividimos un movimiento para generar dos movimientos descendientes, dando nombre a la tabla intermedia de **MovementChild**. La representación de esta relación no es muy precisa debido a que el movimiento está asociado a sí mismo y la relación no es mucho a muchos, si no uno a muchos (los movimientos descendientes solo pueden tener un movimiento padre) y esta representación no está contemplada en **Rails-ERD**.

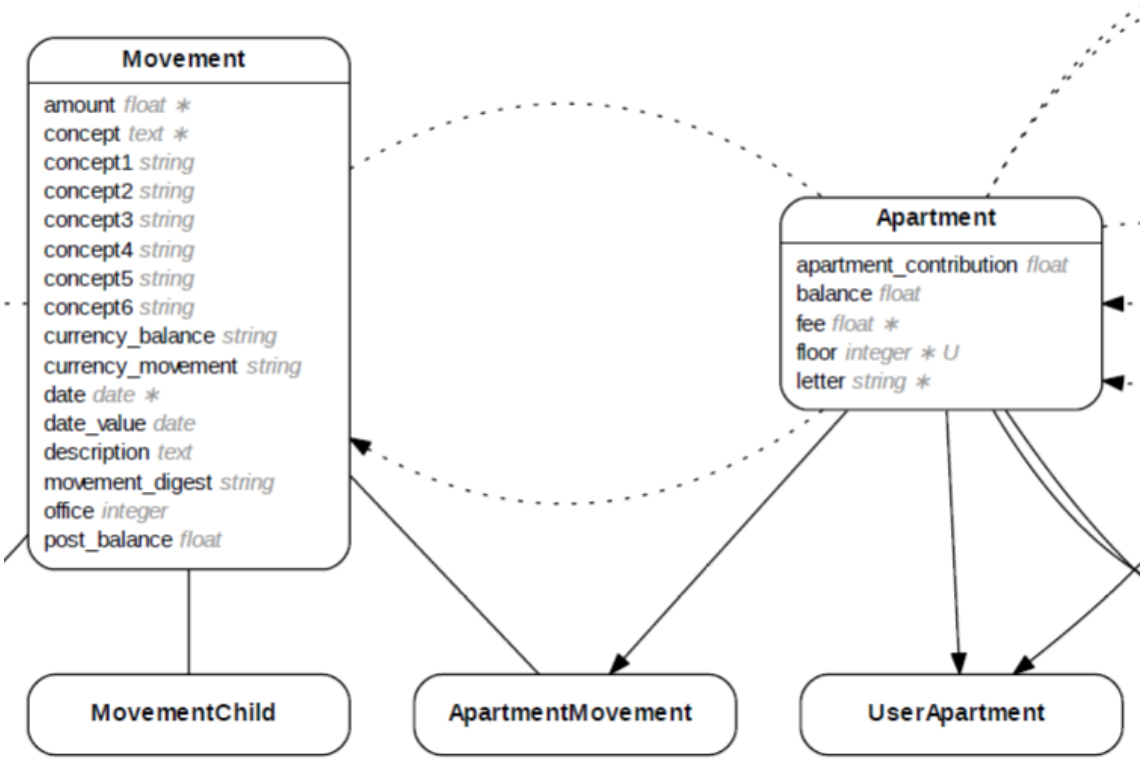

Figura 39: Relación de movimientos con viviendas

En la Figura 39 podemos ver la relación entre viviendas y movimientos, con la tabla intermedia **ApartmentMovement**. Las viviendas pueden tener muchos movimientos bancarios, mientras que un movimiento bancario solo puede pertenecer a una vivienda o a ninguna, de ahí que las flechas continuas hacia **ApartmentMovement** y las discontinuas entre sí no sean equivalente.

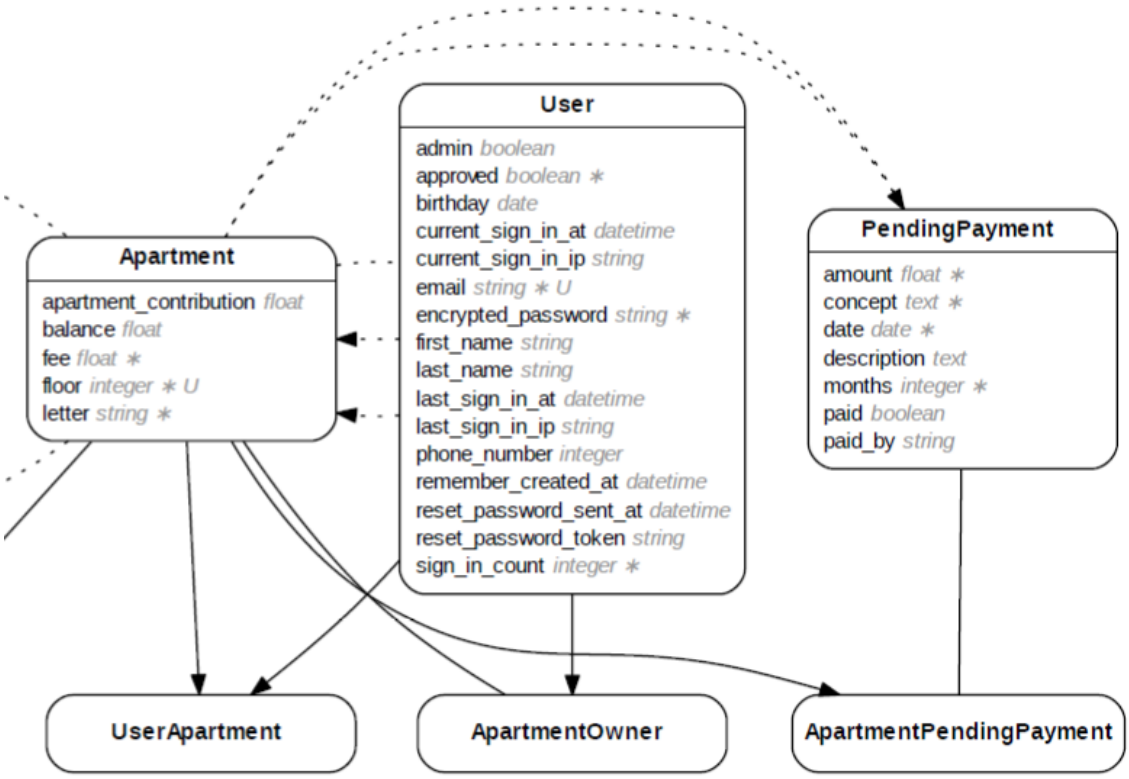

Figura 40: Relación de viviendas con usuarios y pagos pendientes

La última figura de la tabla tiene el resto de las relaciones, que son tres:

- **UserApartment:** La relación actual de los dos modelos es de muchos a muchos, pese a que en el prototipo actualmente estamos aplicando lógica de uno a muchos (un usuario solo puede residir en una vivienda en la implementación del proyecto actual). Esto se debe a que inicialmente se planteó la relación de propiedad y la de residencia como una sola en la relación entre usuarios y viviendas, ya que el propietario de una vivienda no tiene por qué residir en esa misma vivienda. Con este fin, se creó la nueva relación entre usuario y vivienda, mediante la tabla **ApartmentOwner**.
- **ApartmentOwner:** Segunda relación entre usuarios y viviendas de uno a muchos, donde un usuario puede tener muchas viviendas pero una vivienda solo puede tener un propietario.
- **ApartmentPendingPayment:** La última relación de la aplicación, que une en una relación de uno a muchos los pagos pendientes con las viviendas donde un pago pendiente solamente puede estar asociado a una vivienda.

Pese a que las relaciones del ERD se podrían simplificar, el diagrama actual es el que refleja de manera adecuada el estado de la aplicación, con las relaciones siendo fruto del aprendizaje de la tecnología usada y aprendiendo a posteriori que no ha sido la mejor manera de relacionar los modelos.

3.4. Tecnologías

Con este fin, las tecnologías que se utilizarán para el desarrollo del proyecto son:

Ruby, el lenguaje principal que utilizaremos para el desarrollo de la aplicación. En conjunto a **Ruby on Rails**, el objetivo principal de utilizar estas herramientas es para aprender un nuevo lenguaje que no hemos aprendido durante el grado. Se trata de un lenguaje de programación dinámico y código abierto, con una sintaxis fácil de entender y de escribir.

Ruby on Rails, framework basado en **Ruby** para el desarrollo de aplicaciones web. Sigue principalmente el paradigma del patrón Modelo Vista Controlador. Al igual que **Ruby**, **Ruby on Rails** es código abierto, y principalmente permite al desarrollador empezar una aplicación funcional escribiendo poco código y con un mínimo de configuración. La versión que hemos utilizado es la 5.2, que incluye la funcionalidad de ActiveStorage, permitiendo una conexión y subida casi directa con plataformas en la nube, como S3 de AWS, o Google Cloud.

Github, un servicio de control de versiones que utiliza git, dónde alojaremos el proyecto de manera pública. **Github** nos permitirá tener siempre una copia online del proyecto en caso de que ocurra algo de manera local, o trabajar en algún lugar que no sea el entorno habitual sin problemas.

Heroku, una plataforma que nos permite desplegar aplicaciones en la nube. Con **Heroku** podremos desplegar nuestra aplicación de manera gratuita públicamente, simulando a su vez cómo sería un despliegue en entorno de producción.

Para el desarrollo de la aplicación, que se realizará en un entorno local, utilizaremos una máquina virtual con el sistema operativo **Ubuntu** 16.04. Utilizaremos **Ubuntu** porque es más sencillo de configurar **Ruby** y **RoR** para trabajar en nuestra aplicación que en windows.

Utilizaremos una IDE para desarrollar la aplicación, haciendo uso de la versión ultimate de **Rubymine**. Incluye reconocimiento de sintaxis para **Ruby** y **RoR**, además de marcar los errores en el código, permitiendo un trabajo más fluido y eficiente.

NinjaMock es la herramienta de mockup que hemos utilizado para crear nuestros mockups. Hemos utilizado este en particular porque la versión gratuita nos permitía tener un número ilimitado de páginas, característica que muchas versiones de prueba no permitía.

Para la realización del proyecto en sí, hemos utilizado **Google Drive** para almacenar todos los ficheros y recursos utilizados, que es de carácter gratuito, permitiendo una organización simple de

todos los archivos del trabajo de fin de título.

Para la redacción de la memoria, hemos utilizado **Overleaf**, una plataforma online que utiliza Latex para escribir documentos orientados al entorno científico. Esto nos permite generar de manera más sencilla y similar a código las secciones de la memoria y hacer referencias de manera sencilla a otras partes del documento.

El programa que hemos utilizado para la creación de casos de uso y los diagramas de clases del proyecto ha sido con **StarUML**, que nos ha permitido exportar los diagramas en formato png directamente.

Para el almacenamiento de las ficheros que subimos utilizando la aplicación utilizamos el S3 de **Amazon Web Services**, un hosting gratuito inicialmente con el que hemos vinculado directamente el proyecto al utilizar una nueva funcionalidad que añadieron en la última versión de rails.

Asimismo, tenemos un listado de las gemas que hemos incluido y utilizado de manera activa en el desarrollo del proyecto:

Para el CSS de la aplicación hemos utilizado fundamentalmente la gema de **Bootstrap**, fundamentalmente para que la aplicación sea "responsive", permitiendo a los usuarios de móvil a tener una experiencia agradable también y estilizar la página web de manera rápida y eficaz.

Para la creación por defecto hemos utilizado la gema **Devise**, que nos permite una configuración rápida para empezar con un sistema de inicio de sesión y creación de usuario casi inmediato. No obstante, la gestión de usuarios no hace uso de **Devise** y se ha hecho de tal manera que pueda coexistir con la gema.

El código html de la aplicación se ha escrito utilizando la gema **Haml**, que simplifica el código html basado en identación con el objetivo de hacer el código menos repetitivo y más legible.

La gema **Pry** es una alternativa a la consola que utiliza Ruby por defecto, IRB. Su uso primario y fundamental en la aplicación ha sido su capacidad de depurar código, al poder incrustar la gema en cualquier parte del código y utilizar un entorno de consola con las variables que estén declaradas hasta llegar al punto señalado.

Aws-sdk es la gema que nos proporciona Amazon Web Services para poder conectarnos a su plataforma en la nube y poder subir los archivos haciendo uso de rails 5.2 con Active Storage.

La gema utilizada para traducir muchos de los textos que hay por defecto en rails y en devise ha sido **rails-i18n**, donde también podemos añadir nuestras propias frases, que no necesariamente tienen que ser traducciones, para que el sistema interprete de

manera automática.

La paginación de las tablas se ha conseguido gracias a la gema **will_paginate**, una solución sencilla para implementar paginación al momento.

Pese a que es una de las gemas que están por defecto en rails, cabe mencionar **Minitest**, la gema que hemos utilizado para la compilación y comprobación de los test escritos en el proyecto.

Para realizar la subida a producción en **Heroku**, hemos añadido la gema **Pg**, que es postgresql, el sistema de gestión de base de datos que utiliza **Heroku**.

3.5. Desarrollo de las iteraciones

Como se había comentado en la metodología, hemos tomado elementos de algunos de las metodologías de desarrollo ágil para realizar el proyecto, entre ella cabe destacar el uso de iteraciones y de una pila global de producto y uno de iteración, para planificar el trabajo a realizar durante la semana. El desarrollo del proyecto se ha realizado en tres iteraciones, de una duración de una semana por cada iteración, donde se ha ido actualizando al final de cada semana el product backlog con posibles errores que surgiera al final de la iteración. Cada caso de uso o historia de usuario del product backlog tiene asociado un valor llamado "Story points", o puntos de historia, que estiman entre intervalos de tiempo en horas el esfuerzo del caso de uso. El convenio que se ha seguido en el documento ha sido este:

- 1 punto de historia: Menos de una hora.
- 2 puntos de historia: Entre una hora y dos horas.
- 3 puntos de historia: Entre dos y cuatro horas.
- 5 puntos de historia: Entre cuatro y ocho horas.
- 8 puntos de historia: Entre uno y tres días.
- 13 puntos de historia: Más de tres días, recomendable dividir el caso de uso de subtareas.

Se considera como un día de trabajo en el proyecto 8 horas de trabajo, que sería el estándar estipulado en un proyecto. Las horas dedicadas a cada iteración son altamente variantes, por lo cuál se podría decir que la implementación de las iteraciones no ha sido ejecutada de la manera más adecuada.

En las secciones que vienen a continuación iremos explicando el proceso de cada iteración y el tiempo real que ha tomado los casos de uso, explicando el estado inicial de la pila de productos y de la iteración, los casos de uso más relevantes, además de cualquier imprevisto o problema que hubiera surgido a lo largo del desarrollo. En las iteraciones también comentaremos el estado del intento de aplicación de TDD en las distintas etapas del proyecto.

La pila de productos, al igual que el progreso de cada iteración y las tareas realizadas están disponibles en este link. Además, se añadirán capturas al anexo en la Subsección 6.5 del documento del estado del backlog en cada iteración por si el link del documento no estuviera disponible.

3.5.1. Primera iteración

Puntos de historias de usuario realizados: 45

Tiempo real invertido aproximado: 40 horas

El proyecto comenzó directamente desde la iteración uno, considerando antes de empezar que se había hecho una pila de producto con historias de usuario en función de los requisitos que habíamos analizado previamente. El proyecto se empezó con un total de 42 casos de uso, viniendo todos directamente desde los requisitos. La mayoría de los casos de uso rodaban en una estimación de uno y dos puntos de historia, ya que eran casos de uso relacionados con la creación de modelo y las operaciones CRUD de estas. Entre los casos de uso más significativos que se añadieron a la pila inicial de productos, tenemos las que se pueden ver en la Figura 41:

ID	Date Added	Task	Time estimated
26	19/06/2018	Quiero poder importar un extracto a partir de un CSV	8
40	19/06/2018	Ver si se puede automatizar los pagos pendientes de cuota	8
41	19/06/2018	Quiero poder generar cuotas de una vivienda automaticamente cada mes	8
42	19/06/2018	Quiero disponer de filtros para ordenar las tablas en los extractos	8
5	19/06/2018	Método para dividir / juntar las cantidades, creando nuevos movimientos.	5
27	19/06/2018	Quiero poder dividir cantidades en la importación del extracto	5
33	19/06/2018	Quiero poder elegir en el pago pendiente a qué viviendas me quiero asociar	5
37	19/06/2018	Quiero poder pagar un pago pendiente con el saldo de la vivienda	5

Figura 41: Casos de uso con más esfuerzo de la pila de productos inicial

Los casos de uso que se añadieron a la pila de la iteración fueron:

- Creación de los modelos de Usuario, Vivienda, Extracto, Movimiento y Pago pendiente.
- Operaciones CRUD en todos los modelos anteriores.
- Añadir campos adicionales al modelo de Usuario.
- Relación entre Extracto y Movimiento.
- Importar un extracto a partir de un CSV.

- Dividir cantidades de la importación del extracto.

Antes de empezar a realizar los casos de usos que había, se estableció una prioridad basado en los elementos que nos permitieran crear algo tangible lo antes posible. Entre ellos, está principalmente el modelo de usuario, que creamos utilizando la gema **Devise**, para poder tener un usuario que esté realizando las acciones de la aplicación. **Devise** además nos añade métodos auxiliares como el usuario actual de la sesión, el cual hemos utilizado para bloquear todas las acciones disponibles en los controladores de los visitantes.

ID	Task	Story points	Priority
1	Modelo Usuario	2	High
2	Modelo Vivienda	2	High
3	Añadir campo de administrador a usuario	1	High
4	Modelo Extracto	2	High
6	Modelo Movimiento	2	High
7	Modelo pago pendiente	2	High

Figura 42: Casos de uso con mayor prioridad de la pila de iteración

El caso de uso con el que empezaría el administrador al utilizar la aplicación sería el de importar un extracto a partir de un CSV, pero los primeros casos de uso que se escogieron para el desarrollo de la primera iteración fueron fundamentalmente la de creación de modelos y las operaciones CRUD con estos basado en la prioridad establecida, para ir probando la tecnología entre manos. La primera iteración empezó con buenos prospectos de la aplicación, implementando tras la creación de los modelos los test que comprobarían la validez de una instancia del modelo creado. Al crear el modelo de usuario usando devise, la primera modificación que hicimos fue añadir los atributos que se ven en la Figura 32, ya que el rol de administrador no está por defecto en los usuarios creados. Tras terminar de crear los modelos y sus funcionalidades, además de los test para estos, se empezó a implementar funcionalidades que no tuvieran dependecia aún de las relaciones, entre las que estaba dividir los movimientos.

```ruby
def divide_movement
  if current_user.admin
    @new_movement = Movement.new(movement_params)
    amount = @new_movement.amount
    if @new_movement.amount >= @movement.amount
      flash[:danger] = 'La cantidad que quieres separar es mayor o igual al original, prueba con otra cantidad.'
      redirect_to @statement
    else
      @movement.amount -= amount
      @movement.save
      if @new_movement.save
        @statementmovement = StatementMovement.new(statement_id: @statement.id, movement_id: @new_movement.id)
        if @statementmovement.save
          flash[:info] = "Se ha dividido el movimiento #{@movement.concept}"
          redirect_to @statement
        else
          flash[:danger] = 'Ha ocurrido un error a la hora de dividir el movimiento para el extracto bancarios.'
          redirect_to root_url
        end
```

Figura 43: Código de la funcionalidad dividir movimiento

El código de Figura 43 comprueba inicialmente si el usuario actual es un administrador, que en caso de no cumplirse genera un mensaje de error "Tu cuenta no tiene permisos para realizar esta acción. Por favor, contacta con el administrador para más información". Al cumplirse, se comprueba si la cantidad que ha puesto el usuario al que dividir el movimiento es mayor que el que posee el movimiento, generando el mensaje de error que adecuado en caso de superarlo. En caso de que no hubiera más problemas, se crearía un nuevo movimiento y se actualiza el movimiento anterior restándole la cantidad puesta por el usuario. Las cantidades negativas por el usuario se controla desde el modelo del movimiento, donde el movimiento no se considera válido si no tiene un número mayor o igual que 0. El problema inicial que tuvo esta funcionalidad era que se podía crear un nuevo movimiento con 0 de cantidad, que no tiene mucho sentido.

```ruby
class Movement < ApplicationRecord
  has_many :statement_movements
  has_many :statements, through: :statement_movements

  validates :concept, presence: true
  validates :date, presence: true
  validates :amount, presence: true, numericality: { only_integer: true, greater_than_or_equal_to: 0}

end
```

Figura 44: Código del modelo de movimientos en la primera iteración

Tras terminar la primera iteración, la pila de productos de la iteración se puede ver en las figuras siguientes (Figura 45, Figura 46). Se añadieron nuevos casos de uso a la aplicación, en la retrospectiva de la iteración, principalmente las restricciones de rol en las páginas después de haber bloqueado las acciones de los visitantes además de algunos casos de uso puntuales que se descubrieron después de crear los modelos.

ID	Date Added	Task	Story points	Priority	Done?	Test?	Comments	Tiempo real (Story points)	Date Finished
1	19/06/2018	Modelo Usuario	2	High	☑	☐	Los test de devise vienen por defecto	2	20/06/2018
2	19/06/2018	Modelo Vivienda	2	High	☑	☑	Falta asociación muchos a muchos con usuarios, asocicación uno a muchos con movimientos, asociación uno a muchos con pagos pendientes	3	21/06/2018
3	19/06/2018	Añadir campo de administrador a usuario	1	High	☑	☐		2	21/06/2018
4	19/06/2018	Modelo Extracto	2	High	☑	☑		2	22/06/2018
6	19/06/2018	Modelo Movimiento	2	High	☑	☑	Asociación con extractos	2	23/06/2018
7	19/06/2018	Modelo pago pendiente	2	High	☑	☑	Falta asociación con vivienda	2	22/06/2018
9	19/06/2018	Tabla intermedia extracto-movimiento	3		☑	☐	Tests?	3	24/06/2018
12	19/06/2018	Quiero poder crear un nuevo usuario	1	Medium	☑	☐		1	20/06/2018
13	19/06/2018	Quiero poder borrar un nuevo usuario	1	Medium	☑	☐		1	20/06/2018
14	19/06/2018	Quiero poder actualizar un usuario	1	Medium	☑	☐		1	20/06/2018
17	19/06/2018	Quiero poder crear una vivienda	1	Medium	☑	☑	Falta seeds con las viviendas por defecto	2	21/06/2018
18	19/06/2018	Quiero poder actualizar una vivienda	1	Medium	☑	☑		2	21/06/2018
19	19/06/2018	Quiero poder ver el perfil de una vivienda	1	Medium	☑	☑		2	21/06/2018
20	19/06/2018	Quiero poder borrar una vivienda	1	Medium	☑	☑	En un futuro esto debería quitarse, o esconderlo	2	21/06/2018
22	19/06/2018	Quiero poder crear un extracto	1	Medium	☑	☑		1	22/06/2018
23	19/06/2018	Quiero poder actualizar un extracto	1	Medium	☑	☑		1	22/06/2018
24	19/06/2018	Quiero poder borrar un extracto	1	Medium	☑	☑		1	22/06/2018
25	19/06/2018	Quiero poder ver un extracto	1	Medium	☑	☑		1	22/06/2018
26	19/06/2018	Quiero poder importar un extracto a partir de un CSV	8		☐	☐	Falta la funcionalidad, el template está preparado	1	

Figura 45: Pila de producto de la primera iteración

27	19/06/2018	Quiero poder dividir cantidades en la importación del extracto	8		☐	☑	Falta tests, posible necesidad de modificar según los casos de uso futuro		
28	19/06/2018	Quiero poder crear un movimiento	1	Medium	☑	☑	Plantearse si deberían estar fijos tras importación	1	23/06/2018
29	19/06/2018	Quiero poder actualizar un movimiento	1	Medium	☑	☑	Asignación de viviendas actualizando el movimiento, después de crear asociación con viviendas	1	23/06/2018
30	19/06/2018	Quiero poder borrar un movimiento	1	Medium	☑	☑		1	23/06/2018
31	19/06/2018	Quiero poder ver un movimiento	1	Medium	☑	☑		1	23/06/2018
32	19/06/2018	Quiero poder crear un pago pendiente	1	Medium	☑	☑	Falta elegir viviendas a la que crear el pago pendiente	1	22/06/2018
34	19/06/2018	Quiero poder actualizar un pago pendiente	1	Medium	☑	☑	Actualizar usuarios cuando haya asociación	1	22/06/2018
35	19/06/2018	Quiero poder ver un pago pendiente	1	Medium	☑	☑	Verlo como tabla, verlo para todos los usuarios cuando haya asociación, ver el de un usuario	1	22/06/2018
36	19/06/2018	Quiero poder borrar un pago pendiente	1	Medium	☑	☑		1	22/06/2018
52	24/06/2018	Crear un movimiento desde el extracto bancario directamente	2		☑	☐	Habría que quitar la creación de movimiento desde la barra de nav	2	24/06/2018
27.1	24/06/2018	Dividir las cantidades de un extracto	5		☑	☐	La funcionalidad sin estar dentro de la importación	3	24/06/2018
					☐	☐			
		Total:	56		☐	☐	Total:	45	

Figura 46: Continuación de la pila de producto de la primera iteración

Al final de la Figura 46 podemos ver el tiempo que habíamos estimado para todos los casos de uso y el tiempo real que tardamos. Considerando que no conseguimos terminar los dos casos de uso con valor ocho, no incluimos el tiempo en el tiempo que tardamos, por lo que el tiempo estimado de los otros casos de uso realizados se asemejaron adecuadamente al tiempo que tardamos realmente.

3.5.2. Segunda iteración

Puntos de historias de usuario realizados: 30

Tiempo real invertido aproximado: 35 horas

Al comenzar la segunda iteración, se añadieron unos casos de usos adicionales antes de comenzar la iteración, aparte de los que se había sumado al finalizar la primera, sumando a un total de 58 casos de uso. Estos casos de uso surgen a raíz de que se había empezado ya a crear las relaciones entre los modelos, y al interactuar entre ellos surgieron nuevos problemas que no se habían contemplado inicialmente. A diferencia de en la primera iteración, las metas principales de la segunda iteración no fueron cubiertas, debido a que empezaron a surgir problemas tras terminar todas las relaciones entre los modelos. Sumado a esto, el desarrollo de test que hicimos en la primera iteración empezó a complicarse durante el desarrollo de esta iteración, por lo que se acabó aplazando hasta los últimos días de la iteración, resultando en no tener las funcionalidades más grandes terminadas para cuando se acabó la iteración, que era la importación de extractos y la ordenación de las tablas creadas aunque no fueran las más prioritarias. Sumado a esto, muchos de las características de los modelos tuvieron que cambiar debido a restricciones que iban en contra de

las funcionalidades que se querían implementar, haciendo que los test previos que había no funcionaran de manera adecuada.

43	22/06/2018	Añadir a las páginas restricciones de rol	5
44	22/06/2018	Setup para mostrar los tests de 1 en 1	3
45	22/06/2018	Las viviendas no pueden estar repetidas	3
46	22/06/2018	Cambiar el piso 0 a bajo en las viviendas	2
47	22/06/2018	Creación de cuenta sea una solicitud con confirmación de admin	5
48	22/06/2018	Autocapitalize las letras de las viviendas	1
49	22/06/2018	Si no hay viviendas, poner un mensaje descriptivo de ello	1
50	22/06/2018	De la importación de extractos bancarios filtrar los números positivos	3
51	22/06/2018	Cambiar las vistas a tablas	3
52	24/06/2018	Crear un movimiento desde el extracto bancario directamente	2
53	24/06/2018	Arreglar CSS de los formularios	2
54	25/06/2018	Después de tener la importación, quitar la habilidad de crear nuevos extractos desde interfaz	3
55	25/06/2018	Añadir un menú de opciones en las páginas que los requira como un glyphicon	3
56	25/06/2018	Vista personalizada de usuario donde estén sus viviendas	2
57	25/06/2018	Lógica para elegir un dueño de la vivienda en función de sus usuarios	3
58	25/06/2018	Quiero poder quitar usuarios de una vivienda	2

Figura 47: Casos de uso añadidos a la pila de productos

Los casos de uso que se añadieron a pila de la segunda iteración fueron:

- Relaciones entre los modelos existentes:
  - Vivienda - Usuario
  - Vivienda - Pago Pendiente
  - Vivienda - Movimiento
- Añadir y quitar usuarios de la vivienda.
- Añadir y quitar movimientos de la vivienda.
- Asociar, pagar y visualizar pagos pendientes de una vivienda.
- Los casos de uso de la iteración anterior que no se cumplieron.

Las prioridades de este ciclo se centrarón en la creación y correcto funcionamiento de las relaciones entre los modelos, dado a que se consideró más importante que las demás funcionalidades funcionaran con movimientos creados a mano. La importación al fin y al cabo lo único que nos proporciona son movimientos de un extracto, ya que no contribuye al funcionamiento de las demás características.

8	19/06/2018	Tabla intermedia usuario-vivienda	3	High
10	19/06/2018	Tabla intermedia vivienda-pago pendiente	3	High
11	19/06/2018	Tabla intermedia vivienda-movimiento	3	High
56	25/06/2018	Vista personalizada de usuario donde estén sus viviendas	1	High

Figura 48: Prioridad de los casos de uso de la segunda iteración

Entre las funcionalidades que se implementaron, uno de los casos de uso con más esfuerzo fue el de crear pagos pendientes para las viviendas. Además de la creación de un pago pendiente y la posibilidad de asociarlo a una vivienda, se modificó la funcionalidad para poder elegir en el formulario todas las viviendas a la que queríamos crear y asociar directamente el pago pendiente creado.

```
14    def create
15      if current_user.admin?
16        apartments = params[:pending_payment][:apartment]
17        if apartments.nil?
18          create_pending_payment
19        else
20          apartments.each do |id|
21            @pending_payment = PendingPayment.create!(pending_payment_params)
22            @apartment = Apartment.find(id)
23            @apartmentpendingpayment = ApartmentPendingPayment.new(apartment: @apartment, pending_payment: @pending_payment)
24            if @apartmentpendingpayment.save
25              flash[:info] = '¡Nuevo pago pendiente creado!'
26            else
27              flash[:danger] = 'Ha ocurrido un error a la hora de crear un pago pendiente para las viviendas elegidas'
28              redirect_to pending_payments_path
29              return
```

Figura 49: Código de la creación de pagos pendientes

Como en todas las acciones del controlador en las que el usuario no tuviera acceso, controlamos si el usuario es administrador o no. Acto seguido, obtenemos los parámetros de las viviendas elegidas en el formulario. Si no se ha elegido ninguna vivienda, llamamos al método auxiliar para crear un pago pendiente sin asociar con los datos introducidos. En caso de que se haya elegido viviendas en el formulario, creamos un pago pendiente para cada vivienda, creando una nueva asociación entre estos. El método auxiliar crea el pago pendiente si los datos recibidos del formulario son correctos.

```
115   def create_pending_payment
116     @pending_payment = PendingPayment.new(pending_payment_params)
117     if @pending_payment.save
118       flash[:info] = "¡Nuevo pago pendiente creado!"
119       redirect_to @pending_payment
120     else
121       flash[:danger] = 'Ha ocurrido un error en el sistema, por favor, vuelva a intentarlo.'
122       render 'new'
123     end
124   end
```

Figura 50: Código del método auxiliar de la Figura 49

Parecido a la asociación que se crea del pago pendiente con la vivienda en la creación de pago pendiente, implementamos la funcionalidad de asociar el pago pendiente a una vivienda directamente. La diferencia entre las dos asociaciones que se generan, es que esta característica permite asociar un pago pendiente aunque ya tenga una asociación previa. Para ello, primero comprueba si el pago pendiente tiene una asociación previa. En caso de que lo tenga, actualiza la asociación con la nueva vivienda, y genera un mensaje de éxito adecuado. En caso de que no encuentre ninguna asociación previa del pago pendiente con una vivienda, el mensaje de éxito indica simplemente que el pago pendiente se ha asociado a la vivienda.

```
78    def associate_apartment
79      if current_user.admin
80        @apartment = Apartment.find(params[:apartment_id])
81        @apartmentpendingpayment = ApartmentPendingPayment.find_by(pending_payment: @pending_payment)
82        if @apartmentpendingpayment.nil?
83          @apartmentpendingpayment = ApartmentPendingPayment.new(apartment: @apartment, pending_payment: @pending_payment)
84          if @apartmentpendingpayment.save
85            flash[:info] = "Se ha asociado el pago pendiente a la vivienda #{full_name_apartment(@apartment)}"
86            redirect_to pending_payments_path
87          else
88            flash[:danger] = 'Ha ocurrido un error al intentar crear la asociación del pago pendiente'
89            redirect_to pending_payments_path
90          end
91        else
92          if @apartmentpendingpayment.update_attribute(:apartment, @apartment)
93            flash[:info] = "Se ha cambiado la asociación del pago pendiente a la vivienda #{full_name_apartment(@apartment)}"
94            redirect_to pending_payments_path
```

Figura 51: Código de la asociación de un pago pendiente a una vivienda

Al final de la segunda iteración, la pila de productos de la iteración es la que se ve en la Figura 52. Al no poder completar los casos de uso con más esfuerzo de la pila de producto, la cantidad de puntos de historia completada en esta iteración fue mediocre en comparación a la primera iteración. Además, tras realizar una comprobación del estado del producto al acabar la segunda iteración, muchos de los elementos visuales eran muy primitivos, además de que las vistas de las páginas contenían mucha información redudante. Todos los cambios visuales que necesitaba la aplicación hizo que la iteración tres tomara la carga equivalente de dos iteraciones en una, dado a que las funcionalidades principales de la aplicación (importar extracto y filtrar la información) aún no estaban implementadas y al ser una aplicación web se necesitaba un mínimo de trabajo visual.

ID	Date Added	Task	Story points	Priority	Done?	Test?	Comments	Tiempo real	Date Finished
8	19/06/2018	Tabla intermedia usuario-vivienda	3	High	☑	☐	No hubo tanto problemas como el anterior. Falta tests	1	25/06/2018
10	19/06/2018	Tabla intermedia vivienda-pago pendiente	3	High	☑	☐		2	30/06/2018
11	19/06/2018	Tabla intermedia vivienda-movimiento	3	High	☑	☐	Se genera saldo a partir de la lista de movimientos que le "pertenezcan"	2	29/06/2018
16	19/06/2018	Quiero poder añadir un usuario a una vivienda	3	Medium	☑	☐	Tests, esto incluye también quitar usuarios	3	25/06/2018
21	19/06/2018	Quiero poder quitar movimientos de una vivienda	3	Medium	☐	☐	Quizás no sea necesario, considerando que los movimientos siempre deberían estar asociados a alguien		
26	19/06/2018	Quiero poder importar un extracto a partir de un CSV	8	Low	☐	☐	Falta la funcionalidad, el template está preparado		
33	19/06/2018	Quiero poder elegir en el pago pendiente a qué viviendas me quiero asociar	5	Low	☑	☐	A la hora de crear un pago pendiente, elegir viviendas y crear pagos pendientes correspondientes, además de poder asignar el pago pendiente a una vivienda	5	01/07/2018
37	19/06/2018	Quiero poder pagar un pago pendiente con el saldo de la vivienda	5	Medium	☑	☐	Debería comprobar si el saldo que tiene es suficiente o no	3	30/06/2018
38	19/06/2018	Quiero poder ver un historial de los pagos pendientes, estén pagados o no	3	Low	☑	☐		1	30/06/2018
39	19/06/2018	Quiero poder asociar un movimiento a una vivienda	3	Medium	☑	☐	Al asociar a una vivienda, si ya estaba asociado, habría que controlar.	2	29/06/2018
42	19/06/2018	Quiero disponer de filtros para ordenar las tablas en los extractos	8	Low	☐	☐	Esto se debería hacer después de haber juntado ya los extractos con los movimientos, de manera que se pueda ordenar un extracto según varios criterios de los movimientos	3	
56	25/06/2018	Vista personalizada de usuario donde estén sus viviendas	1	High	☑	☐		1	25/06/2018
58	25/06/2018	Quiero poder quitar usuarios de una vivienda	2	Medium	☑	☐		2	25/06/2018
	01/07/2018	Arreglo de tests anteriores, y templates de los nuevos	8		☑	☐		5	01/07/2018
					☐	☐			
					☐	☐			
		Total:	58		☐	☐	Total	30	

Figura 52: Pila de producto de la segunda iteración

3.5.3. Tercera iteración

Puntos de historias de usuario realizados: 110

Tiempo real invertido aproximado: 85 horas

El comienzo de la tercera iteración fue con 93 casos de uso en la pila de productos, siendo la gran mayoría cambios visuales de la aplicación o comportamientos necesarios de algunas partes de las funcionalidades. Como se había mencionado al final de la segunda iteración, esto fue principalmente debido a la necesidad de terminar el prototipo incluyendo dos de las funcionalidades más importantes y un mínimo de aspecto visual atractivo en la aplicación. Entre la implementación de las características principales, la parte html de la aplicación se reescribió para utilizar el sistema de columnas de bootstrap, haciendo la aplicación como se ve ahora, siguiendo una política de "Mobile-first", implicando que la aplicación se tiene que ver bien desde el móvil primero antes de estilizar el sitio web en un escritorio. Los casos de uso añadidos a la pila de productos de la aplicación son las siguientes:

83	03/07/2018	Cambiar el crear pago actual para que se utilice para derramas	5
84	03/07/2018	Pagos pendientes deberían estar ordenados por defecto de menor a mayor	2
85	03/07/2018	Buscar una manera de bloquear todos los pagos menos el más antiguo	3
86	03/07/2018	Cambiar la vista del usuario, como el botón de cancelar cuenta	1
87	03/07/2018	Añadir una vista de usuarios con datos de contacto	1
88	03/07/2018	Añadir al usuario un atributo teléfono	1
89	03/07/2018	Bloquear el edit del usuario para que solo el admin o el propio usuario pueda cambiar determinados datos	3
90	03/07/2018	Añadir al usuario una asociación con los pagos pendientes a la hora de pagar	3
91	03/07/2018	Cambiar quitar usuarios de vivienda si el usuario es el propietario	3
92	03/07/2018	Login directamente en la página principal si no estás logueado	1
93	03/07/2018	Quitar opción de asociar a vivienda si es negativo el movimiento en extractos	1

Figura 53: Alguno de los casos de uso añadidos a la pila de producto

Los casos de uso que se añadieron a la pila de la tercera iteración fueron:

- Importar extracto bancario y ordenación de la tabla por varios filtros.
- Creación de cuenta de usuario no registrado.
- Mejoras de información con respecto a las viviendas, como que el piso "0" se vea en la aplicación como "Bajo".
- Controlar las vistas de los usuarios normales y el administrador.
- Mejora visual de toda la aplicación, incluyendo el formato de las tablas, los formularios, los menús desplegables...

- Cambiado la lógica de muchas de las operaciones CRUD que inicialmente se podían realizar. Crear extracto no se puede aplicar por su cuenta por ejemplo, requiere que se importe un extracto para poder crear extracto.
- Cambiado la creación de pago pendiente a la creación directa de pagos pendientes de todas las viviendas en función de un mes elegido y la cuota que le corresponde.
- Añadir la habilidad de crear pagos pendientes para las derramas de la comunidad, pudiendo repartirse en varios meses. La cantidad a pagar depende de la cuota de contribución de la vivienda.

Todos los casos de usos que se han añadido se pueden encontrar en el link referenciado al final de la Subsección 3.5 o en el anexo.

Las principales prioridades de la tercera iteración se pueden ver en la Figura 54:

ID	Date Added	Task	Story points	Priority
26	19/06/2018	Quiero poder importar un extracto a partir de un CSV	8	High
42	19/06/2018	Quiero disponer de filtros para ordenar las tablas en los extractos	8	High
62	30/06/2018	Controlar la vista de usuarios normales y de admin	5	High
65	03/07/2018	Quitar desplegables de los menús que no lo requieran	2	High
66	03/07/2018	Quitar cantidades negativas de los formularios	1	High
70	03/07/2018	Poner un nuevo atributo de cuota de participación a la vivienda	1	High
71	03/07/2018	Añadir un logo	1	High
77	03/07/2018	Los movimientos de extractos y movimientos deberían mostrar la misma información relevante a primera vista	2	High
78	03/07/2018	Cambiar el modelo movimiento para todos los campos del CSV	2	High

Figura 54: Prioridad de los casos de uso de la tercera iteración

Entre los casos de uso implementados comentaremos el código de importar extracto y la de la ordenación de tablas:

```
1  class StatementsController < ApplicationController
2    helper_method :sort_column, :sort_direction, :sort_movement_column
3    before_action :logged_in_user
4    before_action :statement_getter, except: [:index, :new, :create, :bucket]
5    before_action :permissions, except: [:index, :show]
```

Figura 55: Código de los métodos llamados antes de ejecutar un método

A diferencia de los códigos que se han mostrado en las iteraciones anteriores, ya no realizamos una comprobaciones directamente en el método de si el usuario es administrador. Esto es debido a que hemos creado un método de permisos que realiza esa comprobación, y utilizando la característica de rails **Before_Action**, podemos realizar la llamada al método de permisos antes de ejecutar los métodos que hayamos definido, como se puede ver en la línea 5. La línea 2 que contiene la llamada a **Helper_Method**, se explicará su uso en la ordenación de tablas.

```
15    def create
16      @statement = Statement.new(statement_params)
17      if @statement.date.nil?
18        @statement.date = Date.today.to_s
19      end
20      if @statement.bank_statement.attached?
21        if @statement.save
22          return unless check_csv.nil?
23          flash[:info] = "¡Nuevo extracto #{@statement.name} creado!"
24          return if import_csv.nil?
```

Figura 56: Código de la creación de un extracto bancario

Para la creación de un nuevo extracto bancario, comprobamos primero la fecha que ha puesto el usuario. En caso de no poner ninguna fecha, asignamos la fecha de hoy al extracto que se importa. La siguiente comprobación es la de si hay un fichero adjuntado en los parámetros que se envían en el formulario, en caso de no ser así, se indicaría al usuario que no hay ningún fichero seleccionado y que no se puede importar el extracto.

Al crear el nuevo extracto, procederemos a comprobar si el fichero que se ha adjuntado es un CSV, que veremos en la Figura 57. Si el fichero es un CSV, comenzaremos a importar los datos del CSV para crear movimientos asociados al fichero llamando al método **import_csv**.

```
117   def check_csv
118     unless @statement.bank_statement.blob.content_type == 'text/csv'
119       @statement.destroy
120       flash[:danger] = 'El archivo a importar no es un fichero CSV. Vuelva a intentarlo.'
121       redirect_to new_statement_path
122       false
123     end
124   end
```

Figura 57: Código del método auxiliar “Check CSV”

El método **check_csv** comprueba si el tipo asociado del fichero adjuntado es de tipo CSV. En caso de no serlo, se borra el extracto se había guardado y se redirecciona al administrador a la pantalla de importación de extracto, con un mensaje explicando que el fichero subido no es un CSV.

```ruby
def import_csv
  CSV.parse(@statement.bank_statement.download.force_encoding('UTF-8'), headers: true) do |row|
    row = row.to_hash
    if check_csv_params(row).nil?
      movement_digest = Digest::MD5.hexdigest(row.to_s)
      movement = Movement.find_by(movement_digest: movement_digest)
      if movement.nil?
        movement = Movement.new(date: row['Fecha Movimiento'], date_value: row['Fecha Valor'], concept: row['Concepto'],
                                amount: row['Importe'].sub(',', '.').to_f, currency_movement: row['Divisa'],
                                post_balance: row['Saldo Posterior'].sub(',', '.').to_f, currency_balance: row['Divisa'],
                                office: row['Oficina'], concept1: row['Concepto1'], concept2: row['Concepto2'],
                                concept3: row['Concepto3'], concept4: row['Concepto4'], concept5: row['Concepto5'],
                                concept6: row['Concepto6'], movement_digest: movement_digest)
        movement.lock!
        movement.save!
        StatementMovement.create!(statement: @statement, movement: Movement.last)
      else
        StatementMovement.create!(statement: @statement, movement: movement)
```

Figura 58: Código del método auxiliar “Import CSV”

El método **import_csv** coge el fichero adjuntado, que se ha comprobado que es de tipo CSV, y lo fuerza a codificación UTF-8, cogiendo los nombres de la cabecera del CSV y cada fila para pasar a procesarlo. En primer lugar, transformamos la fila en un hash para poder coger los valores según determinadas claves, que en este caso serían los nombres de las cabeceras. Antes de comenzar a importar el fichero CSV, hacemos una llamada al método auxiliar **check_csv_params**, que se encarga de comprobar si el fichero CSV tiene las columnas mínimas para poder realizar una importación.

En caso de tener las columnas necesarias, creamos primero un valor único en función de todos los valores de la fila, que utilizaremos como ristra identificadora única. Con este atributo del movimiento, podremos garantizar si dos movimientos son únicos o no. En caso de no ser único, el movimiento no debería crear el nuevo movimiento, si no asociar el movimiento al nuevo extracto. Esto implica que si importamos el mismo fichero dos veces, veremos los dos extractos, pero los movimientos asociados a cada extracto es el mismo. La importación entonces se crea en función de las cabeceras del fichero de prueba sobre la que hemos basado nuestra aplicación, que contiene todas las columnas que están en el modelo del movimiento.

```ruby
    def check_csv_params(row)
      if row['Fecha Movimiento'].nil? && row['Concepto'].nil? && row['Importe'].nil?
        @statement.destroy
        flash[:danger] = 'El CSV importado no tiene las columnas mínimas de "Fecha Movimiento"
        # redirect_to new_statement_path
        false
      end
    end
end
```

Figura 59: Código del método auxiliar “Check CSV Params”

Aunque el movimiento tenga una gran cantidad de atributos, para poder utilizar la aplicación solo necesitamos tres columnas cómo mínimo para poder subir el fichero CSV, siendo estas “Fecha movimiento”, “Concepto” e “Importe”. De esta manera,

podremos subir extractos bancarios aunque no tenga todos los campos que hemos utilizado como fichero de prueba.

Por otro lado, tenemos el código de la ordenación de columnas.

```
def sortable(column, title = nil)
  title ||= column.titleize
  css_class = column == sort_column ? "current #{sort_direction}" : nil
  direction = column == sort_column && sort_direction == "asc" ? "desc" : "asc"
  link_to title, request.parameters.merge({:sort => column, :direction => direction, page: nil}), { :class => css_class }
end
```

Figura 60: Código de la ordenación de una tabla

El código del método **sortable** se encuentra en el fichero "application_helper", el cuál indica que es un método que pueden utilizar todos los ficheros de la aplicación. El método recibe dos parámetros, el primero sería el nombre de la columna que se quiere ordenar, y el segundo el título de la columna, que es opcional. En nuestro caso, como los nombres de los atributos están en inglés mientras que la aplicación está en español, pasamos como título el título de la columna que queremos ver.

El método procede entonces a coger una dirección y una clase de CSS en función de la dirección de la ordenación, pudiendo ser ascendente o descendiente. La dirección y el nombre de la columna hace uso de unos métodos auxiliares que varía en función del modelo de la tabla a la que se llama. Esto es debido a que las columnas de un usuario no son las mismas que las de un movimiento. Aquí es donde entra en juego la segunda línea de código de la Figura 55, donde cada controlador tiene sus propios métodos auxiliares en función de las columnas que quieren ordenar. Esto lo podremos ver en la Figura 63 más adelante.

```
35    def show
36      @movements = @statement.movements.order(sort_movement_column + " " + sort_direction).paginate(per_page: 7, page: params[:page])
37    end
```

Figura 61: Variable del extracto bancario que se pasa a la vista

La Figura 61 muestra la variable que se le pasará a la vista del extracto para poder mostrar los movimientos. La sentencia que se ejecuta depende de las mismas variables de método **sorter**, para poder pasar a la vista la variable ordenadas según las variables elegidas. En esta sentencia también podemos ver el funcionamiento de la gema **will_paginate** combinada añadida al final de la línea.

```
.table-responsive
  %table.table.table-hover.table-statement
    %tr
      %th= m_sortable 'concept', 'Concepto'
      %th= m_sortable 'date', 'Fecha'
      %th= m_sortable 'amount', 'Cantidad'
      %th= m_sortable 'concept1', 'Concepto 1'
      %th= m_sortable 'concept3', 'Concepto 3'
      %th= m_sortable 'concept5', 'Concepto 5'
      %th= m_sortable 'description', 'Descripción'
      %th Vivienda asociada
      %th Opciones
    -@movements.each do |movement|
```

Figura 62: Código haml de la vista de movimientos en el extracto

La vista correspondiente a la acción **Show** obtiene la variable que hemos pasado, y procede a crear una tabla en función del resultado de la petición que se ha hecho de la base de datos. Según la ordenación que hemos elegido, la base de datos devuelve los movimientos según la columna y dirección elegida. La llamada al método **sortable** varía para los movimientos porque el comportamiento por defecto de los extractos, que es el controlador al que hemos llamado, es utilizar las columnas de los extractos, que son solamente "nombre" y "fecha".

```
70    def sort_direction
71      %w[asc desc].include?(params[:direction]) ? params[:direction] : "asc"
72    end
73
74    def sort_movement_column
75      Movement.column_names.include?(params[:sort]) ? params[:sort] : "concept"
76    end
```

Figura 63: Código de los métodos auxiliares de **Sortable**

Los métodos auxiliares que se han utilizado para realizar esta petición están restringidas para tomar determinados valores, para evitar inyección SQL. En el caso de la dirección, solamente puede tomar las direcciones ascendientes o descendientes, por lo que cualquier otra entrada tomará por defecto un sentido ascendiente. En el caso de las columnas, solamente se podrán incluir los nombres de los atributos del

modelo seleccionado, y en cualquier otro caso, cogera por defecto el nombre de una columna que elijamos, que en este caso es “Concept”.

Tras finalizar la tercera iteración, terminamos de implementar todas las funcionalidades especificadas inicialmente para el desarrollo del prototipo. Un link del prototipo funcional se proporcionará en la Subsección 3.7, donde explicaremos también cómo hemos puesto la aplicación en producción.

El resultado de la pila de producto de la tercera iteración se puede ver en la Figura 83 y Figura 84.

3.6. Pruebas

Como se ha indicado en la Subsección 3.5, la aplicación comenzó intentado aplicar TDD con las funcionalidades que se iban desarrollando, pero durante el desarrollo de la segunda iteración en adelante, se abandonó por la creación de test en medida de lo posible cerca del final de la iteración por la cantidad de nuevos casos de uso urgentes que se introducían en la pila de productos. En su conjunto, el porcentaje de test automatizados que cubran las acciones de la aplicación puede llegar aproximadamente a un 60-70 %. Este número se calcula inicialmente basado en la cantidad de test que quedan por implementar en la aplicación frente a los que hay actualmente. Los test que cubre la aplicación ahora mismo son las siguientes:

- Todos los modelos creados tienen test para los atributos que requieran alguna validación especial, como por ejemplo los atributos de vivienda, que requiere que el piso sea un número, exista una letra, tenga una cuota de la vivienda y una cuota de contribución.
- Todas las rutas definidas en el archivo “config/routes.rb”, donde se comprueba que se pueden acceder a ellas. Muchos de estas rutas están bloqueadas para usuarios que no sean administradores, y se hacen test para comprobar que los usuarios normales son redirigidos a la página principal, mientras que un administrador puede acceder sin problemas.
- Todas las acciones del controlador de las viviendas están cubiertas por test. Es el único controlador del que se ha podido cubrir completamente, y se ha hecho principalmente porque es el único modelo que está relacionado con todas las demás a excepción de los extractos, haciéndolo un eslabón con mayores vulnerabilidades.
- Algunas de las acciones de los otros modelos cuyos test se hicieron durante la primera iteración, como la actualización y borrado de los modelos.

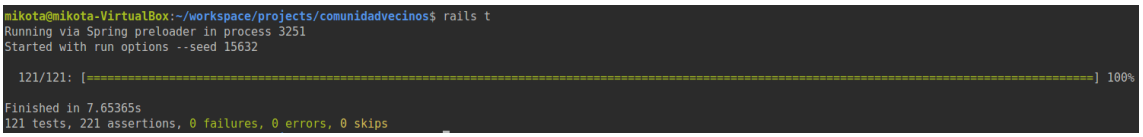

Figura 64: Captura de la ejecución de los test de rails

No obstante, para los test de acciones de controlador que faltan hemos dejado una plantilla con un título descriptivo sobre lo que debería hacer el test. Es por esto que podemos aproximar la cantidad de test que necesitamos para cubrir el código de controladores que hemos escrito restante. Terminar los test que faltan sería un trabajo futuro que se comentará en la Sección 4.

Por otro lado, la comprobación que se ha realizado para comprobar las acciones que faltan se han realizado a mano, probando posibles valores y comportamientos no habituales para ver si la aplicación devuelve error de alguna manera. La realización de las pruebas de esta manera durante la segunda mitad del desarrollo de la aplicación ha dejado claro que una implementación con TDD riguroso puede simplificar con creces la tarea de desarrollo futuro, al tener la seguridad de que el código antiguo está cubierto.

3.7. Despliegue

La aplicación actualmente se encuentra alojada en un servidor de **Heroku** de manera gratuita, que se puede acceder y probar en el siguiente link: comunidad-vecinos. Para poder probar la funcionalidad de la aplicación, se proporcionará adjuntado un fichero CSV que se puede importar, además de las credenciales para acceder a la aplicación como administrador en el anexo.

Para conseguir el despliegue en la aplicación, se ha utilizado una gema en particular, **pg**, o postgresql, que es la base de datos que utiliza Heroku para sus aplicaciones. Instalando en la máquina virtual la línea de comandos de Heroku, podemos iniciar sesión en nuestra cuenta y vincular la aplicación de rails con el repositorio de Heroku. De cara a Rails, había que configurar los ficheros de entorno de producción para que la aplicación apuntará a la URL que nos proporciona Heroku para el alojamiento.

Otro servicio en la nube que hemos utilizado para el proyecto es el S3 de Amazon Web Services. En él, podemos guardar todos los ficheros en la nube. Estos ficheros además se pueden descargar siendo administrador en la aplicación, permitiendo tener acceso en todo momento a los ficheros originales de los extractos por si ocurriera algún problema con los extractos actuales, o para contrastar información de algún tipo.

La configuración para utilizar el servicio de S3 directamente con **Active Storage** de rails es algo más complejo que el de Heroku. Para poder utilizar el servicio, necesitamos generar en nuestra cuenta un usuario con una clave secreta y una llave para poder utilizar los servicios del cubo. Estos credenciales se tienen que guardar de alguna manera para que rails pueda utilizarlo a la hora de utilizar el servicio. En versiones previas a 5.2 de Rails, esto se podía hacer de varias maneras, como por ejemplo crear variables de entorno en Heroku, o tenerlo escrito directamente en el código, que es un método muy poco seguro. En la versión 5.2 rails introdujo también el archivo "secrets.yml", donde se genera una clave maestra única para el proyecto de rails, del cuál estamos encargados de tener guardados. El archivo "secrets.yml" está encriptado utilizando la clave maestra, y contiene variables de entorno que definamos, pero en el proyecto en sí. Con esto garantizamos que mientras tengamos la clave maestra, nuestras variables siempre estarán con el proyecto y de una manera

segura.

Tras configurar las claves y llaves de acceso al S3 de Amazon, hay que configurar el S3 para que pueda aceptar peticiones con la llave que tenemos. Esto fue una de los problemas más grandes que hubo a la hora de importar extractos, dado a que inicialmente no habíamos configurado adecuadamente el cubo S3.

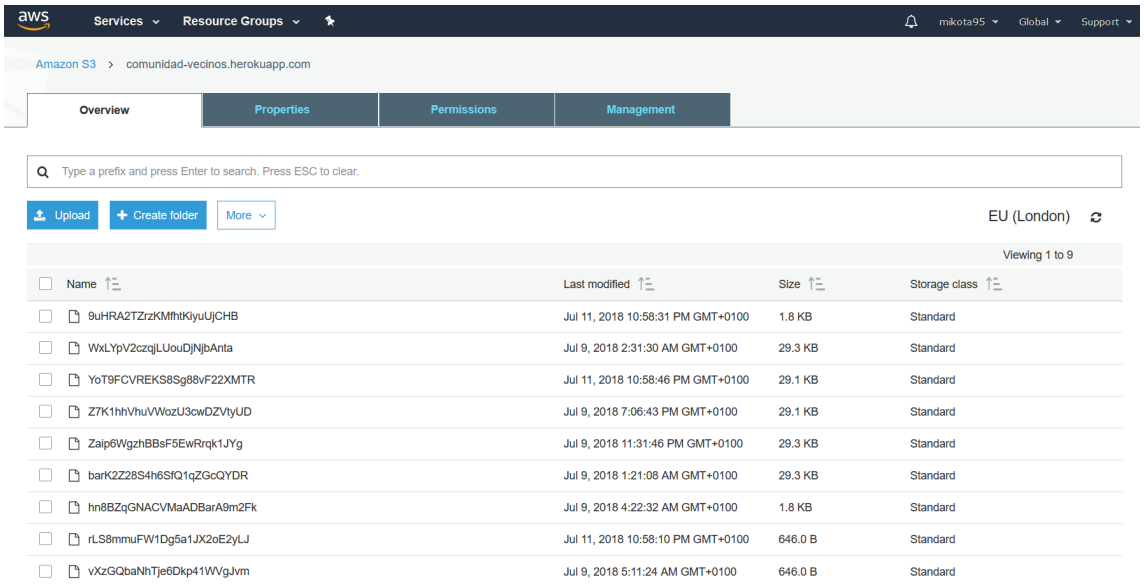

Figura 65: Cubo de S3 con los extractos encriptados

4. Conclusiones

Tras haber visto el análisis llevado a cabo para la realización del problema inicial, y el proceso de desarrollo llevado a cabo en iteraciones, el proyecto ha cumplido los dos objetivos iniciales marcados. El primer objetivo, como se mencionaba en la Subsección 1.1, era crear una aplicación que facilitara la gestión económica de una comunidad de vecinos. Con esta aplicación, el administrador o encargado de llevar las cuentas podrá importar los extractos bancarios de la cuenta corriente, crear los pagos pendientes que le corresponde a cada vivienda con un solo click y sobre todo, poder ver los movimientos de la cuenta y asignársela a las viviendas correspondiente, sin tener que llevar un seguimiento manual en papel, como se suele hacer en muchas comunidades de vecinos. Toda la información que esté relacionado con las viviendas se podrá visualizar, dejando claro si la vivienda tiene algún pago pendiente, o si se ha pagado y quién lo ha pagado. El prototipo desarrollado realiza las funciones principales que se requerían en el análisis realizado, por lo que el primer objetivo se ha alcanzado.

Por otro lado, se ha aprendido a utilizar **Ruby on Rails**, que es un framework actual y relavante, además de haber puesto en práctica el conocimiento que se ha aprendido en la formación académica en las partes de análisis y el desarrollo de la aplicación. Como se ha podido ver en las iteraciones del desarrollo, hemos vivido de manera empírica la implementación de varias metodologías, como el intento de

TDD, que ha dejado claro que habría mejorado el proceso de desarrollo si se hubiera aplicado bien y que se tendrá en cuenta en futuros desarrollos, aunque tenga un coste inicial más alto. Estas experiencias te hacen ver puntos de vista que no se entenderían desde una posición puramente teórica, que es el estado inicial que teníamos al empezar el proyecto. Sumado a esto, se ha aprendido a hacer uso de servicios de la nube, ya sea para el despliegue de las aplicaciones, el uso de un repositorio remoto o el uso de almacenamiento en la nube con S3.

Una experiencia valiosa que se ha aprendido también es el tener que utilizar una tecnología con funcionalidades nuevas en el mercado, implicando que hay poca documentación, y buscar una manera de poder lidiar con ello. En la formación académica se está utilizando principalmente tecnologías con un uso amplio en el mundo laboral, y pese a que rails no es una minoría, el utilizar la funcionalidad nueva de la última versión existente ha sido un desafío y una experiencia muy enriquecedora.

Con respecto al prototipo, la evolución del producto podría tomar varias direcciones. El primer paso que se tomaría antes de decidir la escala del producto sería una nueva iteración dedicada a crear los test que faltan, arreglando los posibles fallos que surjan. La primera funcionalidad que se podría implementar para mejorar la aplicación sería incluir más implicación de los usuarios, de manera que el producto no se quede en una herramienta meramente informativa para ellos. Para ello, se implementaría un sistema de correos en la aplicación, notificando a los usuarios pertinentes de acciones importantes relacionadas con sus viviendas, o que el administrador pueda recibir mensajes de los usuarios mediante la aplicación directamente, como la funcionalidad que no se implementó de solicitar movimientos a una vivienda. Tras hacer el barrido inicial de mejora del producto ya existente, se podría plantear aumentar la escala del producto, como se mencionó en el análisis (Sección 2), incluyendo una gestión de más comunidades, haciendo la aplicación más competitiva con respecto a las otras aplicaciones que vimos.

5. Documentación

Aquí tenemos todas las referencias con enlaces a sitios web y/o documentos que hayamos utilizado para el desarrollo y documentación del proyecto con las referencias de sitios web que hayamos hecho referencia en el documento, páginas con información general de donde hayamos obtenido información, y sitios web de donde hayamos utilizado alguna tecnología durante el proyecto.

Referencias

[1] 10 apps para comunidades de vecinos y administradores de fincas. https://www.prevent.es/blog-mis-vecinos/10-apps-para-comunidades-de-vecinos-y-administradores-de-fincas. Último acceso: 2018-05-19.

[2] Agencia española de protección de datos.

https://www.agpd.es/portalwebAGPD/index-ides-idphp.php.
Último acceso: 2018-05-19.

[3] Amazon web services, servicios en la nube.
https://aws.amazon.com/es/.
Último acceso: 2018-07-12.

[4] Comprendiendo la arquitectura mvc en rails.
https://www.sitepoint.com/
model-view-controller-mvc-architecture-rails/.
Último acceso: 2018-07-12.

[5] Diagramas de clase en uml.
https://www.lucidchart.com/pages/uml-class-diagram/.
Último acceso: 2018-07-12.

[6] Effective use cases por alistair cockburn, cómo escribir casos de uso.
http://alistair.cockburn.us/get/2465.
Último acceso: 2018-05-19.

[7] Ejemplos de la gema rails-erd.
https://voormedia.github.io/rails-erd/gallery.html/.
Último acceso: 2018-07-12.

[8] Extreme programming.
https://en.wikipedia.org/wiki/Extreme_programming.
Último acceso: 2018-05-19.

[9] Github, sistema de control de versiones.
http://github.com/.
Último acceso: 2018-05-19.

[10] Google drive, gestión de archivos del proyecto.
http://drive.google.com.
Último acceso: 2018-05-19.

[11] Gorails, screencasts de funcionalidades en rails actual.
https://gorails.com/.
Último acceso: 2018-05-19.

[12] Guías de ruby on rails oficiales.
http://guides.rubyonrails.org/.
Último acceso: 2018-07-12.

[13] Heroku, servidor de despliegue.
https://www.heroku.com/.
Último acceso: 2018-05-19.

[14] Introducción a diagramas de clase.
http://www.agilemodeling.com/artifacts/classDiagram.htm/.
Último acceso: 2018-07-12.

[15] Labels and crossreferencing, referencias del documento.
https://en.wikibooks.org/wiki/LaTeX/Labels_and_Cross-referencing.
Último acceso: 2018-05-19.

[16] Leyes de protección de datos europea.
https://ec.europa.eu/info/law/law-topic/data-protection_en.
Último acceso: 2018-07-12.

[17] Ninjamock, creador de mockups.
https://ninjamock.com.
Último acceso: 2018-05-19.

[18] Overleaf, desarrollo de la memoria en latex.
https://www.overleaf.com/.
Último acceso: 2018-05-19.

[19] Página web fynkus.
https://www.fynkus.com/.
Último acceso: 2018-05-19.

[20] Página web portalfincas.
http://www.portalfincas.com/.
Último acceso: 2018-05-19.

[21] Página web propietariosonline.
https://www.propietariosonline.com/.
Último acceso: 2018-05-19.

[22] Rails tutorial por michael hartl, tutorial principal seguido para el aprendizaje en rails.
https://www.railstutorial.org/book.
Último acceso: 2018-05-19.

[23] Railscasts, screencasts de funcionalidades en rails.
http://railscasts.com/.
Último acceso: 2018-05-19.

[24] Resumen de la arquitectura de rails para principiantes.
https://www.techcareerbooster.com/blog/ruby-on-rails-architecture-overview-for-beginners/.
Último acceso: 2018-07-12.

[25] Ruby on rails, framework utilizado para el desarrollo de la aplicación.
http://rubyonrails.org/.
Último acceso: 2018-05-19.

[26] Rubymine by jetbrains, entorno ide para rails.
https://www.jetbrains.com/ruby/.
Último acceso: 2018-05-19.

[27] Scrum, software development.
https://en.wikipedia.org/wiki/Scrum_(software_development)/.
Último acceso: 2018-07-12.

[28] Simulación de gestor fynkus.
https://www.fynkus.com/simulacion-gestor/.
Último acceso: 2018-05-19.

[29] Staruml, creador de casos de uso.
http://staruml.io/.
Último acceso: 2018-05-19.

[30] Taller de scrum laspalmasdevops.
http://laspalmasdevops.github.io/Talleres/slides/scrum.pdf/.
Último acceso: 2018-07-12.

[31] Test-drive development phases.
https://www.whizlabs.com/blog/what-is-tdd-and-its-phases/.
Último acceso: 2018-05-19.

[32] Test-driven development.
https://en.wikipedia.org/wiki/Test-driven_development.
Último acceso: 2018-05-19.

[33] Test-driven development essay agiledata.
http://agiledata.org/essays/tdd.html.
Último acceso: 2018-05-19.

[34] Test-first development rules.
http://www.extremeprogramming.org/rules/testfirst.html.
Último acceso: 2018-05-19.

[35] Test-first development xp rules.
http://docs.propietariosonline.com/.
Último acceso: 2018-05-19.

[36] Texexchange, resolver dudas con latex.
https://tex.stackexchange.com/.
Último acceso: 2018-05-19.

[37] Ubuntu 16.04, sistema operativo.
http://releases.ubuntu.com/16.04/.
Último acceso: 2018-05-19.

[38] Uso de bibtex, uso de las referencias y bibliografía.
http://www.bibtex.org/Using/.
Último acceso: 2018-05-19.

[39] Virtual box, entorno linux en máquina virtual.
https://www.virtualbox.org/.
Último acceso: 2018-05-19.

6. Anexos

Aquí irán todos los ficheros adicionales que no se hayan añadido al contenido de la memoria principal. Para el uso de la aplicación, el link del proyecto donde se encuentra alejado es: https://comunidad-vecinos.herokuapp.com.

Se adjunta también un fichero de prueba al proyecto con el nombre **extracto_completo.csv** para la importación del fichero que se ha utilizado a lo largo del desarrollo.

Las credenciales del administrador en la aplicación son:

- **Usuario:** admin@test.com
- **Contraseña:** chicken

6.1. Diagrama de casos de uso

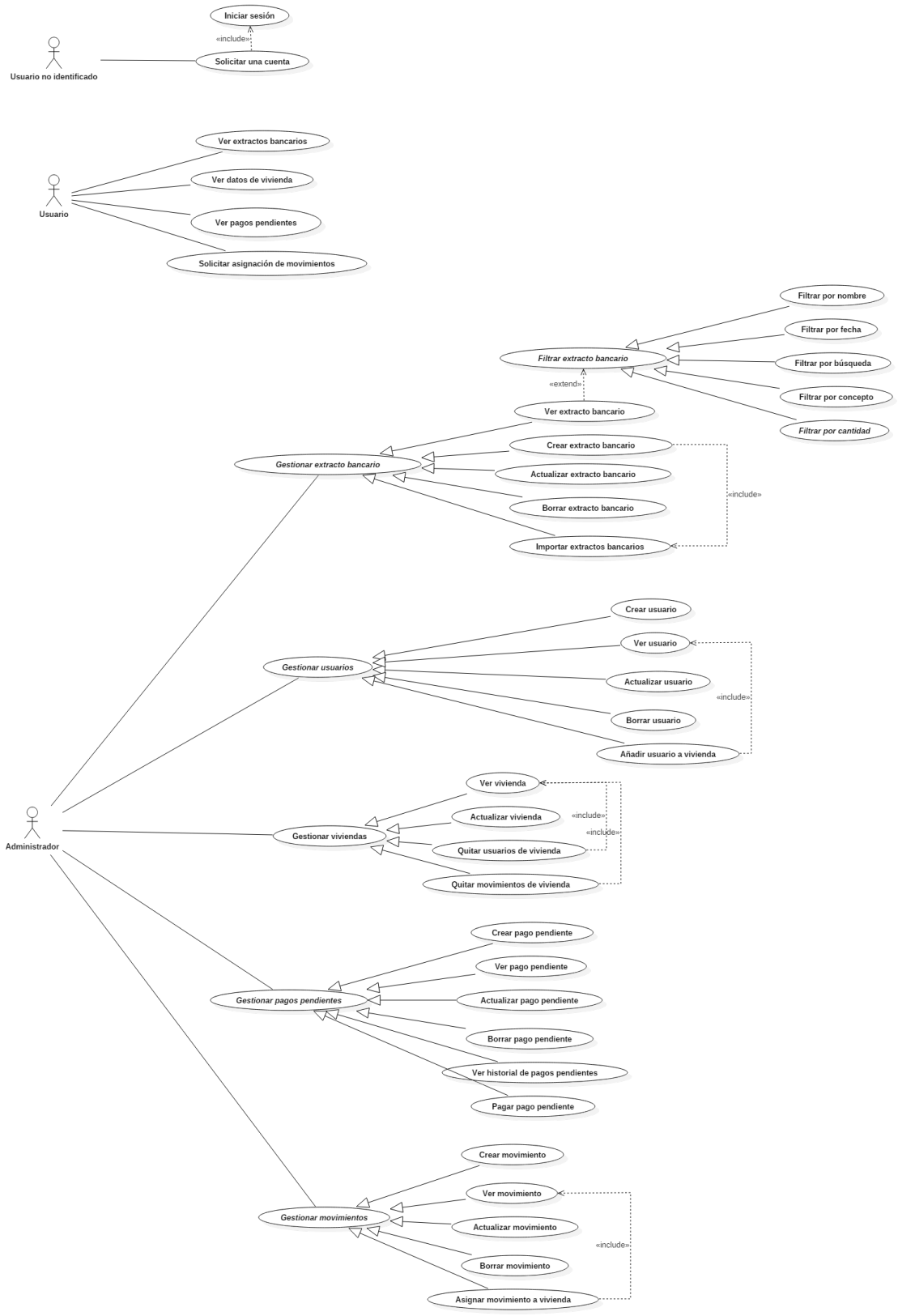

Figura 66: Casos de usos durante la etapa de análisis del proyecto

6.2. Especificación de casos de uso

Todas las especificaciones de casos de uso creadas durante la etapa de análisis previo al desarrollo se puede encontrar adjuntado como dos ficheros adicionales, uno para los de los usuarios, y otro del rol de administrador, con los nombres **especificacion_casos_de_uso_usuario.pdf** y **especificacion_casos_de_uso_administrador.pdf** respectivamente.

6.3. Mockups

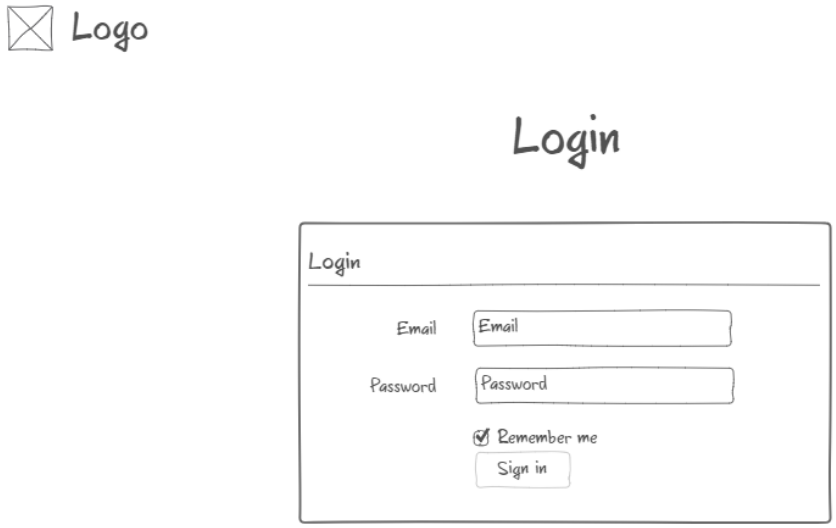

Figura 67: Pantalla de login

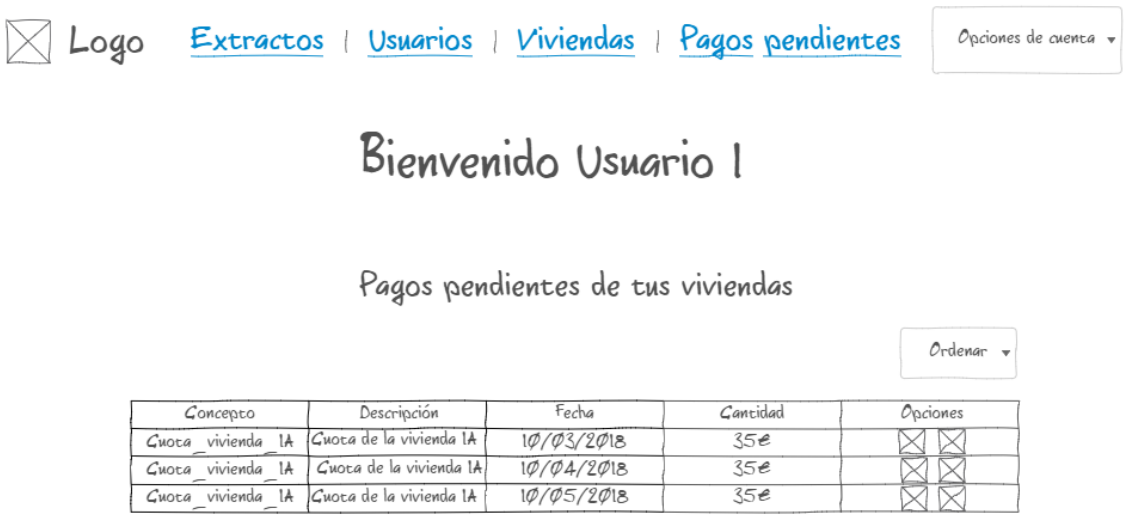

Figura 68: Pantalla principal

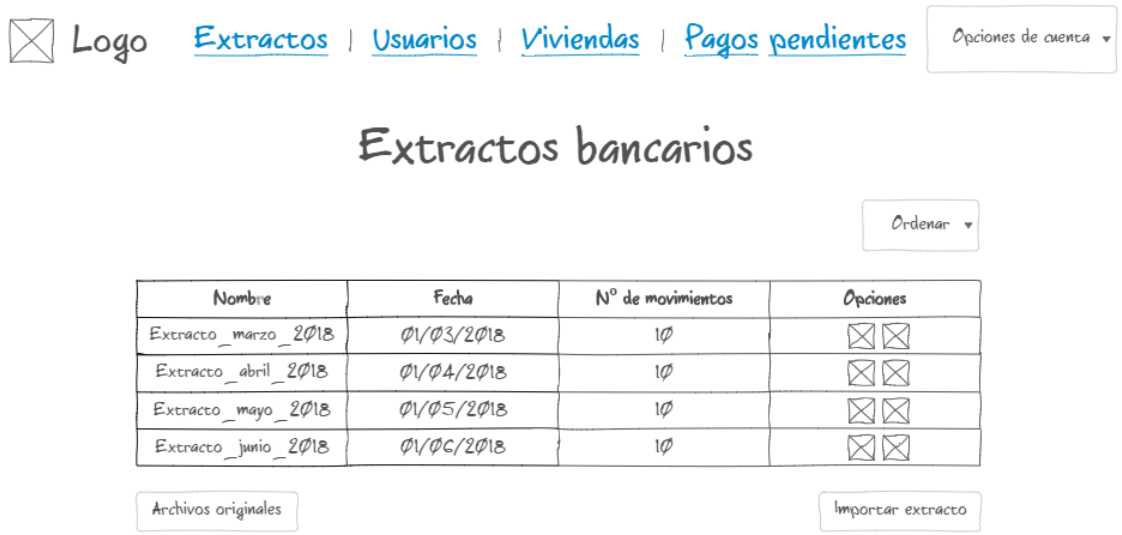

Figura 69: Pantalla de extractos bancarios

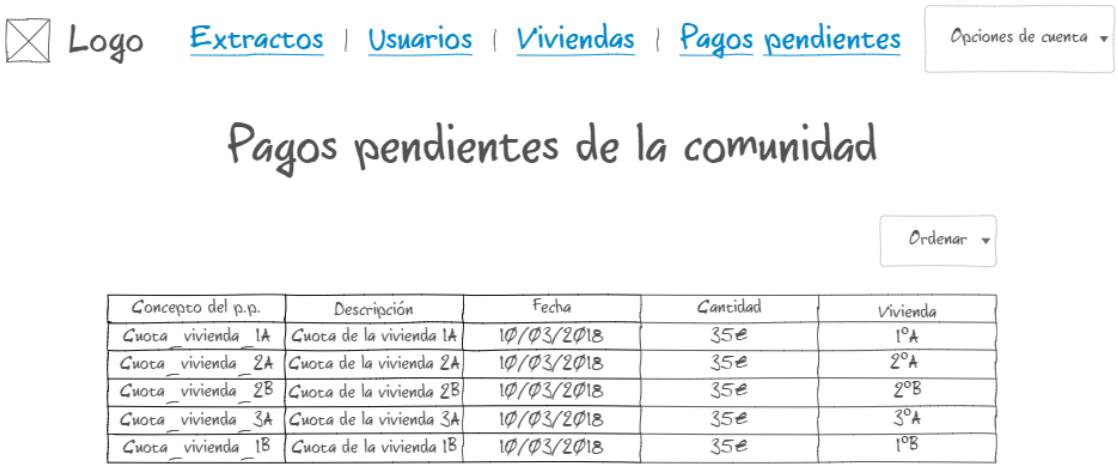

Figura 70: Pantalla de pagos pendientes de la comunidad

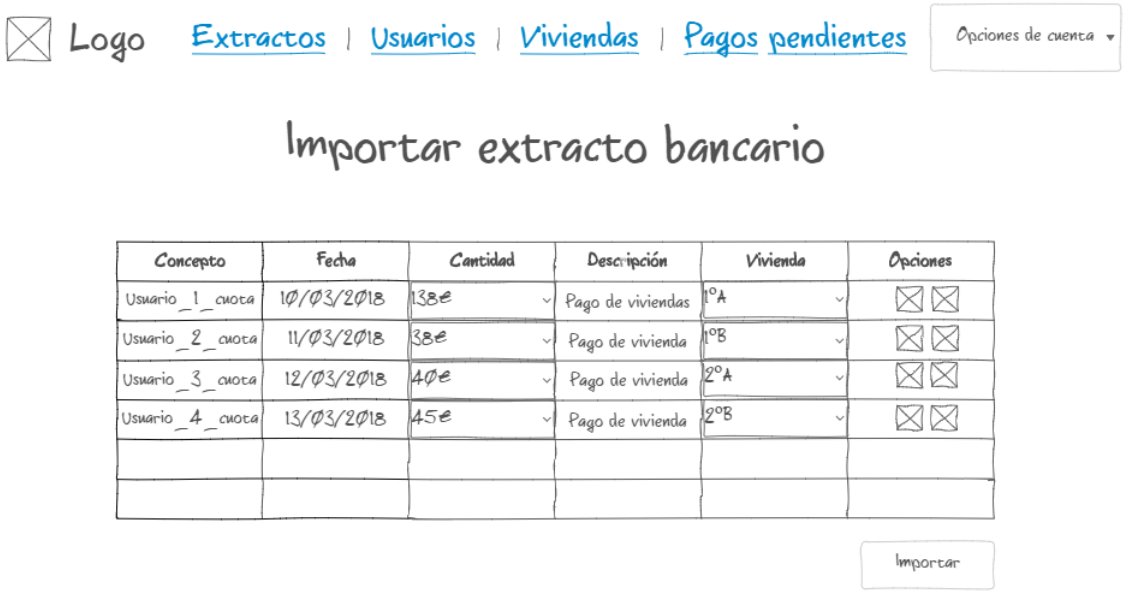

Figura 71: Pantalla de importar extracto

Figura 72: Pantalla de usuarios de la comunidad

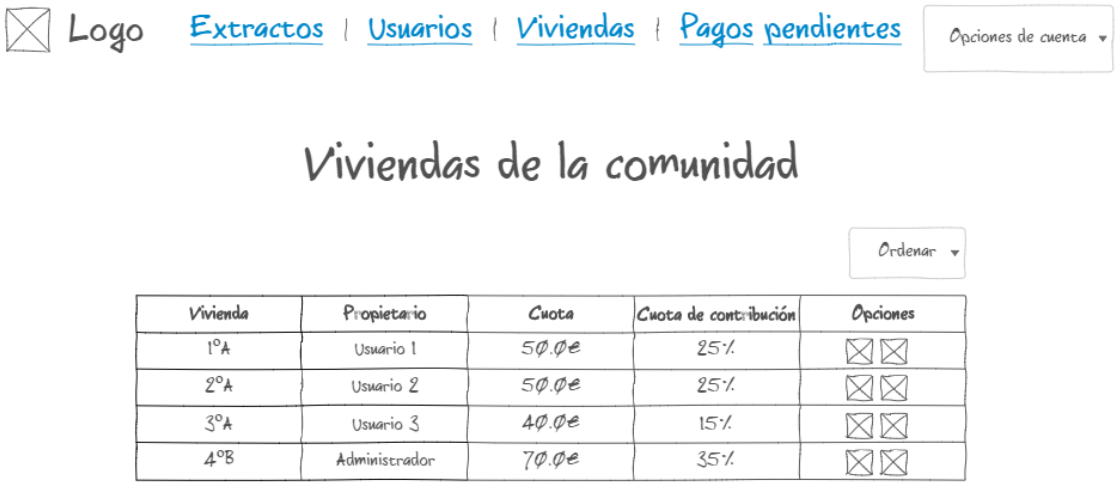

Figura 73: Pantalla de las viviendas de la comunidad

Figura 74: Pantalla del perfil de usuario

Figura 75: Pantalla del perfil de la vivienda

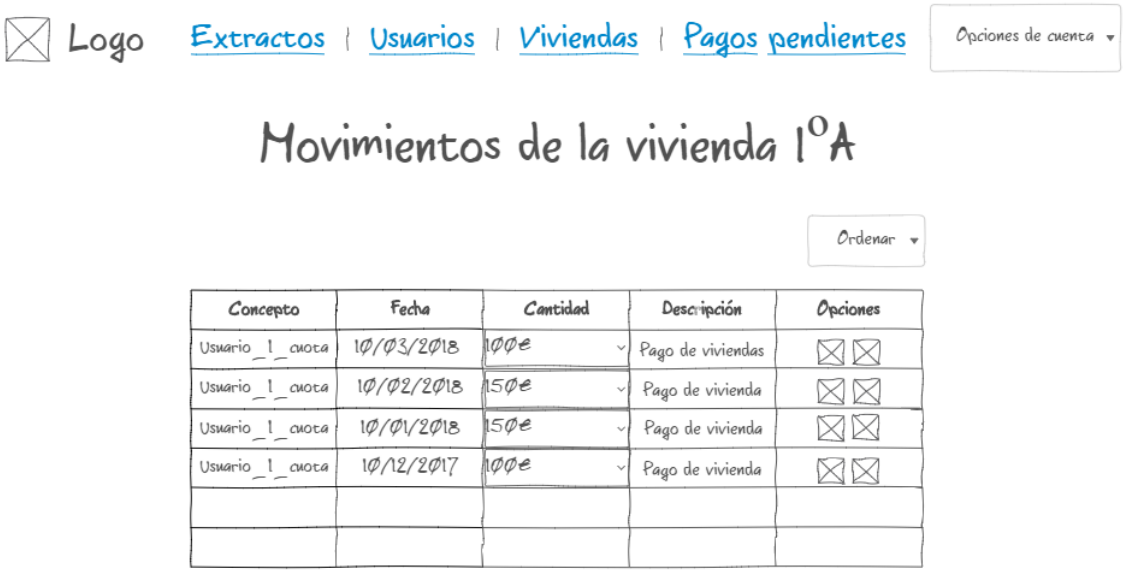

Figura 76: Pantalla de los movimientos de la vivienda

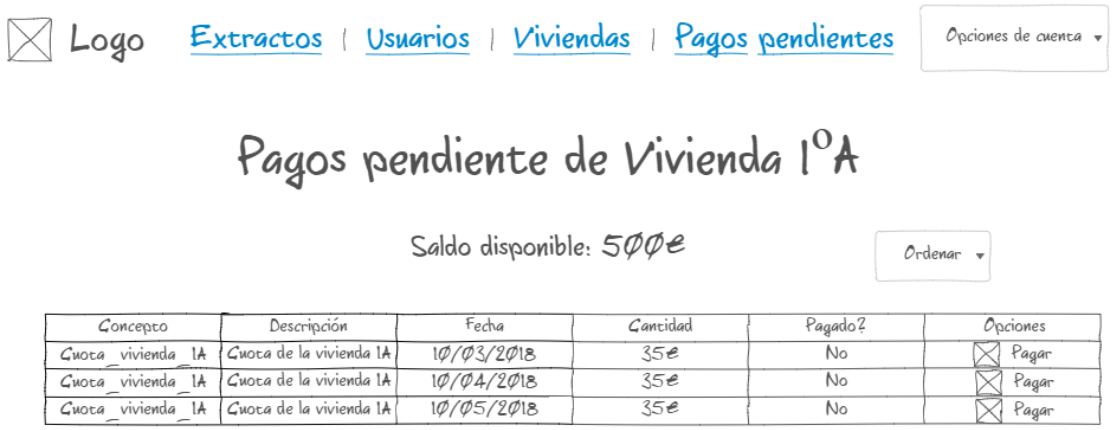

Figura 77: Pantalla de los pagos pendientes de la vivienda

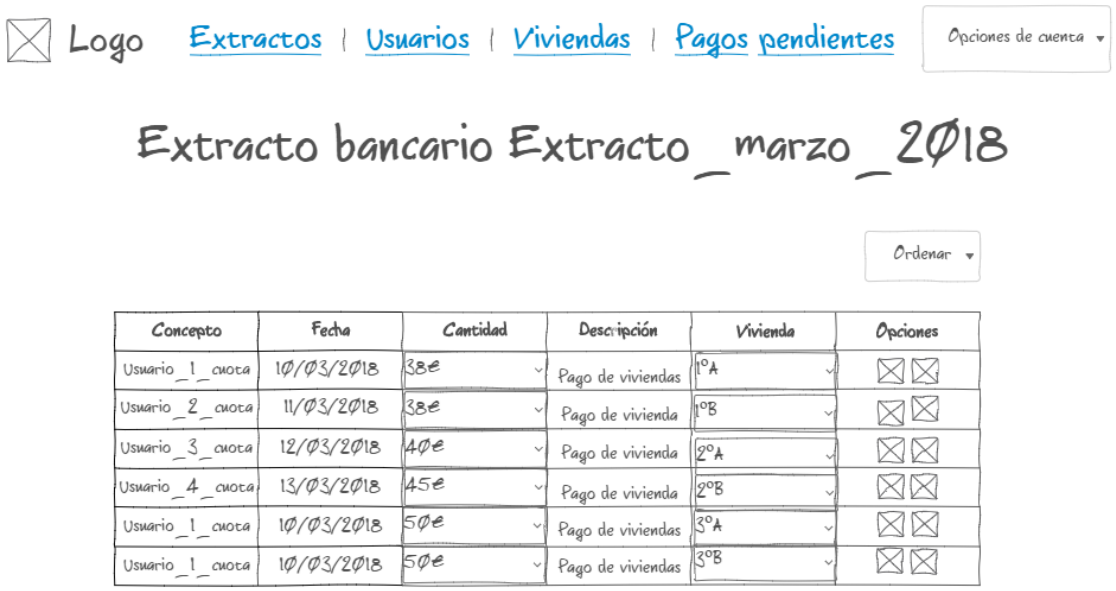

Figura 78: Pantalla del perfil del extracto

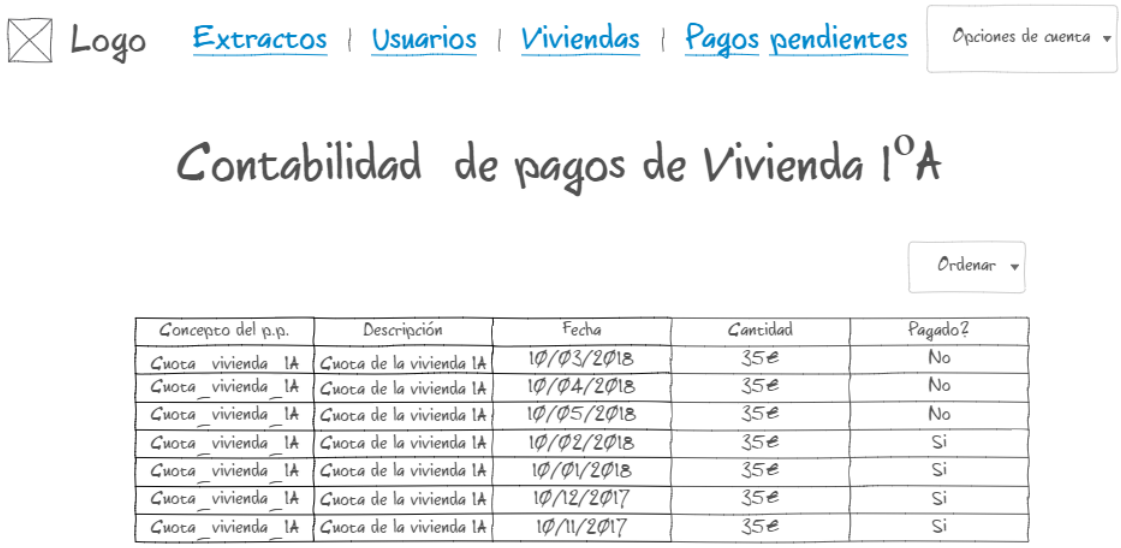

Figura 79: Pantalla de la contabilidad de pagos de una vivienda

Figura 80: Pantalla de los usuarios de una vivienda

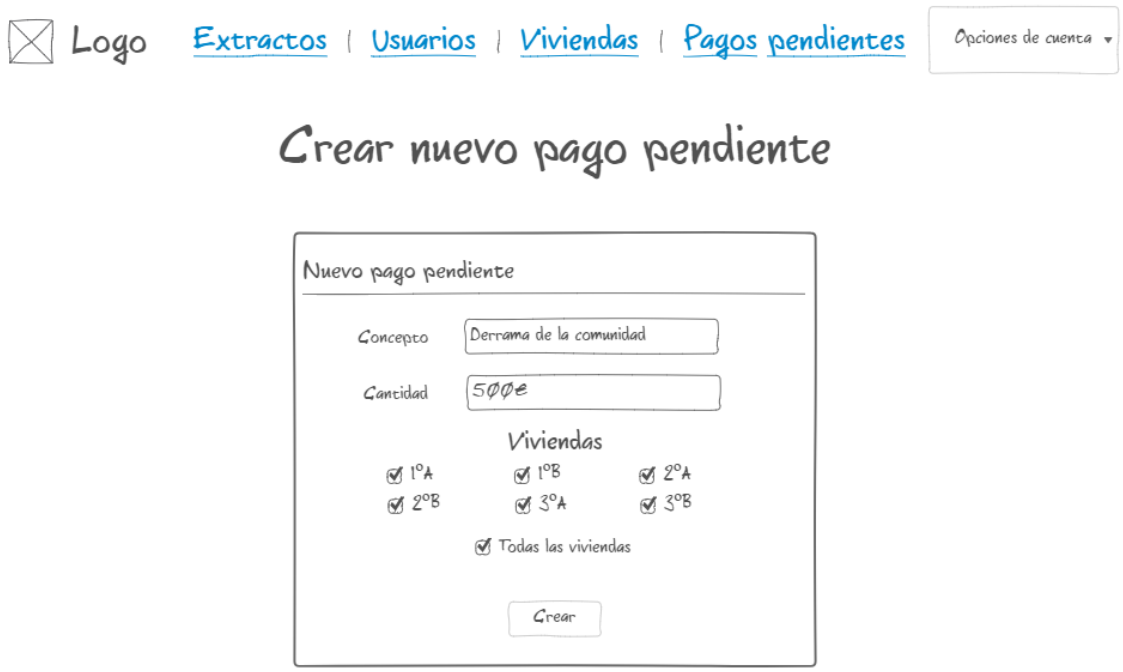

Figura 81: Pantalla de la creación de pagos pendientes

6.4. Diagrama de entidad-relación completa del proyecto

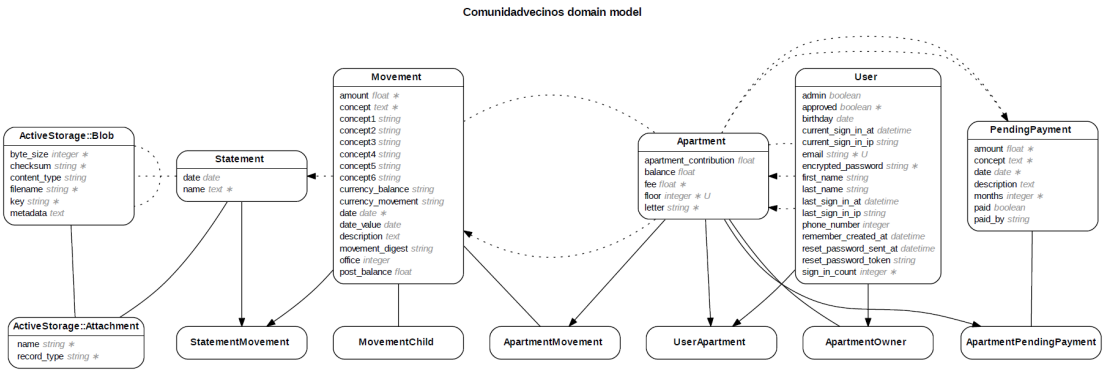

Figura 82: Diagrama de entidad-relación del proyecto

6.5. Estado del backlog durante el desarrollo del proyecto

Para ver el estado de todas las pilas de productos y de la iteración en cada fase del proyecto, tenemos un fichero excel adjuntado a la memoria como **product_backlog.xlsx**. No obstante, se aconseja la visualización del documento original en google spreadsheets, debido a que el formato utilizado en el fichero no se exportó como se creó originalmente.

Aquí tenemos las pilas de producto de la tercera iteración referenciada en la Subsubsección 3.5.3:

ID	Date Added	Task	Story points	Priority	Done?	Test?	Comments	Tiempo real	Date Finished
21	19/06/2018	Quiero poder quitar movimientos de una vivienda	3	Medium	☑	☑	Puede haber movimientos que sean de la comunidad a secas, no de un usuario	2	04/07/2018
26	19/06/2018	Quiero poder importar un extracto a partir de un CSV	8	High	☑	☐		8	02/07/2018
42	19/06/2018	Quiero disponer de filtros para ordenar las tablas en los extractos	8	Medium	☑	☐	http://railscasts.com/episodes/240-search-sort-paginate-with-ajax?autoplay=true	8	06/07/2018
45	22/06/2018	Las viviendas no pueden estar repetidas	3	Medium	☑	☑	Comprobar el piso y la letra si y existe en la base de datos	1	03/07/2018
47	22/06/2018	Creación de cuenta sea una solicitud con confirmación de admin	5	Low	☑	☐		2	08/07/2018
48	22/06/2018	Autocapitalize las letras de las viviendas	1	Low	☑	☑		1	03/07/2018
49	22/06/2018	Si no hay viviendas, poner un mensaje descriptivo de ello	1	Low	☑	☐		1	04/07/2018
53	24/06/2018	Arreglar CSS de los formularios	2	Low	☑	☐		3	06/07/2018
55	25/06/2018	Añadir un menú de opciones en las páginas que los requira como un glyphicon	3	Low	☑	☐	Se ha ido haciendo en todas las tablas de los modelos de la aplicacion	3	04/07/2018
59	29/06/2018	Si la pantalla no tiene datos, poner un mensaje descriptivo	3	Low	☑	☐	Abstracción del ID 49	1	04/07/2018
60	30/06/2018	Al quitar el movimiento de una vivienda, si el saldo acaba siendo negativo, cancelar el último pago pendiente	3	Medium	☑	☑	Hubo que contemplar el caso de que un movimiento podría cancelar más de 1 pago pendiente	3	04/07/2018
62	30/06/2018	Controlar la vista de usuarios normales y de admin	5	High	☑	☑		3	07/07/2018
65	03/07/2018	Quitar desplegables de los menús que no lo requieran	2	High	☑	☐		1	03/07/2018
66	03/07/2018	Quitar cantidades negativas de los formularios	1	High	☑	☐		1	03/07/2018
67	03/07/2018	Vista de viviendas con tablas e información relevante	2	Medium	☑	☐		2	03/07/2018
68	03/07/2018	Cambiar título y texto necesario de viviendas	1	Medium	☑	☐		1	03/07/2018
69	03/07/2018	Añadir propietario a la vivienda en función de un atributo de usuario propietario	2	Medium	☑	☑		2	05/07/2018
70	03/07/2018	Poner un nuevo atributo de cuota de participación a la vivienda	1	High	☑	☑		1	03/07/2018
71	03/07/2018	Añadir un logo	1	High	☑	☐		2	03/07/2018
72	03/07/2018	Quitar crear nuevo extracto bancario	1	Medium	☑	☐		1	02/07/2018
73	03/07/2018	Ver todas las opciones de los extractos con botones	2	Medium	☑	☐		1	02/07/2018
74	03/07/2018	Botones duplicados de importar extracto, al principio y final de tabla	1	Low	☑	☐		1	02/07/2018
75	03/07/2018	Quitar botones que ya estén utilizados en la vista de extractos	1	Low	☑	☐		1	02/07/2018
76	03/07/2018	Quitar repetición del texto Extracto en las descripciones	1	Low	☑	☐		1	02/07/2018
77	03/07/2018	Los movimientos de extractos y movimientos deberían mostrar la misma información relevante a primera vista	2	High	☑	☐		1	02/07/2018
78	03/07/2018	Cambiar el modelo movimiento para todos los campos del CSV	2	High	☑	☐		2	02/07/2018
79	03/07/2018	Dividir las cantidades de un movimiento utilizando asociaciones entre los movimientos	5	Medium	☑	☐	Falta que al borrar los movimientos de uno en uno de la división, se actualicen, o que se bloqueen por que está incompleto, o que se borren todos los hijos del tirón.	8	05/07/2018

Figura 83: Pila de producto de la tercera iteración

80	03/07/2018	Cambiar títulos y texto redundante o repetido de pagos pendientes	1	Medium	☑	☐		1	03/07/2018
81	03/07/2018	Mostrar saldo disponible dentro de los pagos pendientes de la vivienda	1	Medium	☑	☐		1	03/07/2018
82	03/07/2018	Crear pagos pendientes para todas las viviendas de sus cuotas con un solo botón	2	High	☑	☐	Esto tardó un poco más porque hubo que cambiar la lógica de las viviendas también, al final la ruta de creación actual se utilizó para esto	5	03/07/2018
83	03/07/2018	Cambiar el crear pago actual para que se utilice para derramas	5	High	☑	☐		3	03/07/2018
84	03/07/2018	Pagos pendientes deberían estar ordenados por defecto de menor a mayor	2	Medium	☑	☐		2	06/07/2018
85	03/07/2018	Buscar una manera de bloquear todos los pagos menos el más antiguo	3	High	☑	☐		1	05/07/2018
86	03/07/2018	Cambiar la vista del usuario, como el botón de cancelar cuenta	1	Low	☑	☐		1	05/07/2018
87	03/07/2018	Añadir una vista de usuarios con datos de contacto	1	Medium	☑	☐		1	05/07/2018
88	03/07/2018	Añadir al usuario un atributo teléfono	1	Medium	☑	☐		1	05/07/2018
89	03/07/2018	Bloquear el edit del usuario para que solo el admin o el propio usuario pueda cambiar determinados datos	3	High	☑	☐	Los usuarios solo podrán realizar cambios en su cuenta.	2	05/07/2018
90	03/07/2018	Añadir al usuario una asociación con los pagos pendientes a la hora de pagar	3	Medium	☑	☐	Al generarse pagos pendientes porque no hay saldo en la vivienda, el pagador asignado se quita del pago pendiente	3	05/07/2018
91	03/07/2018	Cambiar quitar usuarios de vivienda si el usuario es el propietario	3	Medium	☑	☑	Añadido toda la lógica entre propietarios y usuarios de la vivienda	3	05/07/2018
92	03/07/2018	Login directamente en la página principal si no estás logueado	1	Low	☑	☐		1	05/07/2018
93	03/07/2018	Quitar opción de asociar a vivienda si es negativo el movimiento en extractos	1	Medium	☑	☐		1	04/07/2018
94	04/07/2018	Movimientos negativos no se pueden dividir	1	High	☑	☐		1	05/07/2018
95	05/07/2018	Admin borrar otros usuarios	1	Low	☑	☐		1	05/07/2018
61	30/06/2018	Before action de admin en application controller, poniendo en los controladores las opciones en las que hay que ser admin	5	Medium	☑	☐	Esto se había ido haciendo durante el desarrollo	5	07/07/2018
96	06/07/2018	Quitar restricción de fecha y poner fecha por defecto	1	High	☑	☐		1	06/07/2018
97	06/07/2018	Arreglar error en importación de extracto	3	High	☑	☐		3	06/07/2018
98	06/07/2018	Cambiar glyphicons de botones a texto	3	Medium	☑	☐		3	07/07/2018
99	06/07/2018	Añadir bootstrap a todas las páginas	5	High	☑	☐		3	07/07/2018
100	06/07/2018	Cambiar los formularios y utilizar bien bootstrap	5	High	☑	☐		3	07/07/2018
101	06/07/2018	Añadir pantalla con CSV de la nube	3	Low	☑	☐		2	08/07/2018
					☐	☐			
		Total:	125		☐	☐	Total:	110	

Figura 84: Continuación de la pila de producto de la tercera iteración