Grado en Ingeniería Informática

Trabajo de Fin de Grado

Prototipo de aplicación Ruby on Rails para un servicio de almacenamiento en la nube que emplee un sistema de pago con Amazon Web Services

---

Andrew De Abreu Ruiz

Tutor/es

D. Carmelo Cuenca Hernández

Dª. Francisca Quintana Domínguez

Las Palmas de Gran Canaria, 19 de julio de 2019

Agradecimientos

Me gustaría dedicar este trabajo a mi familia, a mis padres y a mi abuela. Os doy las gracias por estar siempre ahí, por apoyarme incondicionalmente, por vuestros consejos y por vuestro esfuerzo.

Agradecer a todos los profesores que han contribuido en mi formación durante el Grado en Ingeniería Informática.

Agradecer también a todos los compañeros que he tenido la oportunidad de conocer en esta etapa.

Y sobre todo, me gustaría agradecer a mis tutores, por su tiempo, consejos y dedicación que han permitido llevar a cabo este trabajo.

Resumen

Este trabajo tiene como objetivo principal desarrollar un prototipo de aplicación web con el *framework* Ruby on Rails para administrar un *bucket* de Amazon S3 de manera similar a un servicio de almacenamiento en la nube actual. En esta aplicación los usuarios podrán subir, almacenar, compartir y descargar archivos o carpetas. Según el tipo de cuenta y el nivel de autorización, los usuarios pueden disponer de una cantidad de almacenamiento específica. Aquí se distinguen dos tipos de usuarios, el usuario cliente y el usuario administrador. Ambos dispondrán de un panel de administración para gestionar el espacio de almacenamiento y el uso del servicio respectivamente. Para implementar las distintas cantidades de almacenamiento que puede tener un usuario en su cuenta, se integra la aplicación con un servicio de procesamiento de pagos en línea que permite definir un esquema de suscripciones a un plan gratuito o de pago. Por tanto, los usuarios podrán adquirir una suscripción a un plan de pago, cancelarla o cambiarla por otra.

Palabras clave: Ruby on Rails, servicio de almacenamiento en la nube, servicio de procesamiento de pagos, Stripe, aplicación web

Abstract

The main objective of this work is to develop a web application prototype with the Ruby on Rails framework to manage an Amazon S3 bucket in a similar way to a current cloud storage service. In this application, users upload, store, share and download files or folders. Depending on the type of account and the level of authorization, users can have a specific amount of storage. Here you can distinguish two types of users, the client user and the administrator user. Both will have an administration panel to manage the storage space and the use of the service respectively. To implement the different amounts of storage that a user can have in their account, the application is integrated with an online payment processing service that allows defining a subscription scheme for a free or paid plan. Therefore, users can purchase a subscription to a payment plan, cancel or exchange for another.

Keywords: Ruby on Rails, cloud storage service, payment processing service, Stripe, web application

Índice

Índice de figuras

Índice de tablas

1. Introducción

1.1. Motivación

Amazon S3 es un servicio de almacenamiento que ofrece Amazon en su plataforma de computación en la nube, Amazon Web Services (AWS). Este servicio está basado en una arquitectura de objetos. Los usuarios pueden subir, almacenar y descargar cualquier tipo de archivo en forma de objetos que tenga un tamaño de hasta 5 TB y con el proceso de carga limitado a 5 GB [1]. Cada objeto se almacena en un *bucket* que se encuentra repartido por la estructura del centro de datos de Amazon de manera que cada usuario sólo puede acceder a sus propios *buckets* [2].

El acceso a la gestión del almacenamiento en Amazon S3 se realiza mediante una interfaz web que muestra la información almacenada en los *buckets* y desde la que se pueden hacer uso de todas las opciones que proporciona el servicio. El principal inconveniente es que esta interfaz parece estar orientada a un usuario administrador puesto que ofrece funciones poco útiles si quién lo utiliza no es una empresa que las requiera. Por tanto, si un usuario viene de utilizar servicios de almacenamiento en la nube como Dropbox o Mega le puede resultar difícil y poco cómodo trabajar con esta interfaz.

Además, si queremos que otros usuarios puedan acceder a nuestro *bucket* deben disponer de una cuenta de usuario de AWS para que así se le pueda habilitar el acceso mediante listas de control de acceso (ACL) y políticas de *bucket*. Esta cuenta de usuario también es necesaria para emplear la opción de *Pago por el solicitante* en el que el dueño del *bucket* no es el que paga por el servicio si no los usuarios con quiénes lo comparte. Sin embargo, con esta opción no se pueden percibir ingresos [3].

Por estos motivos, en este trabajo se concibe el desarrollo de una aplicación web que permita gestionar un *bucket* de Amazon S3 de manera similar a como se hace con los servicios de almacenamiento en la nube actuales.

Esta aplicación permite a los usuarios subir, descargar y compartir archivos o carpetas. Según el tipo de cuenta y el nivel de autorización, los usuarios pueden disponer de una cantidad de almacenamiento específica. Para gestionar el espacio de almacenamiento, los usuarios dispondrán de un panel de administración desde el cual pueden consultar los archivos que tienen almacenados, ver sus estadísticas de uso y subir, descargar o compartir sus archivos. Por otro lado, existirá un usuario administrador que dispone de un panel de administración desde

el que puede consultar las estadísticas de uso del servicio, gestionar los usuarios, consultar el historial de pagos, etc.

Para implementar las distintas cantidades de almacenamiento que puede tener disponible un usuario en su cuenta, se integra la aplicación con un servicio de procesamiento de pagos en línea que permite definir un esquema de suscripciones a un plan gratuito o de pago. Por tanto, los usuarios podrán adquirir una suscripción a un plan de pago, cancelarla o cambiarla por otra. Este esquema nos permitirá percibir ingresos con la aplicación lo cual no es posible administrando directamente un *bucket* de Amazon S3.

1.2. Objetivos

Lo expuesto en la sección anterior nos permite plantearnos una serie de objetivos tanto a nivel de proyecto como a nivel académico que se pretenden cumplir con la realización de este trabajo.

1.2.1. Objetivos generales y específicos

El principal objetivo es desarrollar un prototipo de aplicación web que permita la gestión de un *bucket* de Amazon S3 de manera similar a un servicio de almacenamiento en la nube.

Para lograr este objetivo se establecen una serie de objetivos más específicos que se cumplirán con el desarrollo del prototipo:

- Implementar un sistema de autenticación y de autorización de usuarios.
- Diseñar e implementar un sistema de gestión de archivos y carpetas.
- Diseñar e implementar un panel de administración para los usuarios.
- Diseñar e implementar un panel de administración para los usuarios administradores.
- Integrar un servicio de procesamiento de pagos por Internet para la definición de un esquema de suscripciones a planes de pago.

1.2.2. Objetivos académicos

A nivel académico se pretende alcanzar los siguientes objetivos:

- Aprender a desarrollar aplicaciones web con el *framework* Ruby on Rails y en el lenguaje de programación Ruby.

- Poner en práctica los conocimientos teóricos y prácticos adquiridos durante los estudios universitarios del Grado de Ingeniería Informática de la Universidad de Las Palmas de Gran Canaria mediante el desarrollo de un prototipo de aplicación web de carácter profesional.

- Completar la formación en el uso de los servicios de computación en la nube que proporciona la plataforma de Amazon Web Services.

- Llevar a cabo la planificación y consecución de un proyecto en sus distintas fases: análisis/estudio, diseño, desarrollo, pruebas y despliegue como preparación para el mundo profesional.

- Aprender y desplegar una aplicación web en un entorno de producción.

1.3. Justificación de las competencias cubiertas

Las competencias específicas cubiertas en la realización de este Trabajo de Fin de Grado son las siguientes:

- **CII08.** Capacidad para analizar, diseñar, construir y mantener aplicaciones de forma robusta, segura y eficiente, eligiendo el paradigma y los lenguajes de programación más adecuados.

  Esta competencia se ve cubierta mediante la planificación y elaboración de un prototipo en sus fases de análisis, diseño, desarrollo, prueba y despliegue empleando el *framework* Ruby on Rails que adopta el patrón arquitectónico Modelo Vista Controlador (MVC) y el lenguaje de programación Ruby cuyas características los hacen adecuados para este proyecto.

- **CII016.** Conocimiento y aplicación de los principios, metodologías y ciclos de vida de la ingeniería de software.

Esta competencia se ve cubierta mediante la aplicación de principios, metodologías y ciclos de vida aprendidos, a lo largo del Grado en Ingeniería Informática, en las distintas asignaturas de la rama de Ingeniería del Software. Para ser más precisos, se emplea una metodología de desarrollo iterativo e incremental.

- **IS01.** Capacidad para desarrollar, mantener y evaluar servicios y sistemas software que satisfagan todos los requisitos del usuario y se comporten de forma fiable y eficiente, sean asequibles de desarrollar y mantener y cumplan normas de calidad, aplicando las teorías, principios, métodos y prácticas de la ingeniería del software.

  Esta competencia se ve cubierta de la misma forma que las anteriores. El prototipo desarrollado satisface los requisitos definidos en la fase de análisis del plan de trabajo llevado a cabo.

- **IS04.** Capacidad de identificar y analizar problemas y diseñar, desarrollar, implementar, verificar y documentar soluciones software sobre la base de un conocimiento adecuado de las teorías, modelos y técnicas actuales.

  Esta competencia se ve cubierta de la misma forma que las anteriores. El prototipo se ha desarrollado haciendo uso de las teorías, modelos y técnicas actuales aprendidas en las asignaturas del Grado (UML, requisitos funcionales y no funcionales, patrones arquitectónicos, patrones de diseño, etc.)

1.4. Planificación inicial

El plan de trabajo propuesto para lograr la consecución de los objetivos establecidos al inicio del proyecto se resume en la Tabla 1-1. Este plan de trabajo se desglosa en una serie de fases que siguen un orden lógico. Cada fase, a su vez, se divide en una serie de actividades concretas que debemos realizar para completarla.

Plan de trabajo		
Fases	**Duración estimada (horas)**	**Tareas**
Estudio previo / Análisis	80	• Estudio de las tecnologías y herramientas necesarias para el desarrollo del proyecto. • Análisis de la aplicación. • Elaboración de un mockup de la aplicación.
Diseño / Desarrollo / Implementación	145	• Iteración 1: Diseño e implementación del sistema de usuarios. • Iteración 2: Diseño e implementación del sistema para gestionar los archivos. • Iteración 3: Diseño e implementación del panel de control del usuario cliente. • Iteración 4: Diseño e implementación del panel de control del usuario administrador. • Iteración 5: Diseño e implementación de un prototipo del sistema de pago para cobrar por los servicios ofrecidos en la aplicación. • Diseño e implementación de la interfaz de usuario.
Evaluación / Validación / Prueba	25	• Evaluación, validación y prueba de la aplicación web.
Documentación / Presentación	50	• Elaboración de la memoria del trabajo. • Elaboración del manual de usuario.

Tabla 1-1. Plan de trabajo inicial

Este plan de trabajo se ha cumplido en su mayoría, salvo en la fase de *Diseño / Desarrollo / Implementación* en la que se ha tenido que realizar una serie de ajustes motivados por el conocimiento adquirido en el estudio de las tecnologías, herramientas y servicios necesarios para el desarrollo del proyecto en la fase de *Estudio previo / Análisis*. Estos cambios tienen que ver con el orden en el que se realizan las iteraciones y el hito a lograr en cada una de ellas tal y como se puede ver a continuación:

- Iteración 1: Sistema de autenticación de usuarios con Devise.

- Iteración 2: Integración de modelo de pagos recurrentes de Stripe para cobrar por los servicios ofrecidos en la aplicación.

- Iteración 3: Implementación del sistema de archivos y carpetas.
- Iteración 4: Implementación de las funcionalidades del usuario administrador.
- Diseño e implementación de la interfaz de usuario.

Cabe destacar que en el plan de trabajo inicial se contemplaba el diseño e implementación del panel de administración del usuario cliente y del usuario administrador como una iteración aparte. No obstante, se optó por definir que una iteración englobaba las funcionalidades que estaban más relacionadas con el hito a alcanzar en ella. Por lo tanto, los paneles de administración de ambos tipos de usuarios se dejaban como subtareas a realizar durante estas iteraciones.

1.5. Metodología

Este trabajo se lleva a cabo mediante una metodología de desarrollo iterativo e incremental [4] con una serie de variaciones para adaptarla al proceso de desarrollo de nuestro prototipo.

Este proceso se divide en una serie de bloques temporales llamados iteraciones. En todas las iteraciones se repite el mismo proceso de trabajo con el objetivo de obtener un incremento que nos acerque al objetivo final que será desarrollar el prototipo funcional. Este proceso de trabajo está compuesto de las fases de diseño, implementación, pruebas y despliegue. Antes de iniciar la primera iteración se realiza el estudio previo/análisis con el que obtendremos los requisitos y casos de uso a implementar durante el proceso. En cada iteración se establecerá un hito a alcanzar y se incluirán los casos de uso relacionados con este.

Una de las ventajas que ofrece emplear esta metodología es que nos permite evolucionar la aplicación a partir de los incrementos obtenidos de las iteraciones anteriores, añadiendo nuevos requisitos o mejorando los que ya fueron completados si se requiere necesario.

1.6. Estructura de la memoria

En esta sección se presenta la estructura escogida para la organización de los elementos de la memoria con el fin de facilitar su lectura. Hemos decidido dividirla en diez apartados, alguno de ellos, con sus respectivas secciones. Es recomendable seguir dicha estructura puesto que cada apartado se corresponde en cierta medida con una de las fases del desarrollo del proyecto, a excepción de la introducción, de las conclusiones y de los anexos. A continuación, describiremos brevemente cada apartado:

- **Introducción.** En este apartado se expone la motivación que lleva a realizar este trabajo, se detallan los objetivos que se han propuesto alcanzar tanto a nivel general como a nivel académico, se justifican las competencias cubiertas en este y se establece el plan de trabajo y la metodología a seguir para lograr la consecución de estos objetivos.

- **Análisis.** En este apartado se explica todo el análisis realizado antes de comenzar con el desarrollo del prototipo.

- **Diseño.** En este apartado se comenta el diseño arquitectónico empleado y los componentes principales de nuestro modelo de la base de datos.

- **Desarrollo.** En este apartado se detalla cómo ha sido el proceso de desarrollo del prototipo. Primero se exponen que características o funciones se han implementado y luego se explican las iteraciones realizadas. En cada una se comentan los aspectos más importantes de su desarrollo.

- **Recursos utilizados.** En este apartado se enumeran los lenguajes de programación, las tecnologías, los servicios, las gemas y las herramientas empleadas para llevar a cabo el proceso de desarrollo. Se indica que hacen y para que se han utilizado en este trabajo.

- **Pruebas.** En este apartado se indica el conjunto de pruebas que se han realizado al prototipo para validar su correcto funcionamiento.

- **Despliegue.** En este apartado se describirán los aspectos más importantes del despliegue del prototipo en un entorno de producción.

- **Conclusiones.** En este apartado se concluye el desarrollo del trabajo. Se comenta el grado de consecución de los objetivos, se presentan las posibles líneas de trabajo de cara a una primera versión de la aplicación y se expone una valoración personal.

- **Documentación.** En este apartado se enumera cada una de las referencias a las fuentes que se han consultado durante la elaboración del trabajo. Para cada una de ellas se indican sus autores y el año. En el caso de las referencias web también se indica la fecha a la que se accedió por última vez. Para ambos tipos de referencias se empleará el formato IEEE.

- **Anexos.** En este apartado se adjunta información adicional que no se ha incluido en los apartados anteriores. Por ejemplo, el manual de usuario que explica detalladamente como hacer uso de la aplicación por parte de los usuarios, los diagramas de casos de uso, las vistas del *mockup*, etc.

2. Análisis

2.1. Estudio de aplicaciones similares

Para comenzar el análisis de la aplicación se realiza un estudio de las aplicaciones web de los servicios de almacenamiento en la nube existentes en el mercado. Esta parte es fundamental puesto que nos permitirá decidir qué funcionalidades incluir en nuestra aplicación y valorar si incluir otras que no estén presentes en esas aplicaciones haciendo que se diferencie del resto. Teniendo en cuenta que existe una gran cantidad de estos servicios, hemos optado por escoger a MEGA, Dropbox y pCloud para este estudio. Se han dejado a un lado otros servicios populares como Google Drive, OneDrive, Box o Sync ya que ofrecen características similares a los anteriores.

Para limitar el alcance de este estudio nos centraremos solamente en el modelo de negocio orientado a usuarios particulares y no a empresas. Dicho esto, se comentan las características principales, dejando a un lado las que no tengan que ver con los objetivos del proyecto, y se consideran aspectos como la experiencia de usuario, la facilidad de uso y el diseño de la interfaz.

2.1.1. MEGA

MEGA	
Desarrollador(es)	MEGA Ltd
Autor(es)	Kim Dotcom
Tipo	Servicio de almacenamiento en la nube
Versión	3.56.3
Escrito en	C++, JavaScript, Java, Objective-C
Licencia	Mega Limited Code Review License
Multiplataforma	Sí
Multilingüe	Sí
Año de lanzamiento inicial	Enero de 2013
Sitio oficial	https://mega.nz

Tabla 2-1. Ficha técnica de MEGA [Wikipedia]

En primer lugar, se hace el análisis de la aplicación web de MEGA. Esta permite realizar funcionalidades esenciales como subir, descargar, renombrar, mover, copiar, buscar, compartir, eliminar y marcar como favoritos archivos o carpetas. También se pueden visualizar archivos si se tratan de imágenes, vídeos o documentos. A diferencia de las aplicaciones web de Dropbox y pCloud, se permite etiquetar los archivos o carpetas con una serie de colores. Esto es útil puesto que podemos filtrarlos en base a ello. Por otra parte, también podemos ordenar los archivos o carpetas por nombre, etiqueta, fecha de creación o favoritos.

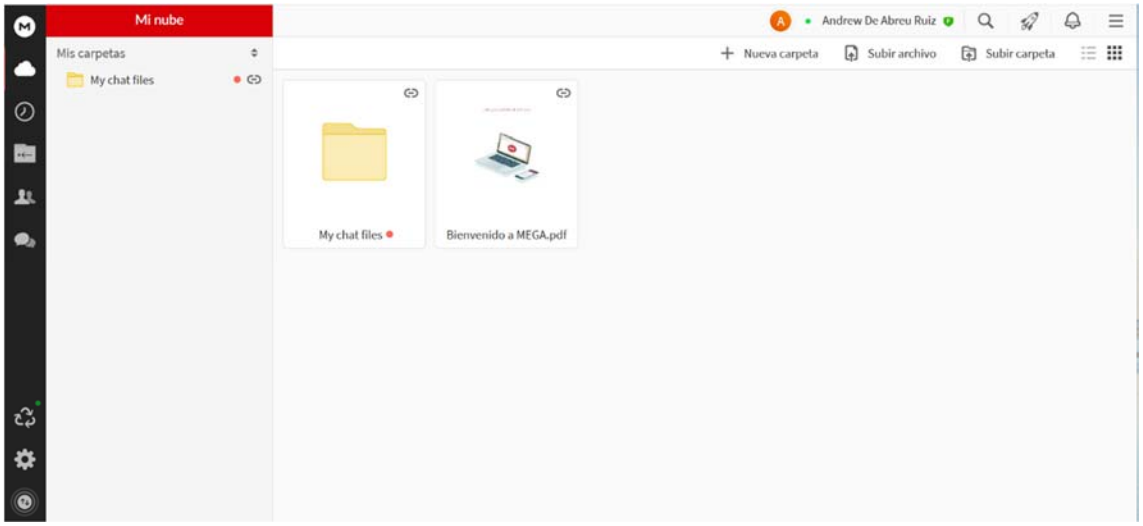

Figura 2-1. Página principal de la aplicación web de MEGA [Captura de pantalla]

La compartición de un archivo o carpeta se puede hacer mediante una invitación, indicando las direcciones de correo electrónico de los usuarios destinarios y estableciendo los permisos de acceso, o mediante la generación de un enlace. Si queremos algo de seguridad, MEGA permite proteger el enlace con una clave de cifrado de manera que solo la persona a la que se le pase la clave puede leerlo. Esta puede adjuntarse con el enlace o pasarse por separado. Por otro lado, si adquirimos alguno de los planes de pago que ofrece MEGA podemos darles una fecha de caducidad o protegerlos con una contraseña.

La interfaz de usuario es una de las fortalezas de la aplicación de MEGA. Es fácil de usar, atractiva y sencilla. De aquí hay que destacar el menú de navegación dado que permite acceder a cualquier parte de la aplicación independientemente de en qué vista nos encontremos, el panel de control puesto que muestra la información de forma clara y el panel de ajustes ya que podemos configurar cualquier aspecto de la cuenta de usuario fácil y rápidamente (véase la Figura 2-1 y la Figura 2-2). Todas estas consideraciones se tendrán en cuenta para el diseño de la interfaz de usuario de nuestra aplicación.

La carencia de herramientas de colaboración es una de las principales debilidades de su aplicación web. No se pueden crear ni editar archivos, aunque si crear carpetas. La razón de esto es que no se integra con herramientas ofimáticas online como Google Drive o Office. Tampoco podemos añadir comentarios a los archivos para que puedan verlos otros usuarios. Sin embargo, ofrece una característica que no se presentan en Dropbox ni en pCloud y es que se dispone de una aplicación de chat con la que podemos conversar con otros usuarios, si los tenemos agregados como contactos, y transferir archivos o carpetas con ellos.

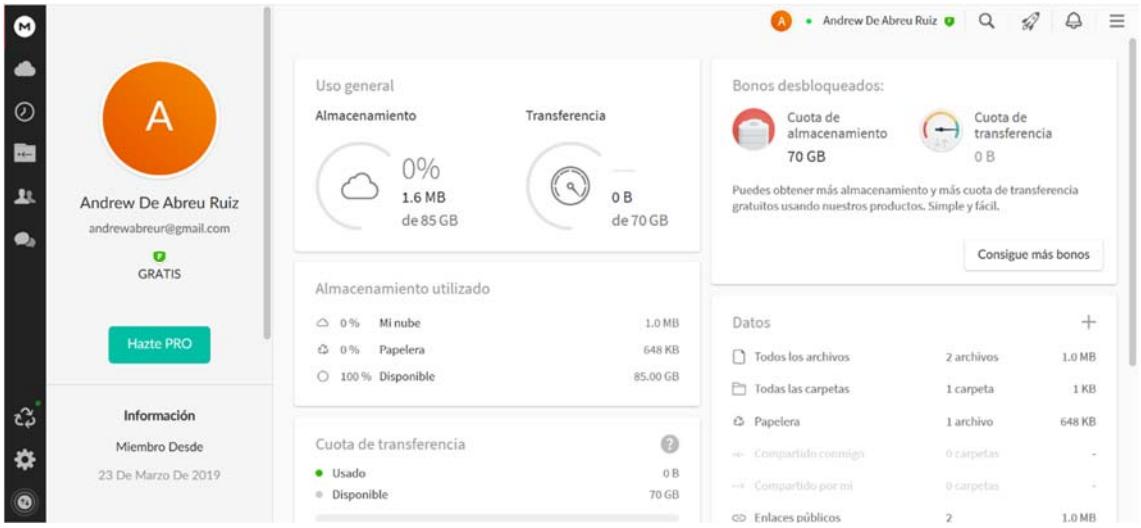

Figura 2-2. Panel de control de la aplicación web de MEGA [Captura de pantalla]

2.1.2. Dropbox

Dropbox	
Desarrollador(es)	Dropbox, Inc.
Autor(es)	Drew Houston y Arash Ferdowsi
Tipo	Servicio de almacenamiento en la nube
Versión	69.4.102
Escrito en	Python, Go, TypeScript, Rust, C++
Licencia	Combina GPLv2 y software propietario
Multiplataforma	Sí
Multilingüe	Sí
Año de lanzamiento inicial	Junio de 2017
Sitio oficial	https://www.dropbox.com/

Tabla 2-2. Ficha técnica de Dropbox [Wikipedia]

En segundo lugar, continuaremos con el análisis de la aplicación web de Dropbox. Esta permite realizar las mismas funcionalidades básicas que la aplicación web de MEGA. No obstante, en vez de permitir añadir archivos o carpetas a favoritos, permite destacar los archivos de manera que aparezcan en la pantalla de inicio para que podamos acceder directamente a ellos.

La compartición de un archivo o carpeta se hace de igual forma que en la aplicación web de MEGA. Sin embargo, los enlaces no se cifran con una clave de cifrado si no que son públicos.

Aparte de las características anteriores, la aplicación web ofrece una característica muy útil que permite solicitar un archivo o carpeta a un usuario incluso si éste no dispone de cuenta. Estos archivos nos aparecerán en el apartado de archivos solicitados de la interfaz.

Una de las principales fortalezas que resaltar de la aplicación web de Dropbox es la simplicidad de su interfaz, así como su facilidad de uso (véase la Figura 2-3 y la Figura 2-4). Ofrece lo mínimo necesario para hacer uso de su servicio sin abrumar al usuario. Desde el panel de control se puede configurar fácilmente la cuenta, consultar nuestro espacio de almacenamiento, etc.

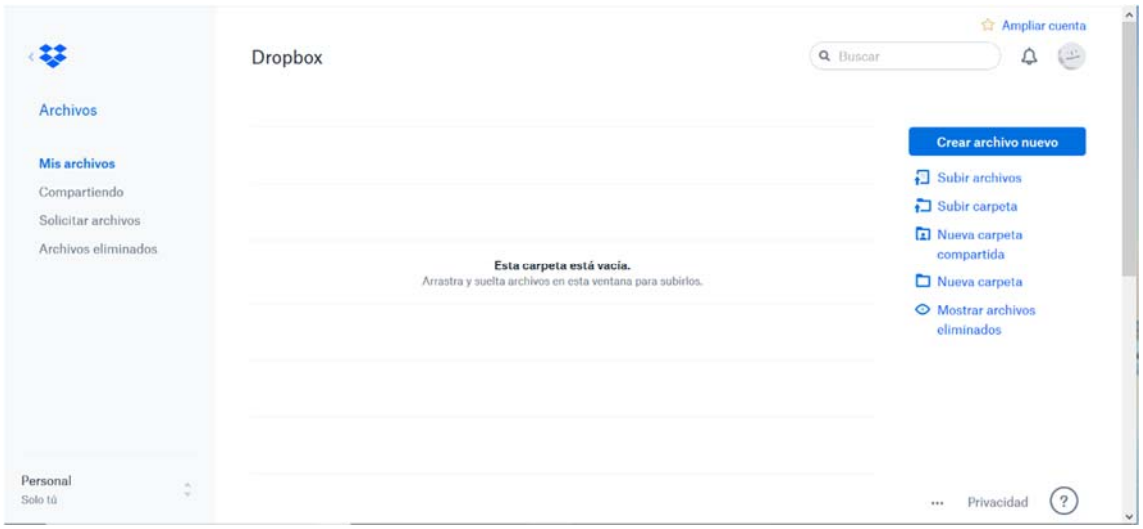

Figura 2-3. Página principal de la aplicación web de Dropbox [Captura de pantalla]

Otra fortaleza que destacar es que, a diferencia de MEGA y pCloud, sí disponemos de herramientas de colaboración. Por ejemplo, se pueden editar archivos y podemos agregarles comentarios que pueden responder los usuarios con los que compartamos los archivos y carpetas.

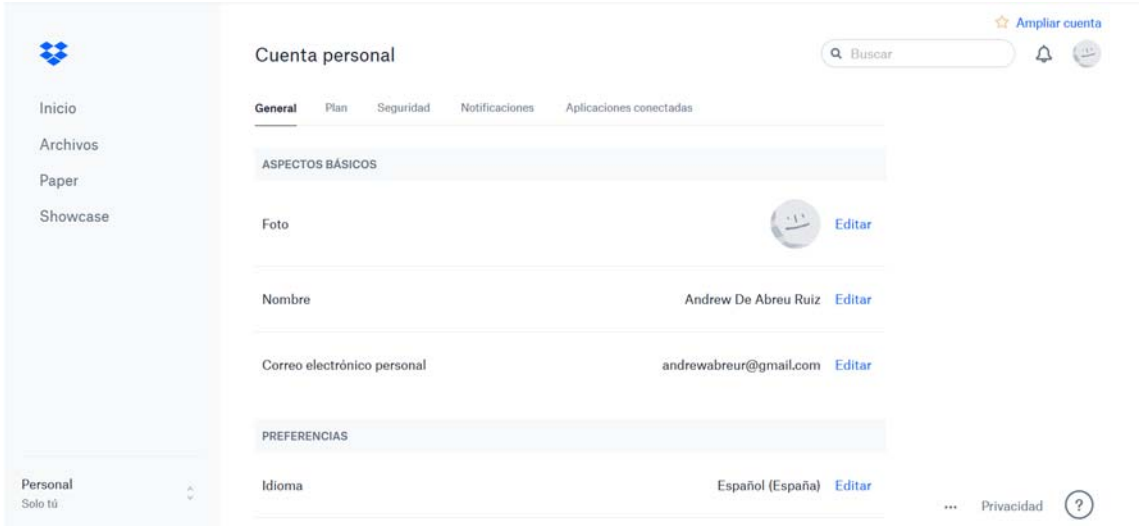

Figura 2-4. Panel de control de la aplicación web de Dropbox [Captura de pantalla]

2.1.3. pCloud

pCloud	
Desarrollador(es)	pCloud Company
Autor(es)	pCloud Team
Tipo	Servicio de almacenamiento en la nube
Versión	3.8.0
Escrito en	No se muestra en ningún sitio.
Licencia	Software propietario
Multiplataforma	Sí
Multilingüe	Sí
Año de lanzamiento inicial	Marzo de 2019
Sitio oficial	https://www.pcloud.com/

Tabla 2-3. Ficha técnica de pCloud [Wikipedia]

En tercer y último lugar se concluye este estudio analizando la aplicación web de pCloud. Tras valorar las aplicaciones web de MEGA y Dropbox, podemos decir que su aplicación ofrece características similares a estas, aunque existen algunas diferencias.

La compartición de archivos o carpetas es similar que en MEGA y Dropbox. Se puede realizar por invitación o por enlace, que puede ser de carga o descarga. Aquí cabe destacar que se permite ver las estadísticas relacionadas con un enlace. Por ejemplo, la fecha de creación, el número total de descargas y el tráfico total. Además, también se permite acortarlos.

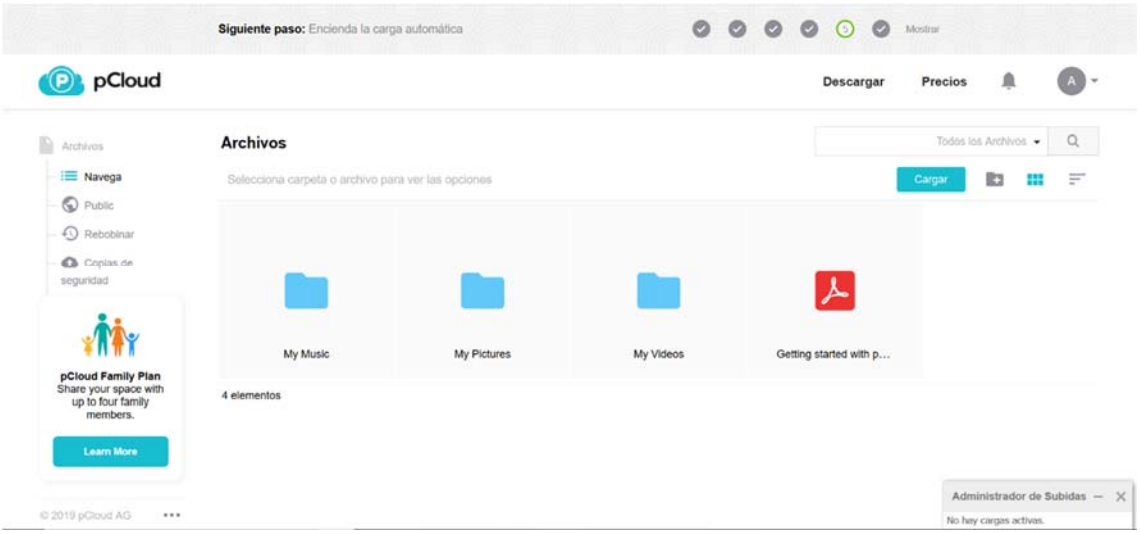

Figura 2-5. Página principal de la aplicación web de pCloud [Captura de pantalla]

A diferencia de las interfaces de usuario de las aplicaciones web de MEGA y Dropbox, la interfaz de usuario de la aplicación de pCloud deja mucho que desear (véase la Figura 2-5 y la Figura 2-6). Es poco atractiva y cuesta mucho encontrar y entender que funcionalidades ofrece teniendo que recurrir, en parte, a la documentación. Este diseño no se tendrá en cuenta para la interfaz de usuario de nuestra aplicación.

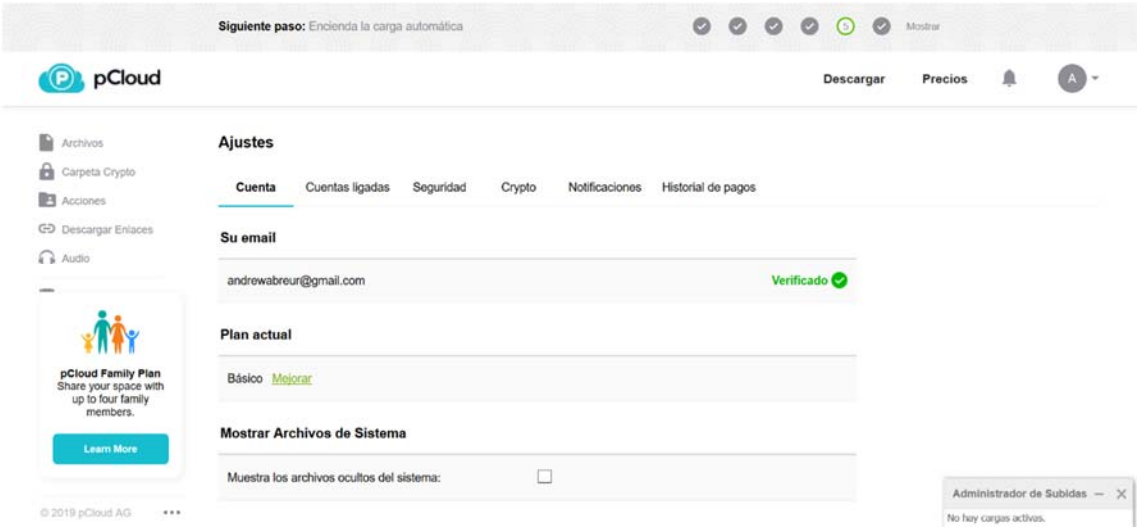

Figura 2-6. Panel de control de la aplicación web de pCloud [Captura de pantalla]

Al igual que en MEGA, también carece de herramientas de colaboración. En cambio, sí permite visualizar archivos y dispone de su propio reproductor de vídeo y música. Por ejemplo, podemos reproducir una película que tengamos almacenada o nuestra lista de reproducción con canciones.

Para terminar, pCloud ofrece dos funcionalidades que no presentan MEGA y pCloud. La primera es que podemos usar una carpeta pública para colocar en ella sitios web con HTML estático o imágenes para incrustar en un sitio web y la segunda es que dispone de un programa de logros con el que se puede ganar más espacio de almacenamiento y dinero si invitamos a otras personas. Este dinero se emplea para pagar la suscripciones al plan Premium.

2.1.4. Comparativa

En la siguiente tabla se hace una comparativa de las tres aplicaciones y se hace un resumen de las características comentadas en las secciones anteriores, además, también se incluyen algunas no mencionadas.

Características	Dropbox	MEGA	pCloud
Subir archivos/carpetas	✔	✔	✔
Descargar archivos/carpetas	✔	✔	✔
Renombrar archivos/carpetas	✔	✔	✔
Mover archivos/carpetas	✔	✔	✔
Copiar archivos/carpetas	✔	✔	✔
Buscar archivos/carpetas	✔	✔	✔
Eliminar archivos/carpetas	✔	✔	✔
Añadir a favoritos	✔	✔	✖
Destacar archivos/carpetas	✔	✖	✖
Etiquetar archivos	✖	✔	✖
Filtrar archivos/carpetas	✖	✔	✖
Ordenar archivos/carpetas	✔	✔	✔
Solicitar archivo/carpeta	✔	✖	✖
Compartir archivos/carpetas	✔	✔	✔
Compartir mediante enlace	✔	✔	✔
Compartir mediante invitación	✔	✔	✔
Permisos de nivel de acceso	✔	✔	✔
Fecha de expiración de enlace	✔ (Professional)	✔ (PRO)	✔ (Premium)
Contraseñas para enlaces	✔ (Professional)	✔ (PRO)	✔ (Premium)

Características	Dropbox	MEGA	pCloud
Acortar enlace	✗	✗	✓
Enlace de carga	✗	✗	✓
Cifrado de enlace	✗	✓	✗
Estadísticas en enlace	✗	✗	✓
Visualizar archivos	✓	✓	✓
Editar archivos	✓	✗	✗
Colaboración	✓	✗	✗
Integración con Office Online	✓	✗	✗
Integración con Google Docs	✗	✗	✗
Reproductor multimedia	✓	✓	✓
Programa de logros	✓	✓	✓
Recompensa por invitar amigos	✓	✓	✓ Con posibilidad de ganar dinero.
Chat	✗	✓	✗
Planes de pago	Plus, Professional	PRO I, PRO II, PRO III	Premium, Premium Plus

Tabla 2-4. Comparativa de las aplicaciones web de MEGA, Dropbox y pCloud

2.2. Requisitos

Al tratarse de un servicio de almacenamiento de archivos en la nube, el prototipo de aplicación web está orientado a cualquier tipo de usuario. En nuestro caso, sólo distinguimos entre los usuarios cliente, los usuarios administradores y los usuarios anónimos. La principal motivación de un usuario cliente es la de almacenar sus archivos de forma segura mientras que la de un usuario administrador es la de administrar el uso que se hace de dicho servicio. Un usuario anónimo no podrá hacer mucho, sólo registrarse o acceder a las páginas estáticas de la aplicación. Por ejemplo, la página de inicio, la página de registro, etc.

A partir del estudio de aplicaciones similares realizado anteriormente hemos obtenido una serie de requisitos que debe cumplir el prototipo para así satisfacer las necesidades de sus usuarios. Los requisitos pueden ser clasificados en dos grupos según su audiencia: requisitos del usuario

y requisitos del sistema [5]. A su vez, los requisitos del sistema se clasifican en dos grupos: requisitos funcionales y requisitos no funcionales. Dado que no disponemos de un cliente con el cual entrevistarnos y validar los requisitos, nos centraremos en los requisitos del sistema. En concreto, en los requisitos funcionales y no funcionales.

2.2.1. Requisitos funcionales

Los requisitos funcionales son las capacidades o servicios que deberá proporcionar el sistema para alcanzar los objetivos del usuario, cómo deberá reaccionar a las entradas que se produzcan y qué comportamiento deberá tener ante situaciones particulares.

A partir de esta definición, en la Tabla 2-5 se incluyen los siguientes requisitos funcionales:

ID	Descripción
F001	El sistema debe permitir a un usuario anónimo registrarse.
F002	El sistema debe permitir a un usuario registrado iniciar sesión.
F003	El sistema debe permitir a un usuario registrado reestablecer su contraseña.
F004	El sistema deberá permitir a un usuario registrado subir archivos a su cuenta de almacenamiento.
F005	El sistema deberá permitir a un usuario registrado descargar archivos de su cuenta de almacenamiento.
F006	El sistema deberá permitir a un usuario registrado subir carpetas a su cuenta de almacenamiento.
F007	El sistema deberá permitir a un usuario registrado compartir archivos de su cuenta de almacenamiento mediante una invitación por correo electrónico con otros usuarios.
F008	El sistema deberá permitir a un usuario registrado compartir archivos de su cuenta de almacenamiento mediante un enlace.
F009	El sistema deberá permitir a un usuario registrado compartir carpetas de su cuenta de almacenamiento mediante una invitación por correo electrónico.
F010	El sistema deberá permitir a un usuario registrado compartir carpetas de su cuenta de almacenamiento mediante un enlace.
F011	El sistema deberá permitir a un usuario registrado descargar carpetas de su cuenta de almacenamiento.
F012	El sistema deberá permitir a un usuario registrado buscar archivos o carpetas de su cuenta de almacenamiento.
F013	El sistema deberá permitir a un usuario registrado cancelar su cuenta.

ID	Descripción
F014	El sistema deberá permitir a un usuario registrado con una suscripción a un plan de almacenamiento gratuito adquirir una subscripción a un plan de almacenamiento de pago.
F015	El sistema deberá permitir a un usuario registrado con una suscripción a un plan de almacenamiento de pago cancelarla.
F016	El sistema deberá permitir a un usuario registrado realizar el pago de una suscripción a un plan de almacenamiento.
F017	El sistema deberá permitir a un usuario registrado renovar manualmente la suscripción a un plan de almacenamiento de pago manualmente.
F018	El sistema deberá permitir a un usuario registrado consultar a que plan de almacenamiento está suscrito.
F019	El sistema deberá permitir a un usuario registrado consultar el historial de pagos de sus suscripciones a planes de almacenamiento de pago.
F020	El sistema deberá permitir a un usuario registrado consultar las estadísticas sobre el uso de su cuenta de almacenamiento.
F021	El sistema deberá permitir a un usuario registrado consultar la actividad reciente de su cuenta de almacenamiento.
F022	El sistema deberá permitir a un usuario registrado consultar los detalles de una carpeta de su cuenta de almacenamiento.
F023	El sistema deberá permitir a un usuario registrado consultar los detalles de un archivo de su cuenta de almacenamiento.
F024	El sistema deberá permitir a un usuario registrado consultar el contenido de una carpeta de su cuenta de almacenamiento.
F025	El sistema deberá permitir a un usuario registrado consultar su perfil.
F026	El sistema deberá permitir a un usuario registrado editar su perfil de usuario.
F027	El sistema deberá permitir a un usuario registrado cambiar su contraseña.
F028	El sistema deberá permitir a un usuario registrado consultar la factura asociada al pago de una suscripción a un plan de almacenamiento.
F029	El sistema deberá permitir a un usuario registrado descargar la factura asociada al pago de una suscripción a un plan de almacenamiento en formato PDF.
F030	El sistema deberá permitir a un usuario registrado eliminar un archivo de su cuenta de almacenamiento.
F031	El sistema deberá permitir a un usuario registrado eliminar carpetas de su cuenta de almacenamiento.
F032	El sistema deberá permitir a un usuario registrado renombrar un archivo de su cuenta de almacenamiento.
F033	El sistema deberá permitir a un usuario registrado renombrar una carpeta de su cuenta de almacenamiento.
F034	El sistema deberá permitir a un usuario registrado renombrar una carpeta de su cuenta de almacenamiento.

ID	Descripción
F035	El sistema deberá permitir a un usuario registrado copiar un archivo de su cuenta de almacenamiento.
F036	El sistema deberá permitir a un usuario registrado mover un archivo de su cuenta de almacenamiento a otra ubicación.
F037	El sistema deberá permitir a un usuario registrado copiar una carpeta de su cuenta de almacenamiento.
F038	El sistema deberá permitir a un usuario registrado dejar de compartir un archivo de su cuenta de almacenamiento.
F039	El sistema deberá permitir a un usuario registrado dejar de compartir una carpeta de su cuenta de almacenamiento.
F040	El sistema deberá permitir a un usuario registrado consultar los archivos compartidos de su cuenta de almacenamiento.
F041	El sistema deberá permitir a un usuario registrado consultar las carpetas compartidas de su cuenta de almacenamiento.
F042	El sistema deberá permitir a un usuario registrado crear una carpeta en un directorio de su cuenta de almacenamiento.
F043	El sistema deberá permitir a un usuario administrador consultar un listado con todos los usuarios registrados en el sistema.
F044	El sistema deberá permitir a un usuario administrador consultar el perfil de un usuario registrado.
F045	El sistema deberá permitir a un usuario administrador consultar las estadísticas de uso de un usuario registrado.
F046	El sistema deberá permitir a un usuario administrador consultar las estadísticas de uso de todo el servicio de almacenamiento.
F047	El sistema deberá permitir a un usuario administrador consultar los ingresos recibidos de las suscripciones de los usuarios registrados a los planes de almacenamiento de pago ofrecidos por el servicio de almacenamiento.
F048	El sistema deberá permitir a un usuario administrador eliminar a un usuario registrado.
F049	El sistema deberá permitir a un usuario administrador editar a un usuario registrado.

Tabla 2-5. Requisitos funcionales

2.2.2. Requisitos no funcionales

Los requisitos no funcionales son las cualidades de ejecución y evolución que debe tener el sistema, restricciones que afectan a los servicios que debe ofrecer y condiciones que ponen límites, tanto al sistema como al proceso de desarrollo.

A partir de esta definición, en la Tabla 2-6 se incluyen los siguientes requisitos no funcionales:

ID	Descripción
NF001	El tiempo de aprendizaje del sistema por un usuario deberá ser menor a 2 horas.
NF002	El sistema debe contar con un manual de usuario que detalle cómo utilizarlo.
NF003	El sistema debe proporcionar mensajes de error que indiquen el problema ocurrido adecuadamente y que estén orientados al usuario final.
NF004	La interfaz de usuario será implementada para navegadores web con *HTML5*, *CSS3* y *JavaScript*.
NF005	La interfaz de usuario deberá soportar el idioma español y el idioma inglés.
NF006	El sistema no debe consumir menos de 512 MB de Memoria RAM. (Para que pueda ser desplegado en *Heroku*)
NF007	El lenguaje de programación a utilizar será *Ruby*.
NF008	El *framework* para el desarrollo del *backend* a utilizar será *Ruby on Rails*.
NF009	La metodología de desarrollo de software debe ser *Behavior Driven Development* (BDD) empleando Respes.
NF010	El entorno de desarrollo deberá estar instalado y configurado en un sistema operativo *Ubuntu*.
NF011	La base de datos en el entorno de producción será *PostgreSQL*.
NF012	La base de datos en el entorno de desarrollo será *SQLite3*.

Tabla 2-6. Requisitos no funcionales

A pesar de que los requisitos no funcionales expuestos en la tabla anterior son prescindibles por ser algunos demasiados obvios hemos decidido incluirlos para así disponer de una serie de ellos. Se han incluido requisitos de desarrollo, de eficiencia y de usabilidad. Sin embargo, no se han incluido los requisitos legislativos puesto que se comentan detalladamente en la sección de Normativa y legislación de este mismo apartado.

2.3. Diagramas de casos de uso

Para capturar los requisitos funcionales definidos [6] en la sección anterior y especificar la comunicación y el comportamiento de la aplicación mediante su interacción con los usuarios y/u otros sistemas emplearemos los diagramas de casos de usos [7].

En estos diagramas se contemplan cuatros actores: el usuario anónimo, el usuario cliente/registrado, el usuario administrador y la pasarela de pago.

En la Figura 2-7 se muestra el diagrama de los casos de uso relacionados con la gestión de las suscripciones en la aplicación. En este diagrama se indica que el usuario con el rol de usuario registrado puede efectuar el caso de uso de *Adquirir suscripción* que incluye efectuar el caso de uso de *Realizar pago*. Por otro lado, también puede efectuar el caso de uso de *Cambiar suscripción* y *Cancelar suscripción* siempre y cuando se den las precondiciones necesarias. También podemos ver como el rol de la pasarela de pago está involucrado en estos casos de uso el cual es necesario para poder efectuarlos correctamente.

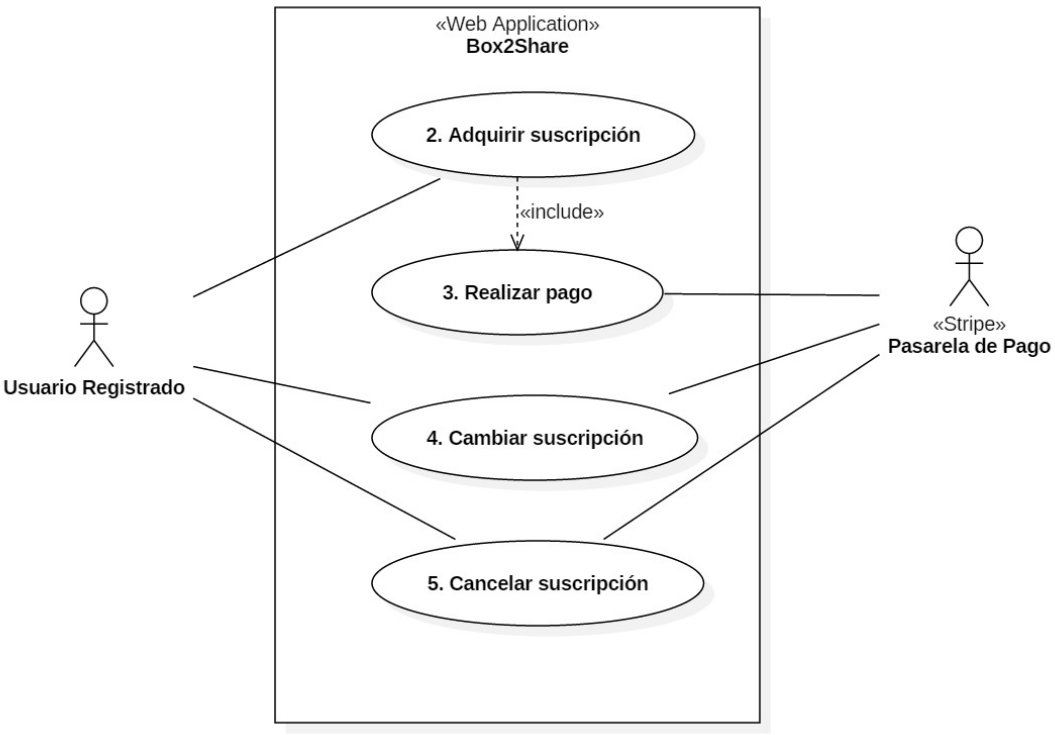

Figura 2-7. Diagrama de casos de uso para la gestión de las suscripciones

En la sección de Diagramas de casos de uso del apartado de Anexos se incluyen el resto de los diagramas de casos de uso. Hay que destacar que se han diseñado así los diagramas para evitar que sean muy profundos horizontalmente. Es verdad que tal y como se implementará la aplicación se podrían haber empleado relaciones <<include>> y <<extends>> entre los actores y casos de uso. Sin embargo, esto implicaría una mayor redundancia y que los diagramas de casos de uso fueran muy extensos. En las precondiciones de la especificación de los casos de uso se establece cuando se pueden efectuar los casos de uso.

En la Tabla 2-7 se resumen todos los casos de uso definidos en la aplicación. En ella se indica el identificador del caso de uso, su nombre y su estado. Es decir, si se ha implementado o no. La ventaja de incluir este identificador es que nos permite relacionarlos mejor con el diagrama.

ID	Nombre	Estado
CU01	Registrarse	Implementado
CU02	Adquirir suscripción	Implementado
CU03	Realizar pago	Implementado
CU04	Cambiar suscripción	Implementado
CU05	Cancelar suscripción	Implementado
CU06	Iniciar sesión	Implementado
CU07	Cambiar contraseña	Implementado
CU08	Confirmar cuenta	Implementado
CU09	Cancelar cuenta	Implementado
CU10	Modificar perfil	Implementado
CU11	Recuperar contraseña	Implementado
CU12	Buscar archivos o carpetas	No implementado
CU13	Consultar estadísticas de uso	Implementado
CU14	Consultar factura	Implementado
CU15	Consultar historial de pagos	Implementado
CU16	Subir archivo	Implementado
CU17	Bajar archivo	Implementado
CU18	Compartir archivo	Implementado
CU19	Dejar de compartir archivo	Implementado
CU20	Renombrar archivo	Implementado
CU21	Copiar archivo	No implementado
CU22	Eliminar archivo	Implementado
CU23	Mover archivo	No implementado
CU24	Ver detalles de archivo	Implementado
CU25	Visualizar archivo	No implementado
CU26	Consultar archivo compartidos	Implementado
CU27	Generar enlace a archivo	No implementado
CU28	Eliminar enlace a archivo	No implementado
CU29	Consultar carpeta	Implementado
CU30	Crear carpeta	Implementado
CU31	Copiar carpeta	No implementado
CU32	Eliminar carpeta	Implementado
CU33	Descargar carpeta	No implementado
CU34	Subir carpeta	No implementado
CU35	Ver detalles de carpeta	Implementado
CU36	Mover carpeta	No implementado
CU37	Renombrar carpeta	Implementado
CU38	Dejar de compartir carpeta	No implementado
CU39	Generar enlace a carpeta	No implementado
CU40	Consultar carpetas compartidas	No implementado

ID	Nombre	Estado
CU41	Compartir carpeta	No implementado
CU42	Eliminar enlace a carpeta	No implementado
CU43	Eliminar usuario	Implementado
CU44	Modificar usuario	No implementado
CU45	Consultar usuario	Implementado
CU46	Consultar estadísticas de uso del usuario	Implementado
CU47	Consultar estadísticas de uso del servicio	No implementado
CU48	Consultar historial de pagos del servicio	Implementado
CU49	Añadir método de pago	No implementado
CU50	Cambiar método de pago	Implementado
CU51	Eliminar método de pago	No implementado

Tabla 2-7. Casos de uso de la aplicación

2.4. Especificación de casos de uso

En esta sección se muestra la especificación de dos de los casos de usos más relevantes de la aplicación. El resto se incluye en la sección de Especificación de casos de uso del apartado de Anexos.

CASO DE USO	**CU02**	**Adquirir suscripción**
Descripción		El sistema permite a un usuario registrado con una suscripción a un plan de almacenamiento gratuito adquirir una suscripción a un plan de almacenamiento de pago.
Actores		Usuario Registrado
Fecha de creación		04/04/2019
Fecha de modificación		12/06/2019
Precondiciones		• El usuario registrado debe haber iniciado sesión en el sistema. • El usuario registrado debe estar suscrito a un plan de almacenamiento gratuito. • El usuario registrado debe encontrarse en la vista con la tabla de precios.
Flujo normal		**Paso** | **Acción**

	1	El usuario registrado selecciona la opción de suscribirse a un plan de almacenamiento de pago.
	2	El usuario registrado realiza el pago de la suscripción (CU03: Realizar pago).
	3	El sistema redirige al usuario registrado a la vista del directorio raíz y le muestra un mensaje de confirmación de la suscripción.
Postcondiciones	• Se actualiza el plan de almacenamiento del usuario registrado cuando se obtenga la respuesta de la pasarela de pago. • Se amplía el espacio de almacenamiento disponible para el usuario registrado. • Se añade un pago al historial de pagos del usuario registrado. • Se envía un correo electrónico con el comprobante y los datos de la suscripción.	
Variaciones	**Paso**	**Acción**
Extensiones	**Paso**	**Condición** / **Caso de Uso**
Excepciones		
Observaciones		

Tabla 2-8. Especificación del caso de uso de Adquirir suscripción

CASO DE USO	**CU18**	**Compartir archivo**
Descripción		El sistema permite a un usuario registrado compartir un archivo de su cuenta de almacenamiento.
Actores		Usuario Registrado
Fecha de creación		04/05/2019
Fecha de modificación		23/06/2019
Precondiciones		• El usuario registrado debe haber iniciado sesión en el sistema. • El usuario registrado debe tener al menos un archivo en su cuenta de almacenamiento.

	• El usuario registrado debe encontrarse en la vista del directorio raíz o en la vista de la carpeta.		
Flujo normal	**Paso**	**Acción**	
	1	El usuario registrado selecciona la opción de compartir un archivo.	
	2	El sistema muestra al usuario una vista con un formulario con los campos a rellenar con la dirección de correo electrónico del usuario con el que se compartirá el archivo y el mensaje (opcional).	
	3	El usuario registrado rellena el formulario.	
	4	El sistema muestra un mensaje al usuario indicándole al usuario que el archivo ha sido compartido con éxito.	
Postcondiciones	• El sistema envía al otro usuario un correo electrónico con el enlace al archivo. • El archivo aparece en el listado de archivos compartidos.		
Variaciones	**Paso**	**Acción**	
Extensiones	**Paso**	**Condición**	**Caso de Uso**
Excepciones			
Observaciones			

Tabla 2-9. Especificación del caso de uso de Compartir archivo

2.5. Mockups, diseño de la interfaz y paleta de colores

Tras estudiar algunas aplicaciones existentes en el mercado, definir los requisitos y elaborar los diagramas de casos de uso realizamos un *mockup* para representar el diseño de la interfaz de usuario de la aplicación. En él se incluyen las vistas más representativas dejando a un lado aquellas que sean similares a las presentes. Todas las vistas se muestran en la sección de Mockups del apartado de Anexos.

El *mockup* realizado se emplea como guía para el diseño de la interfaz de usuario del prototipo. En este diseño hemos procurado combinar las mejores ideas obtenidas del estudio de las interfaces web de las aplicaciones existentes en el mercado. Por ejemplo:

- El menú lateral desde el cual se pueden acceder a las funcionalidades que permite la aplicación.

- La vista del plan actual donde el usuario puede comprobar toda la información acerca del plan al que está suscrito.

- La posición de los botones para las funcionalidades de subir archivo y crear una nueva carpeta.

- El menú de opciones para un archivo o carpeta que mantiene oculto todo el conjunto y evita mostrarlo todo en la vista.

En cuanto a la paleta de colores, emplearemos los colores que se reflejan en la Tabla 2-10. Cabe mencionar que los colores de la cabecera y del menú lateral son personalizados mientras que el resto son los colores predeterminados de Bootstrap 4. La elección de esta paleta se debe a que la mayoría de los servicios de almacenamiento en la nube la emplean. Además, la combinación de los colores gris, blanco y azul hace que la interfaz resulte atractiva y no moleste a la vista de los usuarios cuando interactúan con ella.

Color	Código (HEX)	Uso
	#0275d8	Botones, iconos, …
	#5cb85c	Mensajes de éxito
	#d9534f	Mensajes de error
	#292b2c	Texto, títulos, …
	#f7f7f7	Pie de página
	#1565C0	Cabecera
	#FFFFFF	Color de fondo
	#FBFBFC	Menú lateral

Tabla 2-10. Paleta de colores de la interfaz

2.6. Modelo de negocio

Cuando diseñamos un modelo de negocio se debe pensar que tipo de negocio estamos tratando de crear e introducir al mercado, a quién irá dirigido, cómo se venderá y como se conseguirán ingresos para hacerlo rentable [8]. En nuestro caso estamos tratando de simular un servicio de almacenamiento en la nube que está dirigido a cualquier persona que desee almacenar y compartir sus archivos de forma segura. Esta persona podrá hacer uso de este servicio mediante una aplicación web cuyo prototipo se desarrollará en este trabajo.

El método más utilizado en este sector para percibir ingresos es mediante un modelo de negocio basado en suscripciones. La mayoría de las empresas como MEGA, Dropbox o pCloud ofrecen un plan gratuito con el que los usuarios pueden hacer uso de una serie de características esenciales. Este conjunto de características puede ser ampliado mediante la compra de una suscripción a un plan de pago que se puede pagar mensual o anualmente.

Para crear nuestro propio esquema de suscripciones hemos analizado los planes que ofrecen cada una de estas empresas:

MEGA pone a nuestra disposición cuatro planes de pago llamados LITE, PRO I, PRO II y PRO III que difieren en el precio, en el espacio de almacenamiento disponible y en la cuota de transferencia. El pago de las suscripciones puede hacerse por tarjeta de crédito, cupones, criptomonedas o transferencia bancaria. Así mismo, podemos elegir si se renuevan automática o manualmente.

De manera similar, Dropbox oferta el plan Plus y el plan Professional. Cada uno aporta una serie de nuevas características y más espacio de almacenamiento, 1TB y 2TB respectivamente. El pago de una suscripción a estos planes puede hacerse por tarjeta de crédito, por PayPal o por débito directo.

Por otro lado, pCloud ofrece dos planes de pago llamados Premium y Premium Plus manteniendo la misma filosofía que los de MEGA y Dropbox. No obstante, se diferencian en que el ciclo de facturación es anual o de por vida. Es decir, que el pago se realiza una única vez. Aquí, el pago de una suscripción puede hacerse por tarjeta de crédito, por PayPal, por criptomonedas o por pagos locales en tiendas.

Basándonos en lo anterior, el esquema de suscripciones que planteamos es el siguiente:

En la página de precios se ofrecerán tres planes llamados *Básico*, *Estándar* y *Profesional*. El primero será gratuito y proporcionará un *gigabyte* de espacio de almacenamiento en la cuenta de usuario mientras que el segundo y el tercero serán de pago proporcionando dos y tres

gigabytes de espacio de almacenamiento respectivamente. El ciclo de facturación podrá ser mensual o anual. En cuanto al precio, el coste mensual del plan *Estándar* será de 6 euros al mes (o 72 euros al año) mientras que para el plan *Profesional* será de 12 euros al mes (o 144 euros al año).

Aunque en un principio solo se ofrece la característica de aumentar el espacio de almacenamiento disponible, a medida que la aplicación crezca y se perciban ingresos, se pueden contemplar más funciones a ofrecer, como por ejemplo, la eliminación de la publicidad en la aplicación, asignar fecha de caducidad a los enlaces, etc.

Para finalizar, cabe destacar que gracias a la simplicidad de este esquema de suscripciones, este se podrá implementar en el prototipo mediante la integración del modelo de suscripciones que soporta el servicio de procesamiento de pagos online llamado Stripe.

2.7. Normativa y legislación

A la hora de desarrollar una aplicación es fundamental considerar los aspectos legales a fin de asegurar de que funcione dentro de la ley. Si los ignoramos, podemos tener que hacer frente a sanciones, problemas legales, o incluso a la pérdida de nuestra reputación y la confianza de los usuarios [9].

Dado el planteamiento actual, en la aplicación se realiza el tratamiento de datos de carácter personal de los usuarios, en concreto, el nombre, los apellidos, el correo electrónico y los datos de su tarjeta de crédito. En una primera versión comercial, las normativas más importantes que afectarían a la aplicación son:

- La *Ley Orgánica 3/2018 de Protección de Datos Personales y garantía de los derechos digitales (LOPD-GDD).* Esta ley orgánica tiene como objeto adaptar el Derecho interno español al Reglamento General de Protección de Datos. Esta ley deroga a la anterior *Ley Orgánica 15/1999 de Protección de Datos de Carácter Personal* [10].

- La *Ley de Servicios de la Sociedad de la Información y el Comercio Electrónico (LSSI-CE).* Esta ley fue creada para regular el comercio electrónico en Internet. Su objetivo es el de inspirar más confianza a las personas que utilizan Internet para realizar sus transacciones cotidianas. Contempla todas las obligaciones de los prestadores de servicios establecidos en España y define una serie de garantías de seguridad a los consumidores y a los usuarios. Se debe aplicar siempre que se genere algún tipo de ingresos tal y como ocurre en la aplicación [11].

- El *Reglamento General de Protección de Datos (RGPD).* Este reglamento europeo regula la protección de datos de los ciudadanos que vivan en la Unión Europea. Permite proteger los datos personales de cada usuario en lo que respecta a su tratamiento y libre circulación de estos [11] [12].

- El *Estándar de Seguridad de Datos para la Industria de Tarjeta de Pago (PCI-DSS).* Esta normativa de seguridad se aplica a cualquier organización que almacene, procese o transmita datos de titulares de tarjeta de crédito. Su objetivo es establecer un nivel básico de protección para mantener un ambiente seguro a la hora de realizar nuestros pagos [13].

Para el cumplimiento de la LOPD-GDD, RGPD y la LSSI-CE se debe aplicar una serie de medidas, tanto a nivel organizativo como a nivel de seguridad, en función del alcance del tratamiento que se haga de los datos del usuario y el riesgo al que estén expuestos dentro de la organización. Por otro lado, también se deben aplicar medidas a nivel de aplicación. Algunos ejemplos de ellas son [11] [14]:

- Informar sobre la privacidad de los datos recogidos en los formularios y en la página dónde se muestre la política de privacidad.

- Recibir el consentimiento explícito de los usuarios antes de recoger sus datos.

- Notificar sobre el uso de cookies. También informar de la política de cookies en una página aparte.

- Informar sobre los datos de la empresa, precio de los productos o servicios, impuestos que se aplican, etc., en una página de aviso legal.

- Informar de las condiciones de uso del servicio en otra página distinta.

Para el cumplimiento de la PCI-DSS se debe aplicar el siguiente procedimiento dentro de la organización [13]:

1. Manejar la recepción de los datos de tarjetas de crédito de los usuarios, es decir, reunir y transmitir los datos sensibles de las tarjetas de manera segura.

2. Guardar los datos de manera segura, según se describe en los 12 dominios de seguridad que propone la normativa.

3. Validar anualmente que funcionen los controles de seguridad necesarios, lo que puede implicar formularios, cuestionarios, servicios externos de escaneo de vulnerabilidades y auditorías de terceros.

En nuestro caso se integra a Stripe como servicio de procesamiento de pagos en línea para implementar el modelo de negocio basado en suscripciones de modo que los usuarios puedan pagar por disponer de más características en la aplicación. Al utilizar la librería Elements[1] que nos ofrece, Stripe asume parte de la carga del cumplimiento de esta normativa. Esta librería emplea un campo de pago alojado para manejar todos los datos de las tarjetas, de modo que el titular de esta introduce todos los datos sensibles de pago en un campo que se origina directamente en los servidores de Stripe que están validados conforme a la normativa. Esto evita las complicaciones, los costes y los riesgos que se puedan dar ya que los datos de las tarjetas nunca entran en contacto con nuestro servidor. No obstante, no exime del cumplimiento de algunos controles de seguridad exigidos.

Para concluir, cabe mencionar que el proceso de adaptación de una aplicación y de la entidad que la desarrolla a la legislación es continuo. A medida que crece, se debe contemplar si las nuevas actividades que realiza obligan a aplicar más medidas de las que se llevaban cumpliendo hasta ese momento. A pesar de que en este trabajo nuestro objetivo no es implementar ninguna medida de las nombradas anteriormente, salvo las esenciales como la autenticación y el control de acceso, en la implementación del prototipo se tendrá en cuenta que los datos de tarjetas de pago no entren en contacto con nuestro servidor tal y como se comentó en los párrafos anteriores.

3. Diseño

3.1. Diseño arquitectónico

3.1.1. Modelo Vista Controlador (MVC)

Puesto que en este trabajo se desarrolla un prototipo de aplicación web con el *framework* Ruby on Rails, el patrón de arquitectura de software a emplear es el patrón Modelo Vista Controlador

[1] Stripe Elements es un complemento que se integra con la librería JavaScript, Stripe.js, que permite recolectar los datos de pago de los usuarios mediante elementos de interfaz de usuario predefinidos y personalizables en el *front-end* de la aplicación.

(MVC). Este patrón separa los datos de una aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos, cada uno con sus responsabilidades bien definidas [15]:

- **Modelo.** Es el componente que contiene la representación de la información que maneja el sistema, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación, es decir, la lógica de negocio. Envía a la vista aquella parte de la información que en cada momento se le solicita para que sea mostrada al usuario. Las peticiones de acceso o manipulación de información llegan al modelo a través del controlador.

- **Controlador.** Es el componente que responde a eventos (normalmente acciones del usuario) e invoca peticiones al modelo cuando se hace alguna solicitud sobre la información. También puede enviar comandos a su vista asociada si se solicita un cambio en la forma en que se presenta el modelo, por tanto se podría decir que el controlador hace de intermediario entre la vista y el modelo.

- **Vista.** Es el componente que presenta el modelo en un formato adecuado para interactuar (normalmente la interfaz de usuario), por tanto requiere de la información que gestiona el modelo que debe representar como salida.

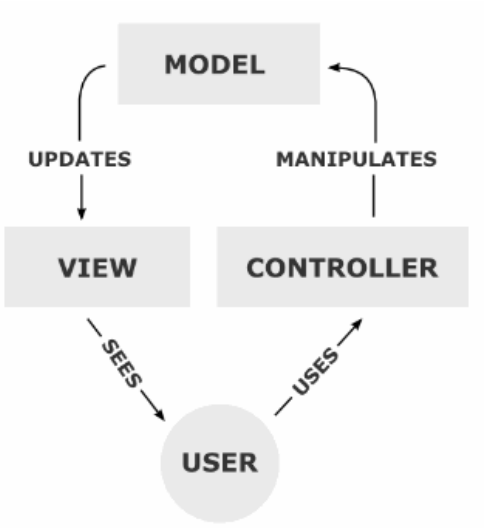

Figura 3-1. Interacción típica entre los componentes de un MVC [Wikipedia]

El flujo de control que se sigue generalmente en este patrón es el siguiente (véase Figura 3-1):

1. El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, pulsando un botón, enlace, etcétera).

2. El controlador recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega a través de un gestor de eventos.

3. El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario.

4. El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se reflejan los cambios en el modelo.

5. La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente.

Algunas de las principales ventajas que proporciona este patrón son [16]:

- Separación clara de dónde tiene que ir cada tipo de lógica, facilitando el mantenimiento y la escalabilidad de la aplicación.

- Sencillez para crear distintas representaciones de los mismos datos.

- Facilidad para la realización de pruebas unitarias de los componentes, así como de aplicar un desarrollo guiado por pruebas (*Test Driven Development* o TDD).

- Reutilización de los componentes.

- Facilidad para desarrollar y mantener aplicaciones.

3.1.2. MVC en Ruby on Rails

El *framework* Ruby on Rails implementa este patrón mediante los componentes Active Record, Action Controller y Action View que están incluidos en su núcleo.

Active Record (M)

Es la capa del sistema responsable de representar los datos y la lógica de negocio. Facilita la creación y el uso de los objetos del negocio cuyos datos requieren un almacenamiento persistente en la base de datos. Es una implementación del patrón de diseño Active Record que a su vez es una descripción de un sistema de Mapeo Relacional de Objetos (ORM) [17].

Action View (V)

Es la capa del sistema responsable de compilar y proporcionar la respuesta empleando los datos recibidos por parte del controlador. Las vistas son plantillas de Action View que se escriben usando Ruby incrustado (ERB) en etiquetas mezcladas con HTML. Para evitar abarrotar las plantillas con el código duplicado, Action View proporciona una serie de métodos ayudantes para varios propósitos [18].

La respuesta que se envía al usuario es la combinación entre la vista, el layout (contenido común a todas las vistas) y los *partials* (porciones de código HTML+ERB extraídas de una vista normalmente para reducir su tamaño o evitar código duplicado) además de los datos obtenidos del modelo mediante el controlador.

Action Controller (C)

Es la capa del sistema responsable de recibir la solicitud (una vez procesada por el enrutador), obtener o guardar datos de un modelo y usar una vista para crear un resultado HTML [19].

Un controlador es una clase Ruby que hereda de la clase *ApplicationController* y dispone de métodos como cualquier otra clase. Cuando la aplicación recibe una solicitud, el enrutador determinará que controlador y acción ejecutar, luego Rails crea una instancia del controlador y ejecuta el método con el mismo nombre que la acción.

Ejemplo de interacción básico

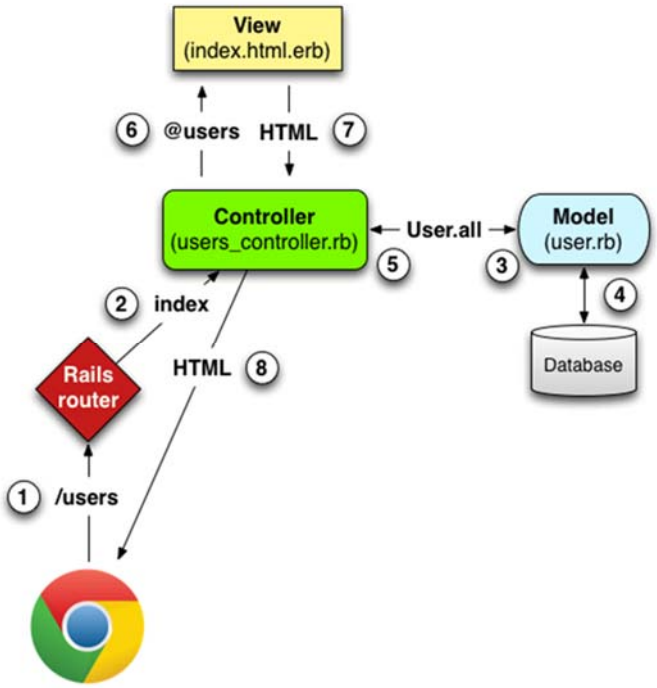

Figura 3-2. Ejemplo de interacción básico de MVC en Rails [Ruby on Rails Tutorial]

Suponiendo que contamos con un modelo *User* tratado como un recurso RESTful (se generan todas las rutas convencionales para tratar con él), un controlador *UsersController* que cuenta con las acciones correspondientes a cada ruta (*index*, *show*, *new*, *edit*, *create*, *update*, *destroy*) y las vistas asociadas a cada acción (para ciertas acciones), el flujo de interacciones para consultar todos los usuarios es el siguiente (véase Figura 3-2) [20]:

1. El navegador emite una solicitud para la URL /users.
2. Ruby on Rails enruta la solicitud a la acción *index* en el controlador *UsersController*.
3. La acción *index* pregunta al modelo *User* para recuperar todos los usuarios (User.all).
4. El modelo *User* recupera todos los usuarios de la base de datos.
5. El modelo *User* devuelve la lista de usuarios al controlador.
6. El controlador captura los usuarios en la variable de instancia *@users*, la cual se pasa a la vista *index.html.erb*.
7. La vista usa Ruby incrustado (ERB) para renderizar la página como HTML.

8. El controlador pasa el HTML de vuelta al navegador.

Para que todo el flujo de interacciones anterior pueda funcionar correctamente también se requieren de otros componentes o elementos como el servidor de Rails (*rails server*), el enrutador (*routes*) y los bienes (*assets*).

3.2. Estructura de la aplicación

En la Tabla 3-1 se comentan los elementos más importantes de la estructura del directorio de una aplicación Ruby on Rails [21]:

Archivo/Carpeta	Propósito
app/	Este directorio contiene los subdirectorios en donde se ubican los *controllers*, *models*, *views*, *helpers*, *mailers* y *assets* de la aplicación. Es dónde se realiza la mayor parte del trabajo de la fase de desarrollo de este proyecto.
bin/	Este directorio contiene todos los comandos de Rails necesarios para trabajar con la aplicación. Los más importantes son el comando *bundle*, *rails* y *rake*.
config/	Este directorio contiene los archivos para configurar el enrutador de Rails (*routes.rb*), la base de datos (*database.yml*), etc. También contiene los subdirectorios con los archivos para configurar los entornos de Rails, los inicializadores (código que se ejecuta al iniciar la aplicación para establecer la configuración de algunas gemas de terceros), etc.
db/	Este directorio contiene el archivo con el esquema actual de la base de datos, el archivo para iniciar la aplicación con la base de datos poblada con valores predeterminados (*seed.rb*) y un subdirectorio en donde se ubican las migraciones que alteran el esquema de la base de datos.
lib/	Este directorio contiene los módulos para extender la aplicación. Aquí suelen residir normalmente las tareas de Rails (*tasks*) que creemos.
log/	Este directorio contiene los archivos de log de la aplicación para los entornos de desarrollo, producción o test.
public/	Este directorio contiene todo aquello que pueda ser accesible por todos, como por ejemplo, las páginas de error, los archivos estáticos, etc.
Rakefile	Este archivo localiza y carga las tareas que se pueden ejecutar desde la línea de comandos. Las definiciones de tareas se definen a través de los componentes de Rails. Para agregar más tareas, se emplea el directorio *lib/tasks*, en vez de alterar este archivo.
tmp/	Este directorio contiene archivos temporales que se crean durante la ejecución de la aplicación como la caché y los *pids* del servidor de Rails.

Archivo/Carpeta	Propósito
spec/	Este directorio sustituye al directorio *test* por defecto de Rails y contiene todos los archivos con los *specs*, organizados por subdirectorios en función de su tipo, que se ejecutan para validar el correcto funcionamiento de la aplicación mediante el *framework* RSpec. Aquí también se ubican los archivos para configurar este *framework* y las gemas de terceros a emplear con él.
storage/	Este directorio contiene los archivos que se asocian a las instancias de un modelo mediante Active Storage cuando se trabaja en el entorno de desarrollo.
vendor/	Este directorio contiene todo el código de terceros.
Gemfile **Gemfile.lock**	Estos archivos permiten especificar que dependencias de gemas son necesarias para la aplicación. Estos archivos son usados por la gema Bundler para su instalación y actualización.
Procfile	Este archivo se emplea para indicarle a Heroku que inicie la aplicación con un servidor de Rails específico.
.gitignore	Este archivo indica a Git que archivos (o patrones) deberían ignorarse a la hora de hacer confirmaciones (*commits*).

Tabla 3-1. Estructura general del directorio de una aplicación Ruby on Rails

3.3. Modelo de la base de datos

En esta sección se muestra el diagrama entidad-relación generado automáticamente por la gema rails-erd basándose en los modelos Active Record de la aplicación [22]. Para cada modelo se incluyen tanto sus atributos como sus relaciones con otros modelos. La notación seguida para este diagrama es la siguiente:

- Los modelos se representan con una caja divida en dos compartimentos. El primero se corresponde con el nombre del modelo y el segundo con los atributos de este. Cabe mencionar que en el esquema de la base de datos este modelo se traducirá a una tabla con el nombre del modelo en plural siguiendo las convenciones por defecto de Ruby on Rails mediante las migraciones generadas automáticamente al crear dicho modelo. En la Figura 3-3 se muestra un ejemplo sacado de la página oficial de la gema.

```ruby
ActiveRecord::Schema.define do
  create_table "photographs" do |t|
    t.decimal  :aperture
    t.binary   :data,        :null => false
    t.text     :description,                 :limit => 512
    t.string   :filename,    :null => false, :limit => 64
    t.boolean  :flash
    t.integer  :iso
    t.float    :shutter_speed
    t.datetime :taken_at,    :null => false
  end
end
```

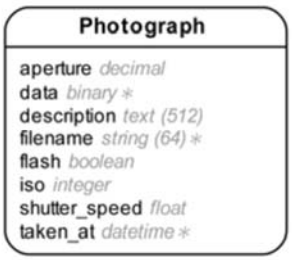

Figura 3-3. Ejemplo de modelo

- Las relaciones uno-a-uno se representan con una línea continua simple. En la Figura 3-4 se muestra un ejemplo.

```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
```

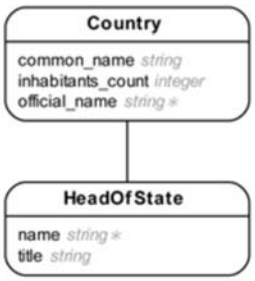

Figura 3-4. Ejemplo de relación uno-a-uno

- Las relaciones uno-a-muchos se representan como una línea continua simple en donde la cardinalidad muchos tiene forma de flecha cerrada. En la Figura 3-5 se muestra un ejemplo.

```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
```

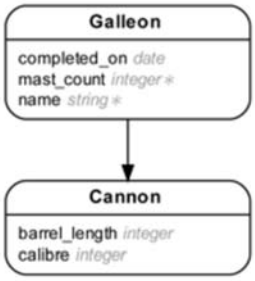

Figura 3-5. Ejemplo de relación uno-a-muchos

- Los modelos especializados de un modelo padre mediante el patrón de diseño *Single Table Inheritance* que soporta Ruby on Rails se muestran en color gris. La relación de herencia apunta al modelo padre y se trata de una línea continua finalizada con una flecha abierta. En la Figura 3-6 se muestra un ejemplo.

```ruby
class Beverage < ActiveRecord::Base
end

class Beer < Beverage
end

class Whisky < Beverage
end
```

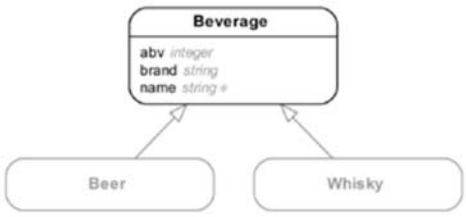

Figura 3-6. Ejemplo de relación de herencia

- Las relaciones polimórficas emplean, en conjunto, las notaciones anteriores. En la Figura 3-7 se muestra un ejemplo.

```ruby
class Barricade < ActiveRecord::Base
  # Defensible structures have many soldiers.
  has_many :soldiers, :as => :defensible
end

class Stronghold < ActiveRecord::Base
  # Defensible structures have many soldiers.
  has_many :soldiers, :as => :defensible
end

class Soldier < ActiveRecord::Base
  # Soldiers belong to a defensible structure.
  belongs_to :defensible, :polymorphic => true
end
```

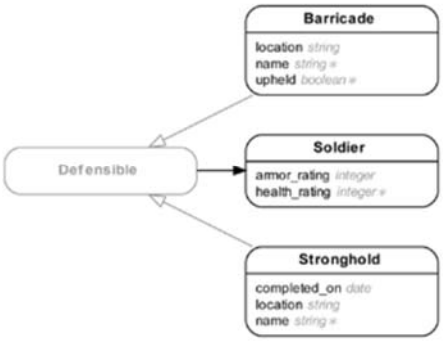

Figura 3-7. Ejemplo de relación polimórfica

Para una mejor visualización este diagrama se divide en tres partes. El diagrama completo se incluye en la sección Diagrama entidad-relación completo en el apartado de Anexos. Cada parte contiene los modelos creados en una iteración. No obstante, se debe tener en cuenta que un modelo creado, por ejemplo, en la primera iteración puede sufrir modificaciones para añadirle nuevos atributos necesarios en las siguientes iteraciones. A continuación se describen los atributos más relevantes de la mayoría de los modelos. El uso del resto de los atributos no descritos se puede deducir de su nombre.

Figura 3-8. Diagrama entidad-relación - Parte 1

En la Figura 3-8 se muestra el único modelo creado en la Iteración 1. Dejando a un lado los atributos que crea por defecto la gema Devise, los atributos más relevantes de este modelo son:

- **admin:** almacena un dato booleano para indicar si el usuario es administrador o no.
- **stripe_id:** almacena el id del cliente de Stripe enlazado al usuario. Se utiliza principalmente para recuperar dicho cliente desde la aplicación empleando la API de Stripe.
- **avatar:** almacena el avatar asociado al usuario.
- **card_last4 / card_brand:** almacenan respectivamente los cuatro últimos dígitos de la tarjeta de crédito del usuario y la marca de la tarjeta. Por ejemplo: 4444 y Visa.

- **downloaded / uploaded_files_count:** almacenan respectivamente el número de archivos descargados y subidos por el usuario.

En la Figura 3-9 se muestran los modelos creados en la Iteración 2. De aquí se comentan los atributos más relevantes de estos modelos.

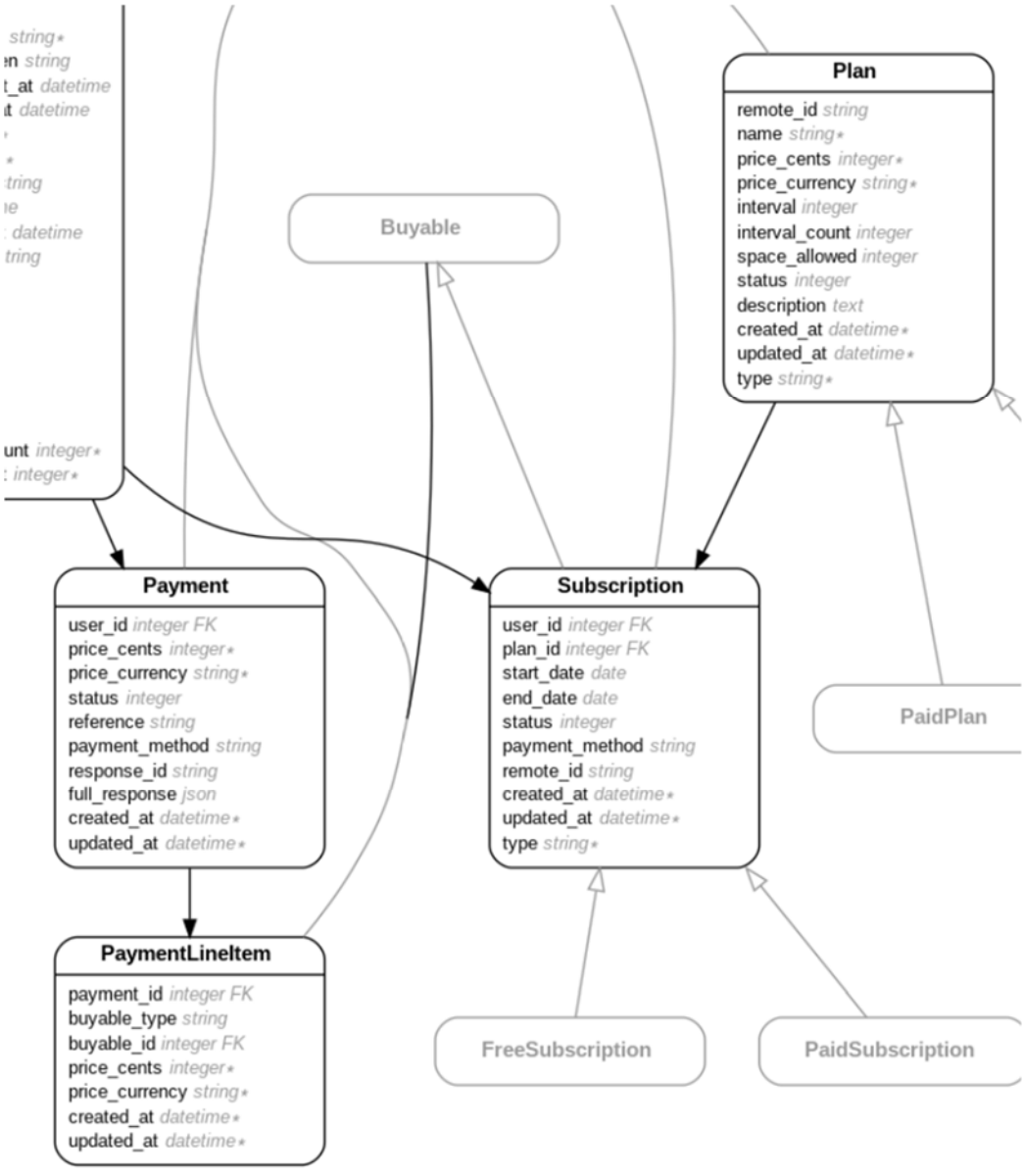

Figura 3-9. Diagrama entidad-relación - Parte 2

Los atributos más relevantes del modelo *Subscription* son:

- **start_date / end_date:** almacenan respectivamente la fecha de inicio y de fin de la suscripción.

- **status:** almacena en forma de entero el estado de la suscripción.

- **remote_id:** almacena el identificador de la suscripción de Stripe asociada a la suscripción. Esto se aplica también al modelo *Plan*.

- **type:** almacena el tipo al que pertenece la suscripción. Esto se aplica también al modelo *Plan*.

Los atributos más relevantes del modelo *Plan* son:

- **price_cents / price_currency:** almacenan respectivamente el precio (en céntimos) y la moneda (EUR) del plan. Estos atributos son creados por la gema money-rails. Esta permite trabajar con dinero en aplicaciones Ruby on Rails. Su principal ventaja es que aporta más precisión evitando que los clientes puedan perder dinero cuando las cantidades son elevadas. Esto se aplica también a los modelos *Payment* y *PaymentLineItem*.

- **space_allowed.** almacena en forma de entero el espacio de almacenamiento disponible para el usuario si está suscrito a este plan.

- **interval / interval_count:** almacenan respectivamente el intervalo del plan (diario, semanal, mensual, anual) y el número de intervalos (especificado por interval) entre la facturación de la suscripción a este plan. Por ejemplo, si interval tiene como valor month y interval_count tiene como valor 3, entonces se cobrará al usuario cada 3 meses. En nuestro caso, por simplicidad, se cobra al usuario mensual o anualmente.

Los atributos más relevantes del modelo *Payment* son:

- **reference:** almacena la referencia del pago.

- **response_id:** almacena el identificador de la factura de Stripe asociada al pago.

- **full_response:** almacena en formato JSON los datos de la factura enviada por Stripe a la aplicación. En estos datos se encuentra la URL para que el usuario pueda acceder a dicha factura desde su navegador web.

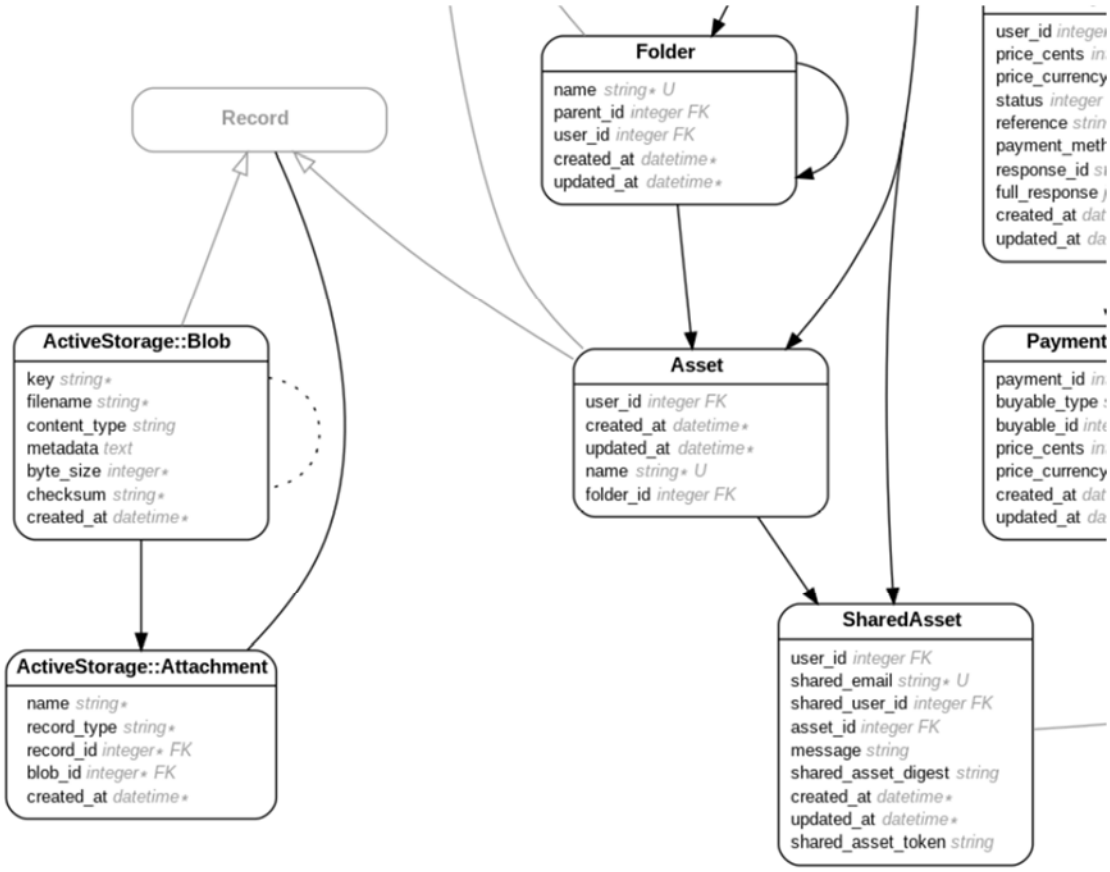

Figura 3-10. Diagrama entidad-relación - Parte 3

En la Figura 3-10 se muestran los modelos creados en la Iteración 3. De aquí se comentan los atributos más relevantes del modelo *SharedAsset*:

- **shared_email:** almacena la dirección de correo electrónico del usuario con el que se ha compartido el archivo.

- **message:** almacena el mensaje que se enviará junto al enlace para acceder al archivo en el correo electrónico.

- **shared_asset_token:** almacena el token para que el usuario con el que se ha compartido el archivo pueda acceder a él desde su cuenta de usuario y desde el correo electrónico.

4. Desarrollo

En este apartado se establece el alcance de la implementación, enumerando todos los casos de usos implementados y se comentan los aspectos más relevantes de la implementación de estos en cada una de las iteraciones realizadas en la fase de desarrollo.

4.1. Alcance de la implementación

En este trabajo se lleva a cabo el desarrollo de un prototipo de aplicación que permite validar las ideas obtenidas en los apartados de Análisis y Diseño. Además, permitirá cumplir con los objetivos propuestos siendo uno de ellos el aprendizaje de los recursos implicados. Es importante destacar que no se implementarán todos los casos de uso sino los necesarios para tal fin dejando a un lado aquellas que excederían el tiempo y los recursos disponibles. Estas funcionalidades se dejarán como línea de trabajo futura en la sección de Trabajo futuro del apartado de Conclusiones. Dicho esto, los casos de uso a desarrollar en este prototipo, que también se muestran marcados en color verde en la Tabla 2-7, son:

- Registrarse.
- Adquirir suscripción.
- Realizar pago.
- Cambiar suscripción.
- Cancelar suscripción.
- Iniciar sesión.
- Cambiar contraseña.
- Confirmar cuenta.
- Cancelar cuenta.
- Modificar perfil.
- Recuperar contraseña.
- Consultar estadísticas de uso.
- Consultar factura.
- Consultar historial de pagos.
- Subir archivo.
- Bajar archivo.
- Compartir archivo.
- Dejar de compartir archivo.
- Renombrar archivo.
- Eliminar archivo.

- Ver detalles del archivo.
- Consultar carpeta.
- Crear carpeta.
- Eliminar carpeta.
- Ver detalles de carpeta.
- Renombrar carpeta.
- Eliminar usuario.
- Consultar usuario.
- Consultar estadísticas de uso del usuario.
- Consultar historial de pagos del servicio.
- Cambiar método de pago.

Todos los casos de uso se reparten en cada una de las iteraciones realizadas en esta fase.

4.2. Iteración 1

En esta iteración se lleva a cabo la implementación del sistema de autenticación con el propósito de permitir a los usuarios registrarse, iniciar sesión, cancelar su cuenta, modificar su perfil o recuperar su contraseña. Para ello, se utiliza la gema Devise que facilita esta tarea al proporcionarnos todos los recursos necesarios para ello. Algunos de estos son métodos ayudantes (*helpers*) que están disponibles en nuestros controladores y que permiten obligar a un usuario a autenticarse antes de ejecutar una acción específica (*authenticate_user!*), obtener la instancia del modelo del usuario que ha iniciado sesión actualmente (*current_user*) y saber si el usuario actual ha iniciado sesión (*user_signed_in?*). A continuación se describen los aspectos más importantes de su uso en el prototipo:

El primer aspecto clave es la creación del modelo *User* el cual se configura con los módulos de Devise que permiten utilizar la mayoría de las funcionalidades de su sistema de autenticación.

El segundo aspecto clave es permitir a los usuarios registrarse con su nombre y apellidos además de con su dirección de correo electrónico y contraseña. Para ello, se añaden los atributos *name* y *last_name* al modelo anterior y se personaliza el controlador *Devise::RegistrationsController*. Esto se hace creando un controlador *RegistrationsController* propio que herede del anterior (véase Figura 4-1). En este nuevo controlador se sobrescriben los métodos *sign_up_params* y *account_update_params* para poder incluir los nuevos atributos durante el registro y la actualización del perfil.

```ruby
class RegistrationsController < Devise::RegistrationsController

private

  def sign_up_params
    params.require(:user).permit(:name, :last_name, :email, :password,
      :password_confirmation, :avatar, :avatar_cache, :remove_avatar)
  end

  def account_update_params
    params.require(:user).permit(:name, :last_name, :email, :password,
      :password_confirmation, :current_password, :avatar, :avatar_cache,
      :remove_avatar)
  end
```

Figura 4-1. El controlador RegistrationsController personalizado

El tercer aspecto clave es la personalización de las vistas por defecto de Devise con el *framework* Bootstrap 4. Este se integra en el *asset pipeline* de Ruby on Rails mediante la gema bootstrap.

A parte de la implementación del sistema de autenticación anterior se realizan otras tres tareas clave:

La primera tarea es la creación del controlador *StaticPagesController* para gestionar las páginas del prototipo, por ejemplo, la página de inicio con la *landing page*.

La segunda tarea es la asignación de un avatar a cada usuario mediante la gema Carrierwave que nos permite subir archivos y asociarlos a una instancia de un modelo específico. Para ello, se añade un nuevo atributo *avatar* al modelo *User*.

La tercera y última tarea es la configuración del componente Action Mailer de Ruby on Rails para permitir que el *mailer* de Devise envíe los mensajes a los usuarios con las instrucciones de confirmación de cuenta o con las instrucciones de recuperación de su contraseña.

4.3. Iteración 2

En esta iteración se lleva a cabo la integración del sistema de pagos recurrentes de Stripe (Stripe Billing) con el propósito de cobrar a los usuarios mediante un esquema de suscripciones a planes de pago y así implementar los casos de uso de adquirir suscripción, realizar pago, cambiar suscripción, cancelar suscripción, cambiar método de pago, consultar el historial de pagos y consultar factura. A continuación se describen las tareas principales realizadas en esta iteración.

La primera tarea es la configuración de la aplicación para acceder a los recursos que ofrece Stripe a través de su API y que conforman el modelo de suscripciones que soporta Stripe Billing. Para ello se emplea la gema stripe que integra la interfaz Ruby para acceder a esta API. Esta gema requiere de la clave secreta asociada a nuestra cuenta de usuario.

La segunda tarea es la adición de planes a la aplicación. Se crea el modelo *Plan* y el controlador *PlansController* con la acción *index* que se encarga de obtener los planes de la base de datos para mostrarlos en la vista asociada a la acción. Para distinguir entre planes gratuitos y planes de pago se emplea el patrón de diseño *Single Inheritance Table* (STI) que soporta el *framework* Ruby on Rails. De esta forma se heredan dos modelos llamados *FreePlan* y *PaidPlan* manteniendo una sola tabla en la base de datos.

La tercera tarea es la adición de suscripciones a la aplicación. Se crea el modelo *Subscription* y el controlador *SubscriptionsController* que se encarga de los procesos de adquirir una suscripción (además de realizar el primer pago), cambiar una suscripción y cancelar una suscripción. Para distinguir entre una suscripción a un plan gratuito y una suscripción a un plan de pago también se aplica el patrón de diseño anterior. De esta forma tenemos los modelos *FreeSubscription* y *PaidSubscription* que heredan del modelo *Subscription*. Cuando un usuario se registra en la aplicación se emplea un *callback* para asociarle una suscripción gratuita al plan gratuito. Esta suscripción se actualiza a una suscripción de pago cuando se adquiere esta.

Para no hacer muy extenso el controlador *SubscriptionsController* todo el flujo de trabajo de cada funcionalidad anterior se encapsula en objetos Ruby llamados *workflows*. Estos son:

- **CreatesSubscriptionViaStripe.** Este *workflow* se encarga del proceso de adquirir una suscripción En la Figura 4-2 se muestra su código fuente.

- **CancelsStripeSubscription.** Este *workflow* se encarga del proceso de cancelar una suscripción.

- **ChangesStripeSubscriptionPlan.** Este *workflow* se encarga del proceso de cambiar una suscripción.

```ruby
class CreatesSubscriptionViaStripe

  attr_accessor :user, :token, :plan, :success, :error_message

  def initialize(user:, plan:, token:)
    @user = user
    @plan = plan
    @token = token
    @success = false
  end

  def subscription
    @subscription ||= Subscription.create!(
        user: user, plan: plan,
        start_date: Time.zone.now.to_date,
        end_date: plan.end_date_from,
        status: :waiting, type: "PaidSubscription")
  end

  def run
    Payment.transaction do
      stripe_customer = StripeCustomer.new(user: user)
      return unless stripe_customer.valid?
      stripe_customer.source = token
      stripe_customer.save_non_sensible_card_info
      subscription.make_stripe_payment(stripe_customer)
      stripe_customer.add_subscription(subscription)
      @success = true
    end
  rescue Stripe::StripeError => e
    @error_message = I18n.t("stripe.errors.#{e.code}")
    Rollbar.error(e)
  end

  def redirect_on_success_url
    root_path
  end

end
```

Figura 4-2. El workflow CreatesSubscriptionViaStripe

Los *workflows* anteriores requieren de dos objetos Ruby llamados *StripeCustomer* y *StripeToken* que encapsulan los accesos a la API de Stripe mediante la gema stripe para manipular los objetos *Customer* y *Token* de Stripe asociados a un usuario de la aplicación.

Al modelo *User* se le añaden dos métodos de instancia para obtener tanto la suscripción gratuita asociada a un usuario que puede estar inactiva o activa y la suscripción actual que puede ser gratuita o de pago.

En la vista asociada a la acción *new* del controlador *SubscriptionsController* se emplea el formulario de pago creado mediante los elementos prefabricados que proporciona el

componente Elements de la librería Stripe.js. En este formulario el usuario introduce sus datos de pago sensibles. En base a ellos, la librería compone un token que se pasa como parámetro a la acción *create* del controlador para iniciar el proceso de adquirir la suscripción que es el que se enlaza como método de pago al cliente de Stripe asociado al usuario para realizar el cobro de la suscripción.

El anterior proceso es similar al utilizado para implementar la funcionalidad de cambiar el método de pago creando para ello el controlador *PaymentMethodsController* y el *workflow ChangesPaymentMethod* que simplemente emplea otro formulario para recolectar el nuevo método de pago mediante un token que sustituye al token actual del cliente de Stripe enlazado al usuario.

La cuarta tarea es la creación de un controlador llamado *StripeWebhookController* para recibir y manejar los eventos que emite Stripe cuando se actualizan los objetos asociados a nuestra cuenta de usuario en la plataforma. Estos objetos pueden ser los clientes, las suscripciones, los planes, las facturas y demás. En nuestro caso, solo se admiten los eventos de cancelación de una suscripción, de pago exitoso y de pago fallido. Este controlador cuenta con una sola acción llamada *action* que recibe el evento mandado por Stripe mediante una petición POST a la aplicación. Esta acción ejecuta un objeto *workflow* cuyo nombre coincide con el tipo del evento. Los *workflows* disponibles son:

- **CustomerSubscriptionDeleted.** Este *workflow* se encarga de cancelar la suscripción del usuario tras cuatro intentos de cobro de la factura fallidos. Esto normalmente ocurrirá si el método de pago asociado al usuario deja de ser válido (la tarjeta de crédito caduca o no hay dinero en la tarjeta) y no lo cambia tras transcurrir esos intentos.

- **InvoicePaymentFailed.** Este *workflow* se encarga de crear el pago asociado a la factura de Stripe con estado fallido si se trata del primer intento de pago o actualizarlo nuevamente al estado fallido si se trata de otro nuevo intento. En ambos casos se inactiva la suscripción de pago actual y se activa la suscripción gratuita.

- **InvoicePaymentSuccedeed.** Este *workflow* se encarga de crear el pago asociado a la factura de Stripe con estado exitoso si se trata del primer intento de pago o lo actualiza a estado exitoso si el anterior intento de pago fue fallido. En ambos casos se activa la suscripción de pago, se actualiza su fecha de expiración y se inactiva la suscripción gratuita.

- **NullHandler.** Este *workflow* no realizada nada en especial. Se ejecuta cuando el tipo del evento no coincide con el nombre de algunos de los *workflows* anteriores.

En la Figura 4-3 se muestra el código fuente de este controlador.

```ruby
class StripeWebhookController < ApplicationController
  before_action :check_signature, only: [:action]

  protect_from_forgery except: :action

  def action
    @event_data = JSON.parse(request.body.read)
    workflow = workflow_class.new(verify_event)
    workflow.run
    if workflow.success
      head :ok
    else
      head :internal_server_error
    end
  end

  private def verify_event
    Stripe::Event.retrieve(@event_data["id"])
  rescue Stripe::InvalidRequestError => e
    Rollbar.error(e)
    nil
  end

  private def workflow_class
    event_type = @event_data["type"]
    "StripeHandler::#{event_type.tr('.', '_').camelize}".constantize
  rescue NameError
    StripeHandler::NullHandler
  end

  private def check_signature
    payload = request.body.read
    signature_header = request.env['HTTP_STRIPE_SIGNATURE']
    singing_key = Rails.application.credentials[Rails.env.to_sym][:stripe][:signing_key]
    event = nil
    begin
      event = Stripe::Webhook.construct_event(
        payload, signature_header, singing_key
      )
    rescue JSON::ParserError => e
      Rollbar.error(e)
      head :bad_request
    rescue Stripe::SignatureVerificationError => e
      Rollbar.error(e)
      head :bad_request
    end
  end

end
```

Figura 4-3. El controlador StripeWebhookController

Los pagos anteriores están representados por registros del modelo *Payment*. Cada registro de este modelo tiene asociado un registro del modelo *PaymentLineItem* por cada ítem adquirido. En nuestro caso este registro se enlaza con una suscripción mediante una asociación polimórfica.

La quinta tarea consiste en la creación de un *mailer* llamado *SubscriptionMailer* para enviar correos electrónicos a los usuarios cuando:

- Su suscripción es cancelada manualmente.
- Su suscripción es cancelada por impago.
- Su suscripción se realizada correctamente.
- El pago anual o mensual de su suscripción se realiza correctamente.
- El pago anual o mensual de su suscripción se realiza incorrectamente.

La última tarea a destacar consiste en la integración y adaptación de la plantilla SB Admin utilizada para implementar el panel de administración del usuario cliente y del usuario administrador. Aquí también se incluye la adición de la paleta de colores comentada en la sección de Mockups, diseño de la interfaz y paleta de colores.

4.4. Iteración 3

En esta iteración se lleva a cabo la implementación de las funcionalidades básicas para la gestión de los archivos y carpetas por parte de los usuarios de la aplicación. A continuación, se comentan los aspectos claves de esta implementación.

El primer aspecto clave es la creación del controlador *AssetsController* que cuenta con las acciones necesarias para mostrar los detalles de un archivo (*show*), subir un archivo (*new* y *create*), renombrar un archivo (*edit* y *update*), eliminar un archivo (*destroy*) y descargar un archivo (*download*).

Para representar el concepto de archivo se crea el modelo *Asset*. A este modelo se le asocia el archivo que se sube con el formulario mediante el método *has_one_attached* proporcionado por el componente Active Storage del *framework* Ruby on Rails. Cada registro de este modelo se asocia únicamente con un usuario.

El segundo aspecto clave es la creación del controlador *FoldersController* que cuenta con las acciones necesarias para mostrar los detalles de una carpeta (*show*), crear una carpeta (*new* y *create*), renombrar un carpeta (*edit* y *update*), eliminar una carpeta (*destroy*) y consultar un carpeta (*browse*).

Para representar el concepto de carpeta se crea el modelo *Folder*. A este modelo se le asocian uno o varios registros del modelo *Asset*. Además, una carpeta pertenece a un único usuario y puede contener varias subcarpetas o archivos. Para representar el directorio de carpetas se emplea la gema acts_as_tree que permite organizar los registros de un modelo Active Record (en este caso *Folder*) mediante relaciones padre-hijo. Solo basta con añadir un atributo *parent_id* y el método *acts_as_tree* en dicho modelo. De esta manera, las carpetas que se encuentren en el directorio raíz tendrán este atributo a nulo mientras que las que se encuentren en un carpeta tendrán, como valor para este atributo, el identificador de la carpeta. Por otro lado, para asociar los archivos a carpetas se añade un atributo *folder_id* que funciona de manera similar.

Para los dos modelos anteriores, cabe destacar el uso de validaciones de unicidad para garantizar que en el directorio raíz y en una carpeta o subcarpeta no se permita crear una carpeta con el mismo nombre a una existente o subir un archivo si ya lo está en la actual ubicación.

Por otra parte, cabe destacar la protección de los archivos y carpetas mediante la gema cancancan. Esta nos permite definir permisos para un usuario mediante el método *can* en un fichero centralizado llamado *ability.rb*. En nuestro caso la empleamos para permitir que solo el usuario dueño del archivo o carpeta pueda acceder a ella. Esto se hace principalmente utilizando el método *authorize!* en los controladores anteriores. En la Figura 4-4 y la Figura 4-5 se muestran el contenido del fichero *ability.rb* y un ejemplo del método *authorize!*.

```
3   class Ability
4     include CanCan::Ability
5
6     def initialize(user)
7
8       can [:show, :edit, :update, :destroy, :share, :members], Asset, user_id: user.id
9       can [:show, :new, :edit, :update, :destroy, :browse], Folder, user_id: user.id
10      can [:show], SharedAsset, shared_user_id: user.id
11
12      can :download, Asset do |asset|
13        asset.user_id == user.id || asset.shared_assets.map(&:shared_user_id).include?(user.id)
14      end
15
16    end
17  end
```

Figura 4-4. El fichero ability.rb

```ruby
def show
  authorize! :show, @asset, message: "No tienes acceso a este archivo."
end
```

Figura 4-5. Ejemplo de uso del método authorize!

El tercer aspecto clave es la compartición de archivos con otros usuarios. Para ello se crea el controlador *SharedAssetsController* que cuenta con las acciones necesarias para compartir un archivo (*new* y *create*), consultar un archivo compartido (*show*), dejar de compartir un archivo con un usuario (*destroy*) y consultar los usuarios con el que se ha compartido un archivo (*members*).

La acción de compartir un archivo se representa mediante el modelo *SharedAsset.* Este se asocia con el dueño del archivo (*user*), el archivo a compartir (*asset*) y el usuario con el que se comparte (*shared_user*). Así mismo, también guarda la dirección de correo electrónico de este usuario y el mensaje que se muestra en el correo electrónico junto al enlace para acceder al archivo.

La recuperación de un archivo compartido cuando el usuario accede a él se realiza mediante un token que se asigna cuando se comparte con ese usuario. Este token también se almacena encriptado mediante la gema bcrypt a partir de un *callback* llamado *create_shared_asset_digest*. En el controlador *SharedAssetsController,* antes de realizar la acción *show,* se verifica la autenticidad del token pasado como parámetro mediante el filtro *require_valid_token* que ejecuta el método de instancia *authenticated?*. En la Figura 4-6 y Figura 4-7 se muestra su código respectivamente.

```ruby
83    def require_valid_token
84      unless (@shared_asset && @shared_asset.authenticated?("shared_asset", params[:id]))
85        redirect_to root_path
86      end
87    end
```

Figura 4-6. El filtro require_valid_token

```ruby
25    def authenticated?(attribute, token)
26      digest = send("#{attribute}_digest")
27      return false if digest.nil?
28      BCrypt::Password.new(digest).is_password?(token)
29    end
```

Figura 4-7. El método de instancia authenticated?

A todo lo anterior hay que añadir que cuando se comparte un archivo con un usuario, si este usuario no está registrado, el valor del atributo *shared_user_id* permanece a nulo. Para garantizar que el usuario se asocie con estos archivos en algún momento, se le obliga a autenticarse cuando intente acceder al archivo compartido. Este usuario deberá crearse una cuenta de usuario en la aplicación. Aquí es donde entra el *callback* añadido en el modelo *User* llamado *check_and_assign_shared_ids_to_shared_assets* que se encarga de actualizar el valor del atributo *shared_user_id* de los registros del modelo *SharedAsset* al identificador del usuario cuyo email coincida con el valor del atributo *shared_email* (véase Figura 4-8).

```
94      def check_and_assign_shared_ids_to_shared_assets
95        shared_assets_with_same_email = SharedAsset.where(shared_email: self.email)
96        if shared_assets_with_same_email
97          shared_assets_with_same_email.each do |shared_asset|
98            shared_asset.shared_user_id = self.id
99            shared_asset.save
100         end
101       end
102     end
```

Figura 4-8. El callback check_and_assign_shared_ids_to_shared_assets

En el repositorio adjuntado a este documento se pueden consultar el resto de código fuente necesario para que esta funcionalidad funcione correctamente. Por ejemplo, validaciones, filtros, rutas, etc.

El cuarto y último aspecto clave es la creación del *mailer UserMailer* para enviar por correo electrónico el enlace para acceder al archivo compartido mediante el método *shared_link_email*.

4.5. Iteración 4

En esta iteración se lleva a cabo la implementación de las funcionalidades de listar usuarios, consultar usuario, eliminar usuario, consultar las estadísticas de uso de un usuario y consultar el historial de pagos del servicio con el objetivo de establecer el panel de administración del usuario administrador y así sentar las bases para las futuras funcionalidades de este panel. Algunas de ellas se mencionan en la sección de Trabajo futuro del apartado de Conclusiones. A continuación se comentan los aspectos más relevantes de esta implementación.

El primer aspecto clave es la creación del *namespace* para organizar el código fuente de la aplicación destinado a las funcionalidades del usuario administrador. Para ello se emplea en el

fichero *config/routes.rb* el método *namespace* pasándole como opción el símbolo *:admin* (véase Figura 4-9). Esto hace que todas las rutas y recursos que se configuren dentro de este *namespace* se prefijen con la palabra admin. Además, esto abre la posibilidad de crear controladores específicos para el usuario administrador dentro de un subdirectorio también llamado *admin* en el directorio *app/controllers*.

```
41    namespace :admin do
42      get "service_statistics", to: "panel#service_statistics"
43      resources :users do
44        get 'statistics', to: "users#statistics"
45      end
46      resources :payments
47    end
```

Figura 4-9. Método namespace del fichero config/routes.rb

Dentro de este subdirectorio se crea el controlador *Admin::ApplicationController* que hereda de *ApplicationController*. En este controlador se emplea el filtro *authorize_admin* para verificar que el usuario que intenta acceder sea un usuario administrador. Primero se pide que se autentique y luego se comprueba que el atributo *admin*, que se añadió en la Iteración 1 para este momento, esté a verdadero. Si no, se le redirige a la ruta raíz de la aplicación mostrándole un mensaje de error indicándole que no tiene permiso para realizar la acción. Todos los controladores creados a partir de aquí heredan de este nuevo controlador. De esta manera todos ellos estarán protegidos de accesos no autorizados. En la Figura 4-10 se muestra su código fuente.

```
1   class Admin::ApplicationController < ApplicationController
2     before_action :authorize_admin
3
4     private
5
6       def authorize_admin
7         authenticate_user!
8         unless current_user.admin?
9           redirect_to root_path, alert: "Necesitas permisos de administrador para realizar esta acción."
10        end
11      end
12
13  end
```

Figura 4-10. El controlador Admin::ApplicationController

El segundo aspecto clave es la creación del controlador *Admin::UsersController*. Este cuenta con las acciones necesarias para los usuarios registrados en la aplicación (*index*), consultar los detalles de un usuario (*show*), eliminar un usuario (*destroy*) y consultar las estadísticas de uso de un usuario (*statistics*). De aquí cabe destacar que para la eliminación de un usuario se emplea

un *workflow* llamado *DeleteUser* que se encarga de eliminar al usuario en la base de datos local y también de cancelar su suscripción de pago (si lo está) en Stripe.

El tercer y último aspecto clave es la creación del controlador *Admin::PaymentsController*. Este solo cuenta con la acción *index* que recupera del modelo todos los pagos de la base de datos y los muestra en su vista asociada.

5. Recursos utilizados

En este apartado se enumeran los recursos utilizados para llevar a cabo el desarrollo del prototipo y lograr así la consecución de los objetivos propuestos al principio. Para cada recurso se hace una breve descripción mencionando alguna de sus características y se indica para que se ha empleado en este proyecto.

5.1. Lenguajes de programación

5.1.1. JavaScript

Figura 5-1. Logo de JavaScript [Wikipedia]

JavaScript es un lenguaje de programación interpretado que implementa el estándar ECMAScript. Se trata de un lenguaje orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico [23].

Se utiliza principalmente en el lado del cliente, implementado como parte de un navegador web, permitiendo mejoras en la interfaz de usuario y páginas web dinámicas. No obstante, también se utiliza del lado del servidor en muchos entornos sin navegador.

Este lenguaje se emplea en este proyecto con el propósito de incorporar la siguientes librerías [24]:

- **Stripe.js.** Es la librería JavaScript de Stripe que se encarga de obtener el token a partir de los datos de pago recolectados en el formulario.

- **Stripe Elements.** Es una característica que soporta la anterior librería con la que se construye los formularios de pago en el prototipo a partir de componentes de interfaz de usuario preconstruidos.

5.1.2. Ruby

Figura 5-2. Logo de Ruby [Wikipedia]

Ruby es un lenguaje de programación creado por el programador japonés Yukihiro Matsumoto que presenta las siguientes características [25]:

- Es un lenguaje de propósito general. Se puede desarrollar cualquier tipo de aplicación con él.

- Es un lenguaje interpretado, no es necesario compilarlo.

- Es un lenguaje dinámico, flexible y multiparadigma. Permite la programación funcional, la programación reflexiva y la programación orientada a objetos.

- Es de alto nivel. Combina partes de los lenguajes de programación Ada, Perl, Smalltalk y Python.

- Es de código abierto y multiplataforma.

Este lenguaje se emplea para el desarrollo de toda la funcionalidad que proporciona el prototipo en combinación con la siguiente tecnología.

5.2. Tecnologías

5.2.1. Ruby on Rails

Figura 5-3. Logo de Ruby on Rails [Wikipedia]

Ruby on Rails es un *framework*, creado por David Heinemeier, para el desarrollo de aplicaciones web modernas de código abierto escrito en el lenguaje de programación Ruby, que sigue el patrón arquitectónico Modelo Vista Controlador (MVC) [26].

Está diseñado para que los programadores puedan desarrollar aplicaciones del mundo real de manera simple, escribiendo menos código a como lo harían con otros *frameworks* y estableciendo un mínimo de configuración necesaria.

Este *framework* se emplea como base para el desarrollo del prototipo. Además, su aprendizaje es uno de los objetivos de este proyecto a nivel académico.

5.2.2. jQuery

Figura 5-4. Logo de jQuery [Wikipedia]

jQuery es una biblioteca multiplataforma de JavaScript, creada inicialmente por John Resig, que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la tecnología AJAX en páginas web [27].

Esta biblioteca se emplea en este proyecto de forma implícita en el uso del *framework* Ruby on Rails y el *framework* Bootstrap. Además, también se utiliza para poder emplear el siguiente plugin de jQuery:

- **bs-custom-file-input.** Es un plugin de jQuery que permite crear un campo de selección de archivos personalizado con el botón del navegador para cargar archivos lo cual el *framework* Bootstrap no permite por defecto.

5.2.3. HTML

Figura 5-5. Logo de HTML5 [Wikipedia]

HTML es un lenguaje de marcado para la elaboración de páginas web. Es un estándar que define una estructura básica y un código para la definición de contenido de una página web, como texto, imágenes, vídeos, juegos, entre otros [28].

Este lenguaje de marcado se emplea para la elaboración de las vistas del prototipo en combinación con el motor de plantillas por defecto de Ruby on Rails, ERB (*Embbeded Ruby*), que nos permite mezclar código Ruby con código HTML para generar páginas dinámicas usando datos de nuestra base de datos [29].

5.2.4. CSS3

Figura 5-6. Logo de CSS3 [Wikipedia]

CSS es un lenguaje de diseño gráfico para definir y crear la presentación de un documento estructurado en un lenguaje de marcado, como por ejemplo, HTML. Es muy usado para establecer el diseño visual de los documentos web, e interfaces de usuario [30].

Esta tecnología se utiliza principalmente en combinación con el *framework* Bootstrap.

5.2.5. Bootstrap 4

Figura 5-7. Logo de Boostrap [Wikipedia]

Bootstrap es una biblioteca multiplataforma de código abierto desarrollada por Twitter para el diseño de sitios y aplicaciones web que se ajustan a cualquier resolución y dispositivo. Contiene plantillas de diseño basado en HTML y CSS, así como extensiones de JavaScript adicionales [31].

En este proyecto se utiliza su última versión, Bootstrap 4, para el diseño de la vistas del prototipo desarrollado con el *framework* Ruby on Rails.

5.3. Servicios

5.3.1. Amazon S3

Figura 5-8. Logo de Amazon S3 [Worldvectorlogo]

Tal y como se comentó en la sección de Motivación del apartado de Introducción, Amazon S3 es el servicio de almacenamiento que se utiliza en este proyecto para almacenar de manera segura los archivos de los usuarios de la aplicación.

5.3.2. Heroku

Figura 5-9. Logo de Heroku [Wikipedia]

Heroku es una plataforma de computación en la nube propiedad de la empresa Salesforce.com. Esta plataforma soporta distintos lenguajes de programación entre los que se encuentra Ruby. Además, dispone de un plan gratuito para desplegar aplicaciones en un *dyno* que es un contenedor basado en Linux que proporciona capacidad de cómputo dentro de la plataforma [32].

En nuestro caso se hace uso de Heroku para el despliegue continuo del prototipo en un entorno de producción. Los aspectos más relevantes de este despliegue se detallan en el apartado de Despliegue.

5.3.3. Stripe

Figura 5-10. Logo de Stripe [Wikipedia]

Stripe es el servicio de procesamiento de pagos en línea que permite a particulares o empresas recibir pagos por Internet en nuestras aplicaciones.

5.4. Entorno de desarrollo

5.4.1. Sublime Text 3

Figura 5-11. Logo de Sublime Text 3 [Wikipedia]

Sublime Text 3 es el editor de texto y editor de código fuente, escrito en C++ y Python por Jon Skinner [33], que se emplea como entorno de desarrollo integrado gracias a la instalación de una serie de paquetes que nos permite programar cómodamente en el *framework* Ruby on Rails. Algunos de ellos son:

- **ApplySyntax:** permite aplicar la sintaxis de los lenguajes de programación.
- **Bootstrap 4x Snippets:** permite insertar en las vistas el código HTML de los componentes de Bootstrap 4.
- **ERB Snippets:** permite insertar en las vistas el código ERB.
- **ReIndent:** permite aplicar la indentación en el código de dos espacio que es característica en el lenguaje de programación Ruby y en el *framework* Ruby on Rails.

5.4.2. Oracle VM VirtualBox, Ubuntu y RVM

El entorno de desarrollo local se configura en una máquina virtual que tiene instalado un sistema operativo de Ubuntu 18.04 LTS, una de las distribuciones de Linux basada en la arquitectura de Debian. Esta máquina virtual se ejecutada con el software de virtualización Oracle VM VirtualBox. El principal motivo para utilizar una máquina virtual es que resulta mucho más sencillo realizar la instalación y configuración de Ruby usando para ello la herramienta Ruby Version Manager (RVM)[2] en un sistema operativo tipo Unix. Además, eludimos los problemas que suelen aparecer cuando trabajamos con Ruby en Windows y podemos llevarnos dicho entorno de desarrollo a otro ordenador si así lo requerimos.

[2] Ruby Version Manager es una plataforma de software para sistemas operativos tipo Unix diseñados para administrar múltiples instalaciones de Ruby en el mismo dispositivo.

5.5. Gemas

Las gemas en Ruby son el equivalente a las librerías en otros lenguajes de programación. Cabe destacar que el propio Ruby on Rails es una gema llamada rails. Todas las gemas desarrolladas por la comunidad de Ruby se encuentran alojadas en un servidor denominado RubyGems. En la siguiente tabla indicamos cada una de las principales gemas que se emplean en el desarrollo del prototipo.

Nombre	Uso
devise	Esta gema se utilizada para implementar el sistema de autenticación en la aplicación. Ofrece los recursos necesarios para permitir a los usuarios registrarse, iniciar sesión, confirmar su cuenta, recuperar su contraseña, cancelar su cuenta y demás.
bootstrap	Esta gema se utiliza para integrar el *framework* Bootstrap 4 en el *asset pipeline* del *framework* Ruby on Rails.
pg	Esta gema se utiliza para integrar la interfaz Ruby necesaria para que la aplicación pueda gestionar la base de datos PostgreSQL en el entorno de producción.
devise-i18n	Esta gema se utiliza para traducir las vistas generadas por la gema Devise al idioma español.
faker	Esta gema se utiliza para poblar la base de datos con usuarios de ejemplos mediante la tarea de Rails, *users:create*. Estos datos consisten principalmente en el nombre, los apellidos y el avatar de esos usuarios.
rails-i18n	Esta gema se utiliza para poder traducir al idioma español los errores en la aplicación que por defecto están en el idioma inglés.
stripe	Esta gema se utiliza para acceder a la API de Stripe desde la aplicación. Por ejemplo, crear una suscripción a un cliente, cancelar una suscripción, cambiar el método de pago, etc.
money-rails	Esta gema se utiliza para integrar la gema money con Ruby on Rails. Permite trabajar con dinero en los modelos de Active Record de la aplicación.
carrierwave	Esta gema se utiliza para asociar un avatar a cada usuario de la aplicación.
font-awesome-sass	Esta gema se utiliza para integrar la librería de iconos Font Awesome en el *asset pipeline* del *framework* Ruby on Rails.

Nombre	Uso
mini_magick	Esta gema se utiliza para integrar la interfaz Ruby necesaria por la gema Carrierwave para acceder a la herramienta ImageMagick instalada en la máquina virtual para poder gestionar las imágenes. Por ejemplo, para redimensionar la imagen si esta es demasiado larga.
fog-aws	Esta gema proporciona una interfaz simple para trabajar con los servicios de Amazon Web Services. Se utiliza para configurar la gema Carrierwave en el entorno de producción con los parámetros para acceder al bucket de Amazon S3 y poder así guardar los avatares de los usuarios.
cancancan	Esta gema se utiliza para implementar un sistema de autorización simple en la aplicación. Se utiliza principalmente para proteger el acceso a los archivos y carpetas de los usuarios además de los archivos compartidos. No obstante, también se puede emplear para más cosas en el futuro. Por ejemplo, para añadir más roles en la aplicación.
aws-sdk-s3	Esta gema se utiliza para integrar la interfaz Ruby necesaria por el componente Active Storage de Ruby on Rails para acceder al SDK del servicio Amazon S3 de la plataforma Amazon Web Services en el entorno de producción para guardar los archivos en el *bucket* especificado en la configuración del componente.
rails-erd	Esta gema se utiliza para generar de forma automática el diagrama entidad-relación a partir de los modelos de Active Record creados en la aplicación.
acts_as_tree	Esta gema da soporte para organizar los registros de un modelo de Active Record en base a relaciones padre-hijo. En nuestro caso la empleamos para implementar el directorio de carpetas en la aplicación.
active_storage_validations	Esta gema proporciona las validaciones que por defecto no implementa Active Storage. Permite validar si se ha asociado un archivo al modelo *Asset*, validar el contenido de ese archivo, validar el tamaño en bytes de ese archivo, etc.

Tabla 5-1. Gemas utilizadas

5.6. Herramientas

5.6.1. GitHub

Figura 5-12. Logo de GitHub [Wikipedia]

Como servidor para el alojamiento del proyecto utilizando el sistema de control de versiones Git emplearemos GitHub [34]. Esto nos permitirá guardar el repositorio de nuestra aplicación de forma segura y disminuir las posibilidades de perder todo el trabajo realizado. Así mismo, podremos acceder el proyecto independientemente de que entorno de desarrollo nos encontremos.

5.6.2. StarUML

Figura 5-13. Logo de StarUML [Wikipedia]

Para la elaboración de los diagramas de clases y los diagramas de casos de uso se utilizará la herramienta UML de MKLab llamada StarUML [35]. Aunque sólo la usaremos para este tipo de diagramas lo cierto es que permite realizar cualquiera que se encuentre dentro del lenguaje unificado de modelado UML 2.

5.6.3. Justinmind

Figura 5-14. Logo de Justinmind [Web oficial]

Justinmind es la herramienta de creación de prototipos de aplicaciones web y móviles empleada para la elaboración del *mockup* de la interfaz de usuario de la aplicación.

5.6.4. ngrok

Figura 5-15. Logo de ngrok [luciano.im]

ngrok es la herramienta que nos permite crear un túnel accesible a través de un dominio aleatorio, que asigna la propia herramienta, para así acceder a un servidor local. En este trabajo empleamos su versión gratuita para probar en el entorno de desarrollo las llamadas del servicio Stripe a nuestro manejador de eventos a través de los *webhooks* configurados para tal fin.

5.7. Otros

En el prototipo se adapta la plantilla de Bootstrap 4 llamada SB Admin que se emplea para implementar el panel de administración del usuario cliente y del usuario administrador. Mediante el siguiente enlace se puede acceder a su repositorio:

https://github.com/BlackrockDigital/startbootstrap-sb-admin

Los iconos (archivos, nube del logo, vehículos de los planes de la tabla de precios, etc.) mostrados en la aplicación se han obtenido de las plataformas de Flaticon y Iconfinder.

Para el desarrollo de la aplicación nos ha servido de ayuda los siguientes tutoriales, libros o comunidades:

- Take My Money – Accepting Payments on the Web de Noel Rappin. Este libro está disponible en Safari Books Online:

  https://learning.oreilly.com/library/view/take-my-money/9781680502428/

- The Ruby on Rails Tutorial (Rails 5) de Michael Hartl. Este tutorial puede ser accedido mediante este enlace: https://www.railstutorial.org/

- Stack Overflow

- Github

6. Pruebas

Aunque en un principio nuestra intención era evaluar, validar y probar el correcto funcionamiento del prototipo, tanto en el entorno de producción como en el entorno de desarrollo, a partir de pruebas automatizadas utilizando el *framework* RSpec, la complejidad de las pruebas a medida que se avanzaba en el desarrollo y la falta de tiempo obligó a realizar esta fase mediante pruebas manuales simulando que éramos el usuario final.

Además de estas pruebas, también se han hecho comprobaciones de que el estilo de las vistas se vea correctamente, los mensajes de éxito o error sean entendibles por el usuario, etc.

7. Despliegue

El prototipo desarrollado en este proyecto se encuentra desplegado en un *dyno* de la plataforma de computación en la nube Heroku. El proceso de despliegue se ha llevado a cabo mediante una serie de etapas que se comentan a continuación [36].

Etapa 1: Configuración del entorno de desarrollo local (máquina virtual)

En esta primera etapa, se realiza la instalación de la interfaz de línea de comandos de Heroku (Heroku CLI) con la cual podemos ejecutar todos los comandos que nos proporciona esta plataforma desde la consola *bash* de Ubuntu [37].

Una vez instalada podemos iniciar sesión con nuestra cuenta de usuario de Heroku y crear el contenedor dentro del directorio de la aplicación para así enlazar el repositorio local con el repositorio remoto que administra dicho contenedor.

Etapa 2: Configuración de los servicios utilizados

En esta tercera etapa, se realiza la configuración de los servicios de Amazon S3 y Stripe, los cuáles se integraron durante el desarrollo del prototipo, con el objetivo de poder utilizarlos en el entorno de producción.

Por un lado, la configuración de Amazon S3 se divide en dos pasos:

1. El primer paso es la creación del *bucket* en donde se almacenan los archivos de los usuarios en el entorno de producción. Cabe recordar que, en el entorno de desarrollo, estos archivos se almacenan en el directorio *storage* de la aplicación. El acceso público a este *bucket* se bloquea para que sólo se pueda acceder programáticamente.

2. El segundo paso es la creación del usuario de IAM, el servicio que permite la administración y el acceso a los recursos de AWS de manera segura, al cual se le asocia la política de IAM que le brinda todos los privilegios para acceder de manera programática al *bucket*. Este usuario dispone de un identificador de clave de acceso y una clave de acceso secreta que se requieren para que Active Storage y Carrierwave accedan al *bucket* a través de la gemas aws-sdk-s3 y fog-aws que se instalan mediante el fichero Gemfile.

Por otro lado, la configuración de Stripe se divide también en dos pasos:

1. El primer paso es la creación de dos cuentas de usuario asociadas al mismo email para evitar conflictos entre el entorno de desarrollo y el entorno de producción. Aunque Stripe proporciona en una sola cuenta de usuario un entorno de prueba y un entorno de producción (*live*), éste último requiere darse de alta como empresa. Es por eso por lo que optamos por simular el entorno de producción con otro entorno de prueba en una cuenta de usuario independiente. Cada cuenta de usuario tiene asociadas una clave pública y una clave secreta que se requieren para acceder a la API de Stripe a través de la interfaz que implementa la gema stripe.

2. El segundo paso es la configuración del punto de conexión para el *webhook* en la cuenta de usuario destinada al entorno de producción. Aquí debemos especificar la ruta a la que Stripe debe mandar los eventos seleccionados cuando se produzcan. Estos son: la cancelación de una suscripción, un pago exitoso y un pago fallido. Stripe asocia a cada punto de conexión creado un clave de firma que se emplean en el manejador de eventos para verificar las llamadas entrantes y tener una mayor seguridad.

Las claves obtenidas de los pasos anteriores se deben guardar de forma segura para que no estén expuestas en el código fuente de la aplicación, y por tanto, que no puedan ser accedidas

desde el repositorio público en GitHub. Para ello, empleamos la característica de la versión 5.2 de Rails, el fichero *credentials.yml.enc*, que está destinado a almacenar estas claves [38].

La principal ventaja de este fichero es que está encriptado y solo se puede ser desencriptado si contamos con la clave que se encuentra en el fichero *master.key* el cual se ignora con el fichero *.gitignore* de nuestro repositorio. Por este motivo, se debe incorporar manualmente como una variable de entorno en el contenedor de Heroku bajo el nombre de *RAILS_MASTER_KEY*. Este nombre permite que al iniciar la aplicación Rails busque y acceda a dicho archivo. Para acceder desde el código fuente, se emplea el método *Rails.application.credentials*. Un ejemplo sería:

Rails.application.credentials[:production][:service][:key] donde :service es el servicio (aws, stripe) y key es el nombre de una clave específica. No obstante, todo depende de cómo esté configurado el archivo *credentials.yml.enc*.

Otro aspecto importante es que Active Storage y Carrierwave requieren que una serie de herramientas (o programas) estén instaladas en el entorno en donde se ejecuta la aplicación. En el entorno de desarrollo éstas se deben instalar a mano mientras que en Heroku se deben aprovisionar mediante *buildpacks*.

Para el redimensionamiento de imágenes se necesita tener instalado ImageMagick a la que se accede a través de la interfaz Ruby implementada por la gema mini_magick. Afortunadamente, cuando se crea un contenedor con el lenguaje de programación Ruby, Heroku aprovisiona automáticamente este recurso. Sin embargo, para la previsualización de videos y documentos en formato PDF, que es una de las características por defecto de Active Storage, es necesario tener instaladas las herramientas FFmpeg y muPDF las cuáles no vienen por defecto. Por lo tanto, debemos aprovisionarlas a mano mediante un *buildpack* que se encarga de instalarlas en el contenedor específico [39].

Etapa 3: Despliegue

En esta cuarta y última etapa, se efectúa el despliegue de la aplicación. Este proceso consiste transferir el repositorio nuestro repositorio local al repositorio remoto que se encuentra enlazado con el contenedor de Heroku, ejecutar las migraciones en la base de datos de PostgreSQL y poblar esta con información de ejemplo, manteniendo la sincronización con Stripe, mediante la tarea *users:create* creada para tal fin.

Resultados

El resultado de este proceso es un prototipo completamente desplegado y funcionando en un entorno de producción. Para acceder a él se puede acudir al siguiente enlace:

https://box2share.herokuapp.com/

8. Conclusiones

8.1. Consecución de los objetivos

El resultado obtenido con la realización de este Trabajo de Fin de Título permite cumplir con los objetivos que se establecieron, tanto a nivel de proyecto como a nivel académico, al inicio de este.

A nivel de proyecto, el objetivo general que se propuso fue desarrollar un prototipo que permitiese administrar un *bucket* de Amazon S3 de manera similar a un servicio de almacenamiento en la nube como los existentes en el mercado. Para alcanzar este objetivo se marcaron una serie de objetivos más específicos que se debían lograr durante la fase de desarrollo definida en nuestro plan de trabajo inicial y que se cumplen en el prototipo resultante.

A nivel académico, se propusieron una serie de objetivos que se han alcanzado en las distintas fases definidas en el plan de trabajo inicial:

- Durante la fase de estudio previo/análisis y la fase de desarrollo se aprendió a desarrollar una aplicación web desde cero con el *framework* Ruby on Rails y en el lenguaje de programación Ruby.

- Durante todas las fases se pusieron en práctica los conocimientos teóricos y prácticos adquiridos durante los estudios universitarios del Grado de Ingeniería Informática de la Universidad de Las Palmas de Gran Canaria los cuáles han permitido superar todas las situaciones a las que nos hemos enfrentado.

- Durante la fase de desarrollo se aprendió a utilizar el servicio Amazon S3 de la plataforma Amazon Web Services. En concreto, sus características básicas como la creación de un bucket, la configuración para acceder a él y su integración con el prototipo empleando librerías de terceros ampliamente usadas por la comunidad.

- Durante la fase de despliegue se desplegó el prototipo en un entorno de producción mediante Heroku. Esto sirve como primera toma de contacto con el mundo profesional donde el despliegue continuo está cobrando una gran importancia para entregar valor cuanto antes a los clientes.

8.2. Trabajo futuro

En la sección de Alcance de la implementación del apartado de Desarrollo se mencionó que el prototipo no implementaría todas las funcionalidades planteadas en el apartado de Análisis debido a que la dedicación, recursos y tiempo que requerían excedía el tiempo disponible para este trabajo.

En esta sección se presentan una serie de líneas de trabajo que se podrían seguir de cara a obtener un primera versión de la aplicación. No obstante, algunas podrían dejarse para versiones posteriores. Es importante destacar que se incluyen funcionalidades que no se contemplaron en el análisis inicialmente y que aportarían valor a esta aplicación.

Estas líneas de trabajo se dividen en tres grupos: las destinadas a mejorar la aplicación, las destinadas a completar la aplicación y las destinadas a ampliar la aplicación con nuevas funcionalidades.

En primer lugar, algunas de las líneas de trabajo destinadas a mejorar la aplicación serían:

- Emplear el *framework* React en el lado del cliente manteniendo el *framework* Ruby on Rails en el lado del servidor pero actuando como una API REST. Si bien esto supondría redefinir la arquitectura establecida, nos aportaría la ventaja de poder desarrollar una aplicación de una sola página. Esto nos facilitaría mejorar las funcionalidades existentes e implementar las nuevas funcionalidades que suponen una lógica de negocio mucho mayor. Para esto, se podría emplear la gema React on Rails que permite integrar fácilmente ambas tecnologías.

- Añadir más estadísticas en la vista del panel de control. De cara al usuario administrador, se podría incluir las estadísticas que se muestran en el panel de administración de Stripe.

- Añadir el soporte para la traducción a varios idiomas mediante la gema rails-i18n.

- Realizar una batería de pruebas profesional con el *framework* RSpec. El principal hito aquí sería probar toda la integración de Stripe en la aplicación con la gema stripe-ruby-mock.

En segundo lugar, la principal línea de trabajo destinada a completar la aplicación sería implementar todas las funcionalidades que no se llevaron a cabo en este trabajo y que se marcan con color rojo en la Tabla 2-7. Si partiésemos de la base de que ya contamos con el *framework* React en el lado del cliente, la mayoría de estas funcionalidades serían menos complejas de implementar. Por ejemplo, de cara a desarrollar la función de subir una carpeta o subir varios archivos, es necesario emplear JavaScript. El uso de componentes que existen para este *framework* podría facilitar esta tarea. Esto mismo se puede aplicar al resto de funciones.

En tercer y último lugar, algunas de las líneas de trabajo más importantes destinadas a ampliar el prototipo con nuevas funcionalidades serían:

- Integrar todo el soporte que ofrece Stripe para implementar todo el modelo de suscripciones [40].

- Ampliar el panel de administrador del usuario administrador apoyándonos en la API de Stripe para permitir manejar todo desde él. Por ejemplo, permitir gestionar los usuarios de manera completa, los planes, las suscripciones, etc. También se podrían contemplar nuevos roles como un usuario super administrador que pudiese crear usuarios administradores y asignar permisos. Para ello, nos podemos apoyar en la gema cancancan que permite implementar un sistema de autorización basado en roles.

- Añadir más características a los planes de pago de la aplicación, por ejemplo, límite en la carga de archivos, número máximo de archivos, excluirle del sistema de publicidad para usuarios suscritos a planes gratuitos, aplicar fecha de caducidad a los enlaces, etc.

- Implementar un sistema de notificaciones similar para que los usuarios puedan recibir avisos en vivo si alguien les ha compartido un archivo o carpeta, si el pago de su suscripción es exitoso o no, etc. Para el usuario administrador valdría para ser notificado de problemas que ocurran, de solicitudes de los usuarios, etc. En el mercado contamos con gemas como activity_notification y notifications además de la característica de Action Cable de Ruby on Rails que nos darían la posibilidad de hacerlo fácilmente.

8.3. Valoración personal

Para finalizar debo decir que este Trabajo de Fin de Título ha supuesto el mayor reto al que me he enfrentado, tanto a nivel personal como a nivel académico, desde que comencé esta andadura hace cuatro años.

Por un lado, he tenido la oportunidad de aprender a desarrollar una aplicación desde cero en el *framework* Ruby on Rails y en el lenguaje de programación Ruby. Aunque en un principio tuve

dificultades por ser una tecnología y lenguaje con los que no estaba acostumbrado a trabajar, a medida que iba descubriendo algunas de las características que ofrecen y me familiarizaba con ellos, resultaban cada vez más de mi agrado. Durante su aprendizaje me han surgido ideas que antes ni se me pasaban por la cabeza y que con Rails y Ruby pueden ser posibles fácilmente. Todo esto me anima a seguir aprendiendo más con ellos y a emplearlos en mis futuros proyectos personales.

Por otro lado, he podido enfrentarme a problemas, inconvenientes, cambios inesperados, planteamientos que al final no se podían aplicar, o que con una gema se podía hacer de forma rápida. Ante estas situaciones ha surgido la necesidad de leer código de otros desarrolladores o buscar soluciones en las distintas comunidades como Stack Overflow las cuáles son tareas que tarde o temprano deberé realizar como desarrollador ya sea en el mundo profesional o de manera independiente.

En definitiva, este trabajo ha permitido ver como el esfuerzo día a día a lo largo de la carrera ha dado sus frutos. Con él terminar esta gran etapa. Toda esta valiosa experiencia que me llevo servirá como punto de partida para lo que está por venir.

9. Referencias

[1] M. Rouse, «What is Amazon S3?,» Noviembre 2018. [En línea]. Available: https://searchaws.techtarget.com/definition/Amazon-Simple-Storage-Service-Amazon-S3. [Último acceso: 11 Junio 2019].

[2] TIC Portal, «Amazon S3: almacenamiento ¿seguro? en la nube,» [En línea]. Available: https://www.ticportal.es/temas/cloud-computing/amazon-web-services/amazon-s3. [Último acceso: 11 Junio 2019].

[3] Amazon Web Services, «¿Qué es Amazon S3? - Amazon Simple Storage Service,» 1 Marzo 2006. [En línea]. Available: https://docs.aws.amazon.com/es_es/AmazonS3/latest/dev/Welcome.html. [Último acceso: 11 Junio 2019].

[4] Proyectos Ágiles, «Proyectos Ágiles,» [En línea]. Available: https://proyectosagiles.org/desarrollo-iterativo-incremental/. [Último acceso: 31 Mayo 2019].

[5] I. Sommerville, Ingeniería del Software 9ª Edición, 9 ed., Pearson, 2011, pp. 83-90.

[6] uml-diagrams.org, «UML Use Case Diagrams,» [En línea]. Available: https://www.uml-diagrams.org/use-case-diagrams.html. [Último acceso: 12 Junio 2019].

[7] Colaboradores de Wikipedia, «Caso de uso - Wikipedia, la enciclopedia libre,» [En línea]. Available: https://es.wikipedia.org/wiki/Caso_de_uso. [Último acceso: 12 Junio 2019].

[8] R. Peiró, «Economipedia,» [En línea]. Available: economipedia.com/definiciones/modelo-de-negocio.html. [Último acceso: 01 05 2019].

[9] M. Lauciute, «Qué es PCI DSS y por qué es imprescindible para tu eCommerce,» 19 Septiembre 2018. [En línea]. Available: https://marketing4ecommerce.net/que-es-pci-dss-y-por-que-es-necesario-para-tu-ecommerce/. [Último acceso: 6 Junio 2019].

[10] C. d. Wikipedia, «Ley Orgánica de Protección de Datos Personales y garantía de los derechos digitales,» [En línea]. Available: https://es.wikipedia.org/wiki/Ley_Orgánica_de_Protección_de_Datos_Personales_y_garantía_de_los_derechos_digitales. [Último acceso: 4 Junio 2019].

[11] PymesWorld, «Cómo cumplir la ley LOPD, RGPD y la LSSI en Páginas web y Blogs. Cómo tener una web legal.,» 20 Mayo 2018. [En línea]. Available: https://pymesworld.com/cumplir-ley-lopd-lssi-pagina-web-blog/#Publicar_los_textos_legales_para_cumplir_la_LSSI. [Último acceso: 6 Junio 2019].

[12] C. d. Wikipedia, «Reglamento General de Protección de Datos,» [En línea]. Available: https://es.wikipedia.org/wiki/Reglamento_General_de_Protección_de_Datos. [Último acceso: 6 Junio 2019].

[13] M. Dahn, «Guía para cumplir con la normativa PCI,» [En línea]. Available: https://stripe.com/es/guides/pci-compliance. [Último acceso: 4 Junio 2019].

[14] S. Gómez, «Cumplir con la RGPD, la LOPD y LSSI – Requisitos legales para tu web o tienda online,» [En línea]. Available: https://lawebdetuvida.com/requisitos-legales-cumplir-lopd-lssi/. [Último acceso: 04 Junio 2019].

[15] Colaboradores de Wikipedia, «Modelo Vista Controlador,» [En línea]. Available: https://es.wikipedia.org/wiki/Modelo–vista–controlador. [Último acceso: 19 Junio 2019].

[16] J. A. Camarasa, «¿Qué aporta MVC al desarrollo de aplicaciones web?,» 14 Mayo 2012. [En línea]. Available: https://www.clavei.es/blog/que-aporta-mvc-al-desarrollo-de-aplicaciones-web/. [Último acceso: 19 Junio 2019].

[17] Colaboradores de Ruby on Rails, «Active Record Basics,» [En línea]. Available: https://guides.rubyonrails.org/active_record_basics.html. [Último acceso: 19 Junio 2019].

[18] Colaboradores de Ruby on Rails, «Action View Overview,» [En línea]. Available: https://guides.rubyonrails.org/action_view_overview.html. [Último acceso: 19 Junio 2019].

[19] Colaboradores de Ruby on Rails, «Action Controller Overview,» [En línea]. Available: https://guides.rubyonrails.org/action_controller_overview.html. [Último acceso: 19 Junio 2019].

[20] M. Hartl, «Ruby on Rails Tutorial,» [En línea]. Available: https://www.railstutorial.org/book/toy_app#sec-mvc_in_action. [Último acceso: 19 Junio 2019].

[21] Colaboradores de Ruby on Rails, «Getting Started with Rails,» [En línea]. Available: https://guides.rubyonrails.org/getting_started.html#creating-the-blog-application. [Último acceso: 20 Junio 2019].

[22] Voormedia, «Rails ERD - Entity-Relationship Diagrams for Ruby on Rails,» [En línea]. Available: https://voormedia.github.io/rails-erd/. [Último acceso: 22 Junio 2019].

[23] Colaboradores de Wikipedia, «JavaScript,» [En línea]. Available: https://es.wikipedia.org/wiki/JavaScript. [Último acceso: 13 Junio 2019].

[24] Stripe, «Stripe.js and Elements,» [En línea]. Available: https://stripe.com/docs/stripe-js. [Último acceso: 13 Junio 2019].

[25] J. García, «¿Qué es Ruby? | OpenWebinars,» 20 Octubre 2017. [En línea]. Available: https://openwebinars.net/blog/que-es-ruby/. [Último acceso: 13 Junio 2019].

[26] Wikipedia, «Ruby on Rails,» [En línea]. Available: https://es.wikipedia.org/wiki/Ruby_on_Rails. [Último acceso: 13 Junio 2019].

[27] Colaboradores de Wikipedia, «jQuery,» [En línea]. Available: https://es.wikipedia.org/wiki/JQuery. [Último acceso: 13 Junio 2019].

[28] Colaboradores de Wikipedia, «HTML,» [En línea]. Available: https://es.wikipedia.org/wiki/HTML. [Último acceso: 13 Junio 2019].

[29] J. Castello, «Ruby Templating Engines: ERB, HAML & Slim,» Noviembre 2018. [En línea]. Available: https://www.rubyguides.com/2018/11/ruby-erb-haml-slim/. [Último acceso: 13 Junio 2019].

[30] Colaboradores de Wikipedia, «Hoja de estilos en cascada,» [En línea]. Available: https://es.wikipedia.org/wiki/Hoja_de_estilos_en_cascada. [Último acceso: 13 Junio 2019].

[31] Colaboradores de Wikipedia, «Bootstrap (framework),» [En línea]. Available: https://es.wikipedia.org/wiki/Bootstrap_(framework). [Último acceso: 13 Junio 2019].

[32] Colaboradores de la Wikipedia, «Heroku,» [En línea]. Available: https://es.wikipedia.org/wiki/Heroku. [Último acceso: 13 Junio 2019].

[33] Colaboradores de Wikipedia, «Sublime Text,» [En línea]. Available: https://en.wikipedia.org/wiki/Sublime_Text. [Último acceso: 13 Junio 2019].

[34] Colaboradores de Wikipedia, «GitHub,» [En línea]. Available: https://es.wikipedia.org/wiki/GitHub. [Último acceso: 13 Junio 2019].

[35] Colaboradores de Wikipedia, «StarUML,» [En línea]. Available: https://en.wikipedia.org/wiki/StarUML. [Último acceso: 13 Junio 2019].

[36] Heroku, «Getting Started on Heroku with Rails 5.x,» 22 Febrero 2019. [En línea]. Available: https://devcenter.heroku.com/articles/getting-started-with-rails5. [Último acceso: 14 Junio 2019].

[37] Heroku, «The Heroku CLI,» 6 Mayo 2019. [En línea]. Available: devcenter.heroku.com/articles/heroku-cli. [Último acceso: 14 Junio 2019].

[38] J. Mines, «Hiding Your Secrets in Rails 5 Using Credentials,» 16 Septiembre 2018. [En línea]. Available: https://medium.com/@jonathanmines/hiding-your-secrets-in-rails-5-using-credentials-e37174eede99. [Último acceso: 14 Junio 2019].

[39] Heroku, «Active Storage on Heroku,» 8 Mayo 2018. [En línea]. Available: https://devcenter.heroku.com/articles/active-storage-on-heroku. [Último acceso: 14 Junio 2019].

[40] Stripe, «Subscription Lifecycle and Events | Stripe Billing,» [En línea]. Available: https://stripe.com/docs/billing/lifecycle. [Último acceso: 16 Junioo 2019].

10. Anexos

10.1. Diagramas de casos de uso

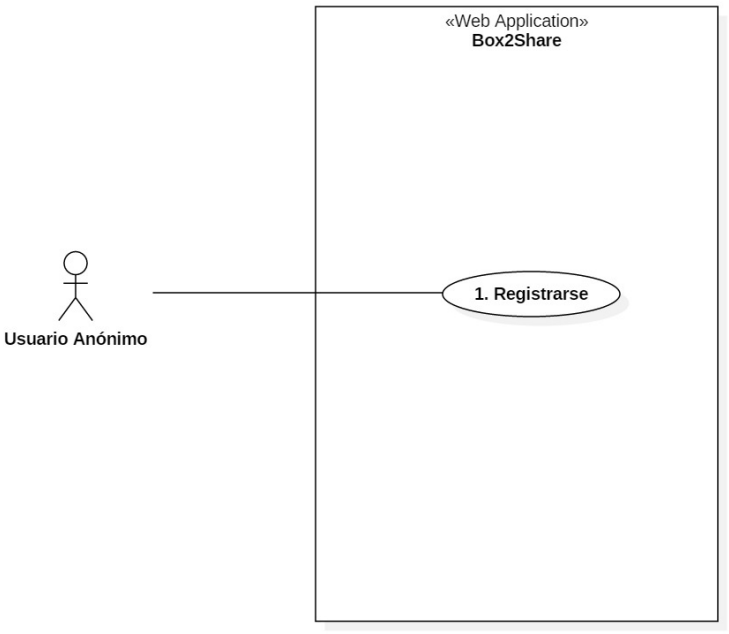

Figura 10-1. Diagrama de casos de uso del usuario anónimo

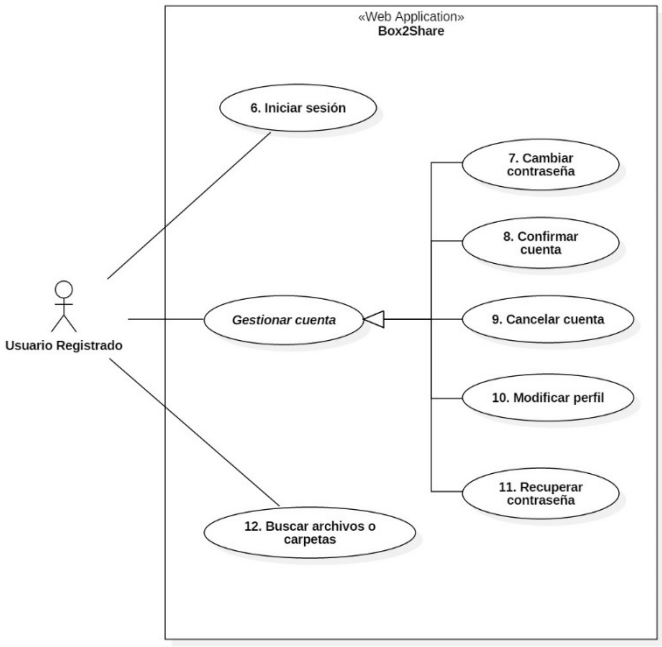

Figura 10-2. Diagrama de casos de uso del usuario registrado (parte 1)

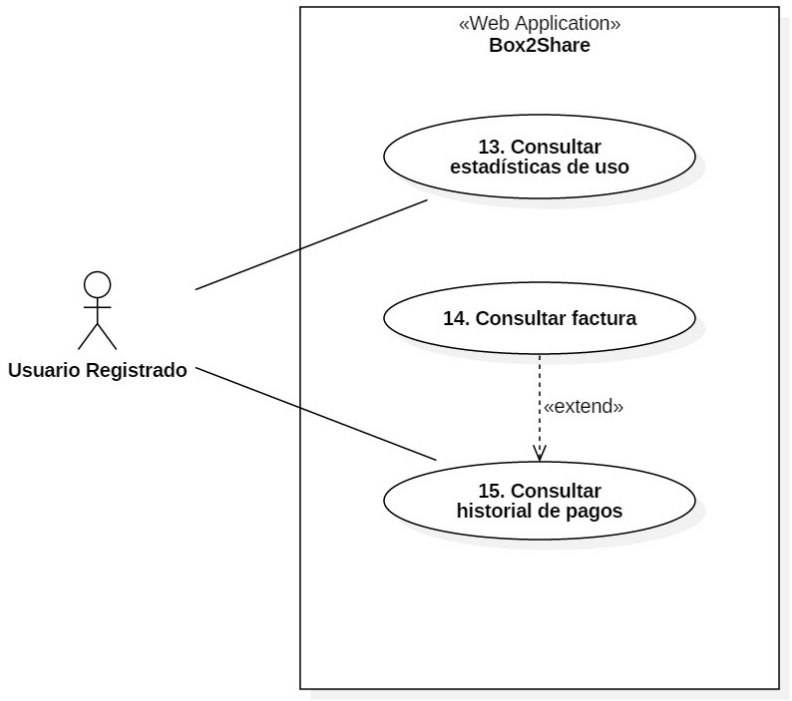

Figura 10-3. Diagrama de casos de uso del usuario registrado (parte 2)

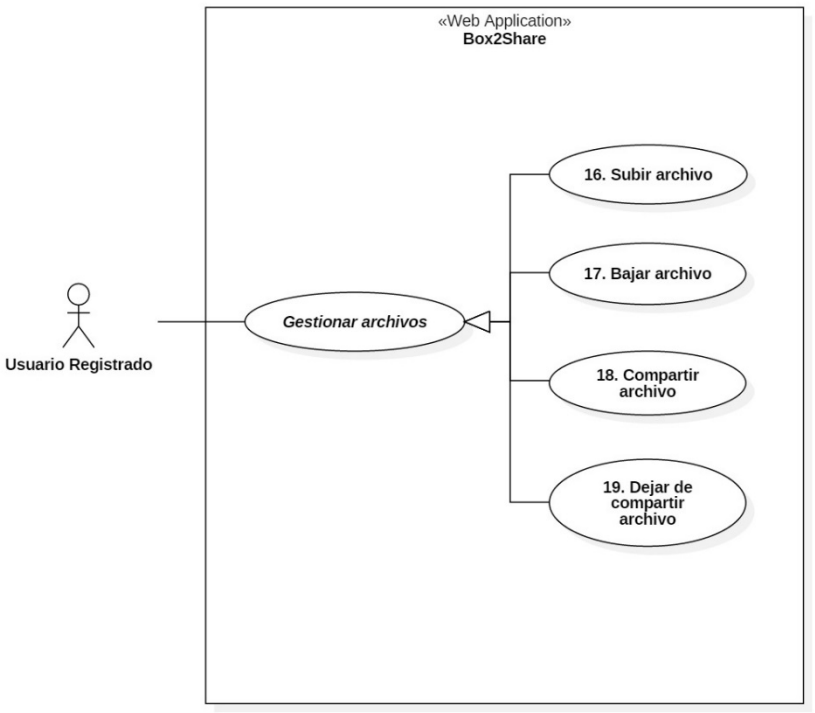

Figura 10-4. Diagrama de casos de uso del usuario registrado (parte 3)

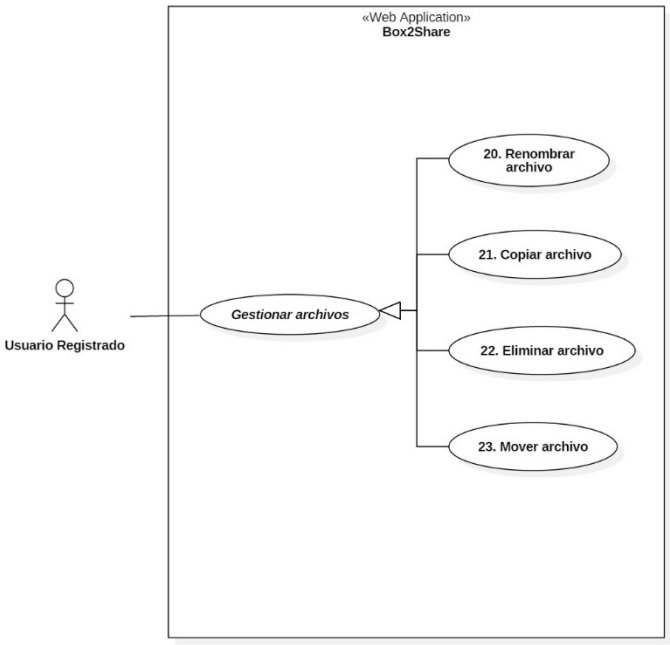

Figura 10-5. Diagrama de casos de uso del usuario registrado (parte 4)

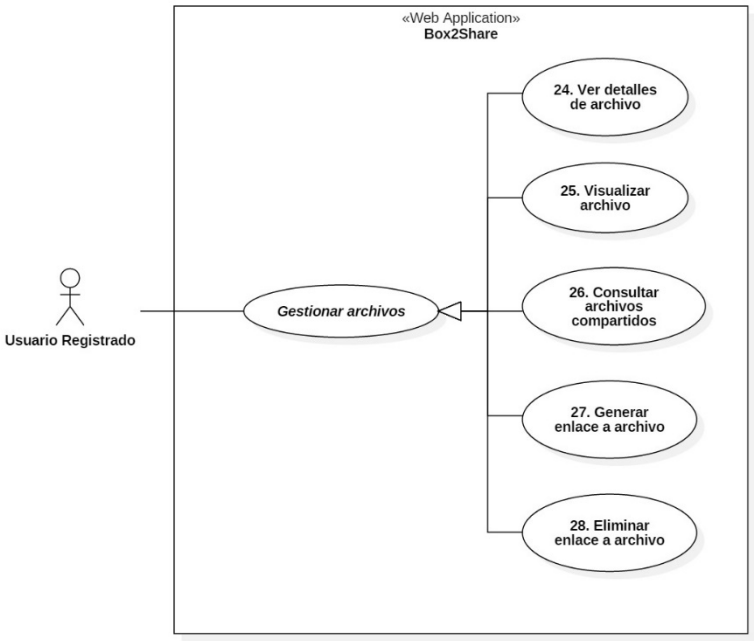

Figura 10-6. Diagrama de casos de uso del usuario registrado (parte 5)

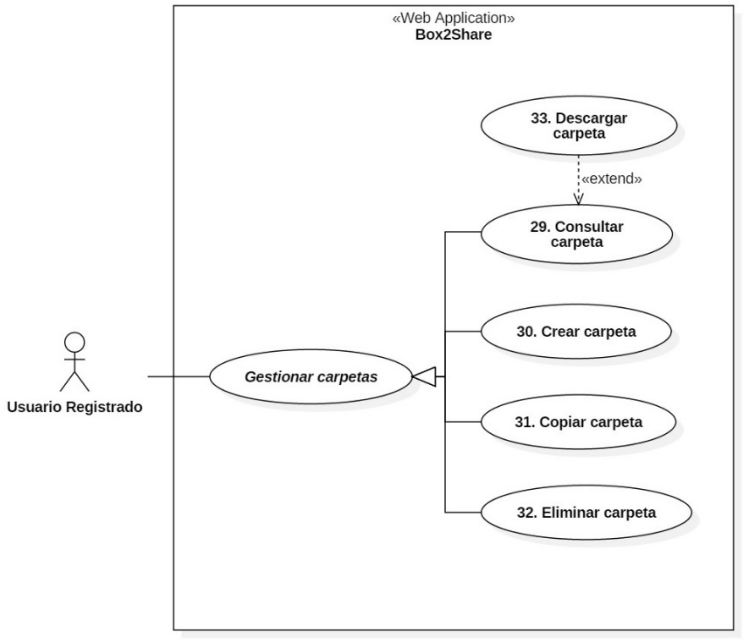

Figura 10-7. Diagrama de casos de uso del usuario registrado (parte 6)

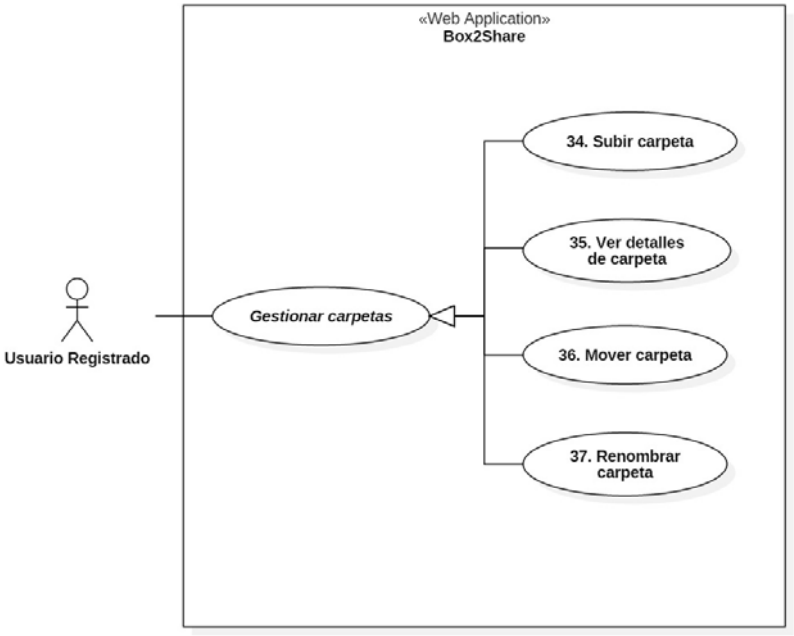

Figura 10-8. Diagrama de casos de uso del usuario registrado (parte 7)

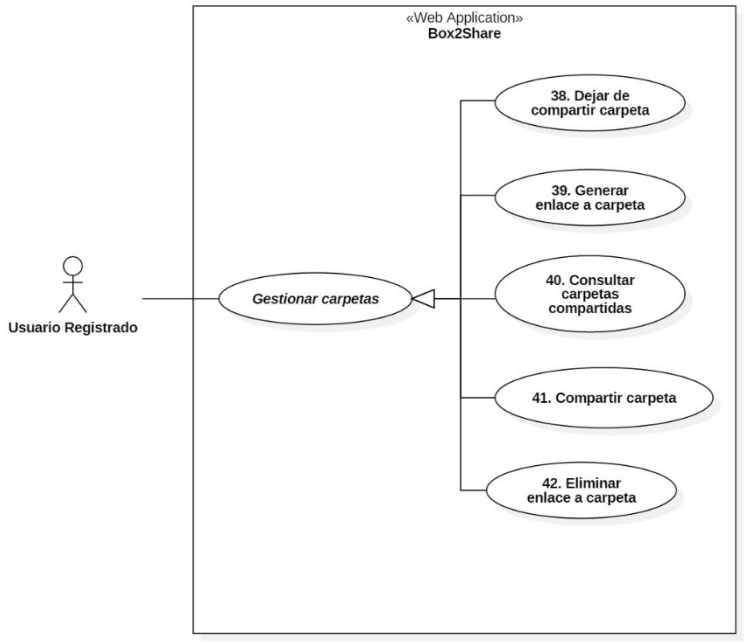

Figura 10-9. Diagrama de casos de uso del usuario registrado (parte 8)

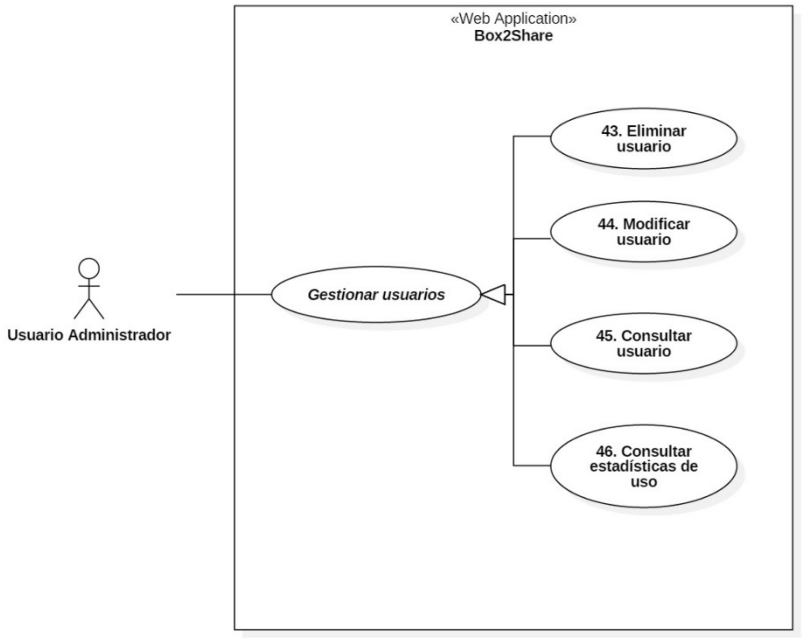

Figura 10-10. Diagrama de casos de uso del usuario administrador (parte 1)

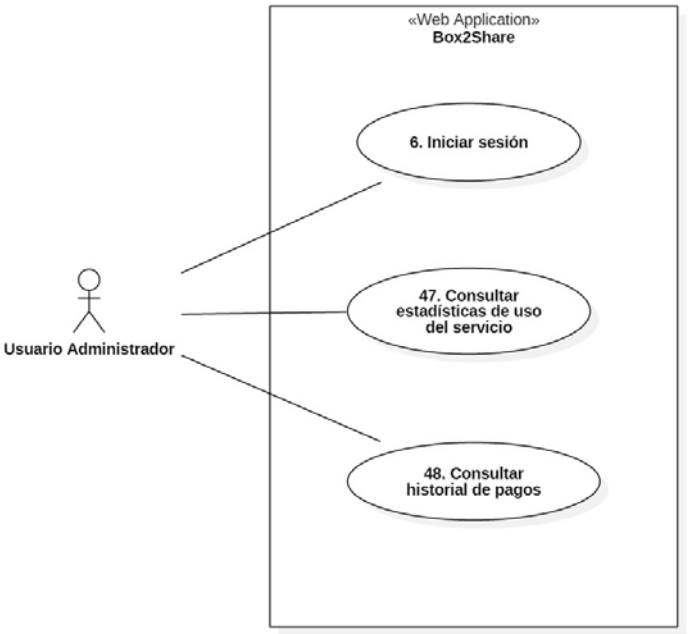

Figura 10-11. Diagrama de casos de uso del usuario administrador (parte 2)

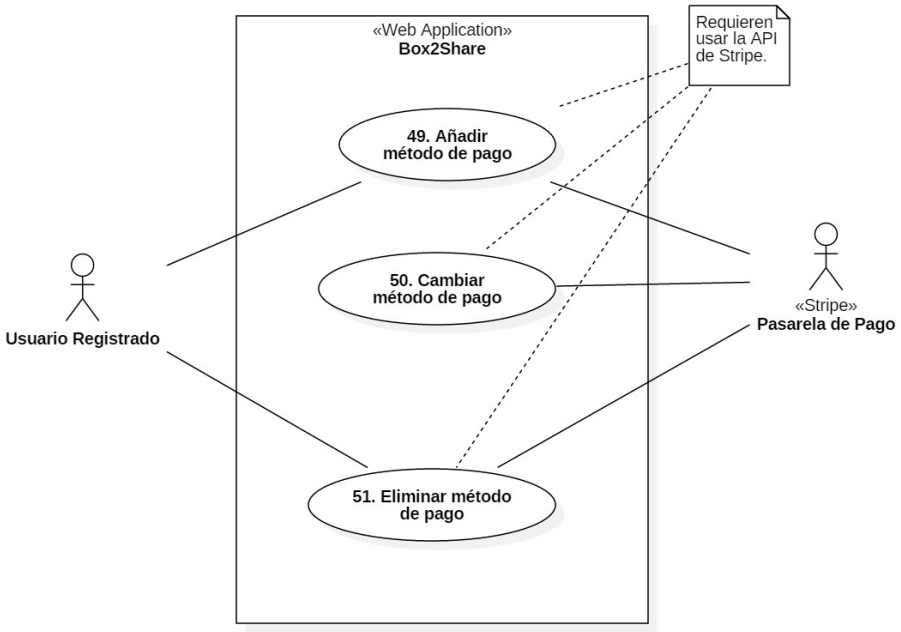

Figura 10-12. Diagrama de casos de uso del usuario registrado (parte 9)

10.2. Especificación de casos de uso

CASO DE USO	CU03	Realizar pago	
Descripción		El sistema permite a un usuario registrado realizar el pago de una suscripción a un plan de almacenamiento.	
Actores		Usuario Registrado, Pasarela de Pago	
Fecha de creación		04/04/2019	
Fecha de modificación		12/06/2019	
Precondiciones		• El usuario registrado debe haber iniciado sesión en el sistema. • El usuario registrado debe haber iniciado el proceso de adquirir una suscripción. • El usuario registrado debe encontrarse en la vista del formulario de pago.	
Flujo normal		**Paso**	**Acción**
		1	El sistema muestra al usuario registrado un formulario con los campos a rellenar con los datos de pago.
		2	El usuario realiza la cumplimentación del formulario.
		3	El usuario registrado confirma el pago de la suscripción.
		4	El sistema envía los datos a la pasarela de pago.
		5	La pasarela de pago confirma el pago de la suscripción.
Postcondiciones		• Se añade un pago al historial de pagos del usuario registrado. • Se cambia el estado del pago a exitoso.	
Variaciones		**Paso**	**Acción**
Extensiones		**Paso**	**Condición**	**Caso de Uso**
Excepciones		5	• La pasarela de pago rechaza el pago de la suscripción. • El sistema cambia el estado del pago a fallido.

	5	• No queda saldo disponible en la tarjeta de crédito. • El sistema cambia el estado del pago a fallido.
Observaciones		

Figura 10-13. Especificación del caso de uso de Realizar pago

CASO DE USO	**CU04**	**Cambiar suscripción**	
Descripción		El sistema permite a un usuario registrado cambiar su suscripción a otro plan de almacenamiento de pago.	
Actores		Usuario Registrado, Pasarela de Pago	
Fecha de creación		04/05/2019	
Fecha de modificación		12/06/2019	
Precondiciones		• El usuario registrado debe haber iniciado sesión en el sistema. • El usuario registrado debe estar suscrito a un plan de almacenamiento de pago. • El usuario registrado debe encontrarse en la vista de plan de almacenamiento actual	
Flujo normal		**Paso**	**Acción**
		1	El usuario registrado selecciona la opción de cambiar su suscripción.
		2	El sistema muestra al usuario registrado la vista con los planes de almacenamiento disponibles.
		3	El usuario registrado selecciona un plan de almacenamiento.
		4	El sistema solicita al usuario registrado que confirme si está seguro de cambiar su suscripción a este nuevo plan de almacenamiento.
		5	El usuario registrado confirma el cambio de su suscripción.
		6	El sistema cambia la suscripción en la pasarela de pago.
		7	El sistema actualiza la suscripción en la base de datos.

	8	El sistema muestra al usuario un mensaje indicándole que su suscripción ha cambiado con éxito.	
Postcondiciones	• La nueva suscripción se muestra en la vista de plan de almacenamiento actual. • Se envía un correo electrónico indicando que se ha cambiado la suscripción.		
Variaciones	**Paso**	**Acción**	
Extensiones	**Paso**	**Condición**	**Caso de Uso**
Excepciones			
Observaciones			

Figura 10-14. Especificación del caso de uso de Cambiar suscripción

CASO DE USO	**CU05**	**Cancelar suscripción**
Descripción		El sistema permite a un usuario registrado cancelar su suscripción a un plan de almacenamiento de pago.
Actores		Usuario Registrado, Pasarela de Pago
Fecha de creación		04/05/2019
Fecha de modificación		12/06/2019
Precondiciones		• El usuario registrado debe haber iniciado sesión en el sistema. • El usuario registrado debe haber estar suscrito a un plan de almacenamiento de pago. • El usuario registrado debe encontrarse en la vista de plan de almacenamiento actual.
Flujo normal		**Paso**	**Acción**
		1	El usuario registrado selecciona la opción de cancelar su suscripción.
		2	El sistema solicita al usuario que confirme la cancelación de su suscripción.
		3	El usuario registrado confirma la cancelación de su suscripción.

	4	El sistema elimina la suscripción de la pasarela de pago.
	5	El sistema cambia el estado de la suscripción a cancelada.
	6	La pasarela de pago confirma la eliminación de la suscripción.
	7	El sistema muestra al usuario un mensaje indicándole que su suscripción ha sido cancelada.
Postcondiciones	• El usuario registrado pasa a tener una suscripción al plan de almacenamiento gratuito. • El usuario registrado dispone de menos espacio de almacenamiento en su cuenta de usuario. • Se envía un correo electrónico indicando que se ha cancelado la suscripción.	
Variaciones	**Paso**	**Acción**
Extensiones	**Paso**	**Condición**	**Caso de Uso**
Excepciones		
Observaciones		

Figura 10-15. Especificación del caso de uso de Cancelar suscripción

CASO DE USO	**CU16**	**Subir archivo**
Descripción		El sistema permite a un usuario registrado subir un archivo a su cuenta de almacenamiento.
Actores		Usuario Registrado
Fecha de creación		04/05/2019
Fecha de modificación		12/06/2019
Precondiciones		• El usuario registrado debe haber iniciado sesión en el sistema. • El usuario registrado debe encontrarse en la vista del directorio raíz o en la vista de una carpeta.
Flujo normal		**Paso**	**Acción**

	1	El usuario registrado selecciona la opción de subir un archivo.	
	2	El sistema muestra al usuario registrado un formulario con el campo para subir un archivo.	
	3	El usuario registrado selecciona un archivo de su ordenador.	
	4	El usuario registrado confirma la subida del archivo.	
	5	El sistema muestra al usuario un mensaje de confirmación indicándole que se ha subido el archivo con éxito.	
Postcondiciones -	• El archivo subido aparece en el directorio raíz del panel de administración del usuario o en la carpeta. • El número de subidas se incrementa en una unidad. • El espacio de almacenamiento incrementa en el tamaño del archivo subido.		
Variaciones	**Paso**	**Acción**	
Extensiones	**Paso**	**Condición**	**Caso de Uso**
Excepciones	4	• El tamaño del archivo excede el tamaño máximo permitido. • Volver al Paso 1.	
	4	• El usuario registrado no dispone de espacio de almacenamiento suficiente en su cuenta. • Volver al Paso 1.	
Observaciones			

Figura 10-16. Especificación del caso de uso de Subir archivo

10.3. Mockups

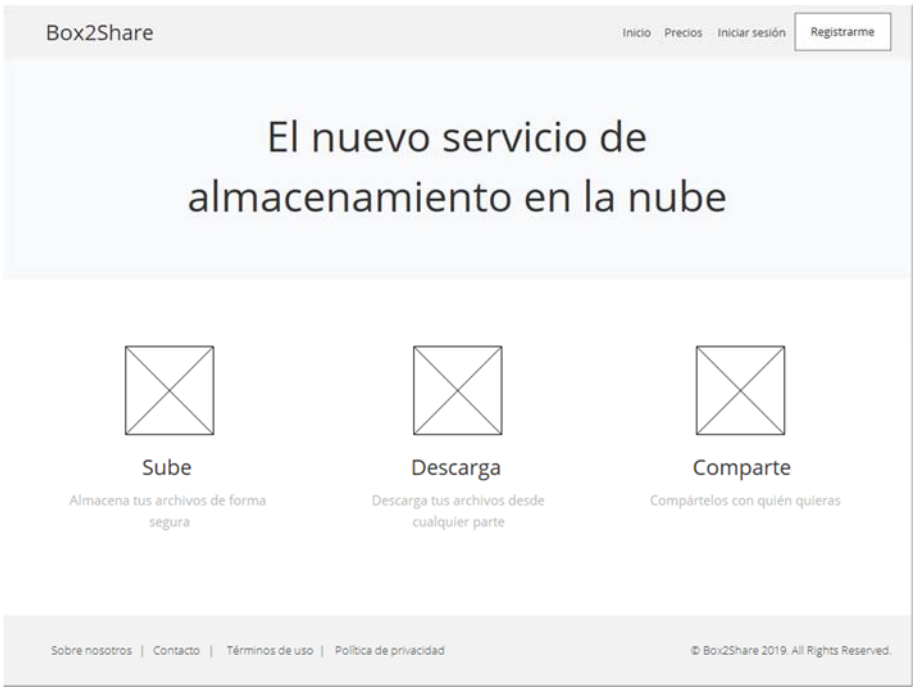

Figura 10-17. Vista de la página de inicio

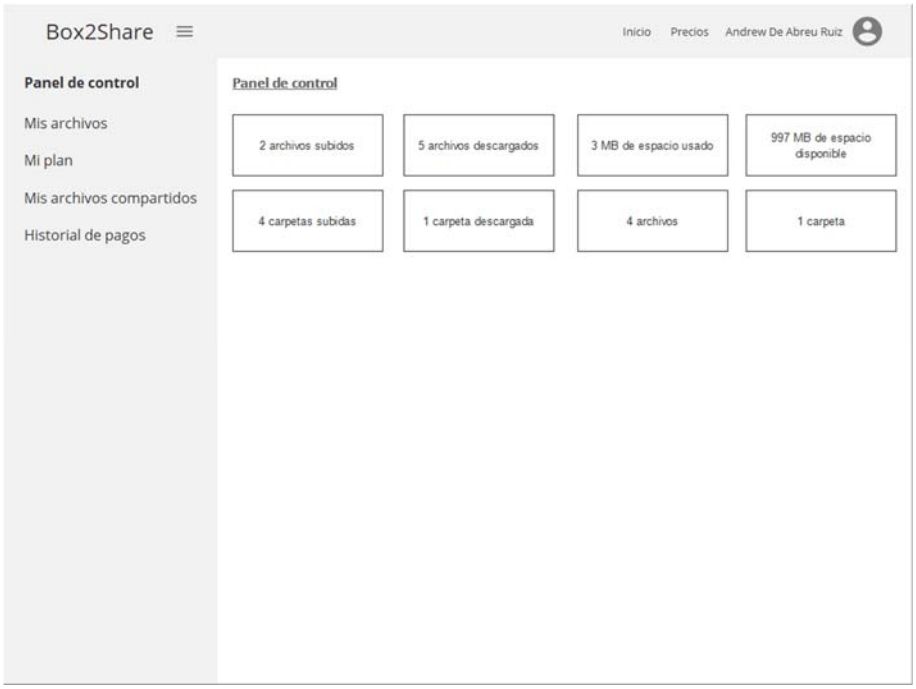

Figura 10-18. Vista de la página de inicio para un usuario cliente

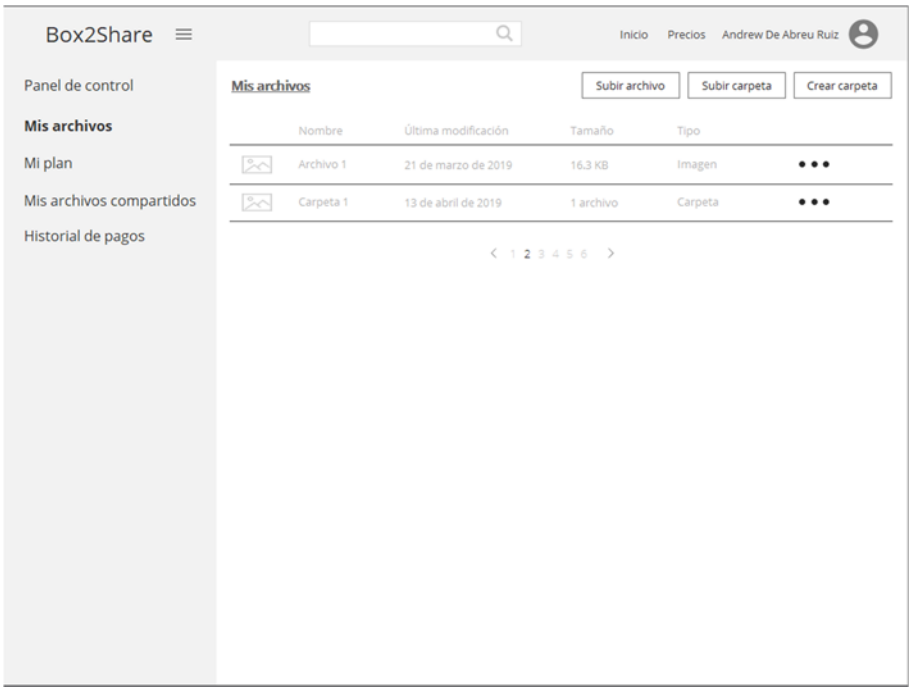

Figura 10-19. Vista del directorio raíz

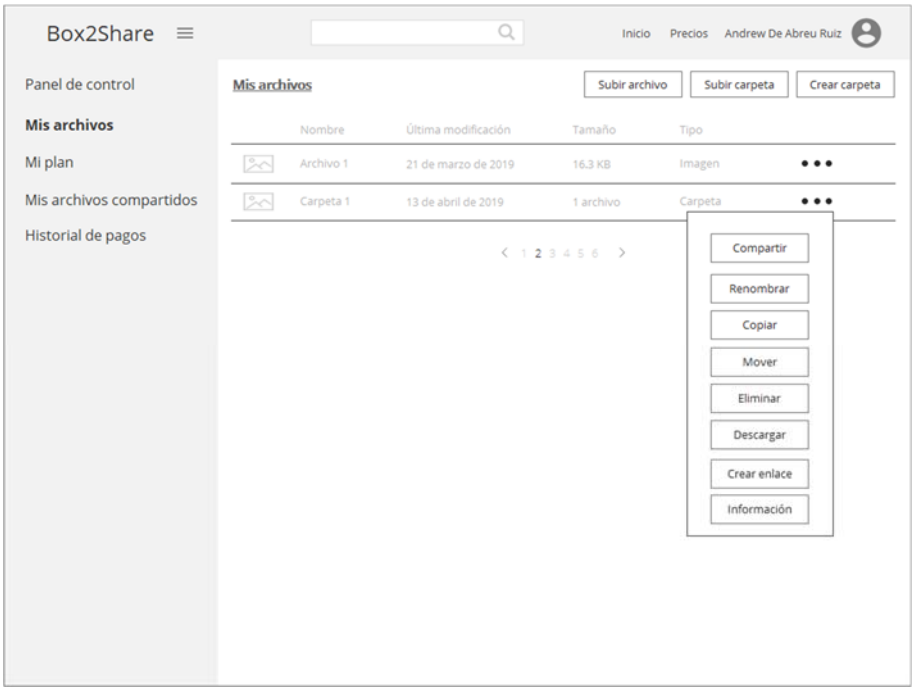

Figura 10-20. Vista del directorio raíz con la opciones para un archivo

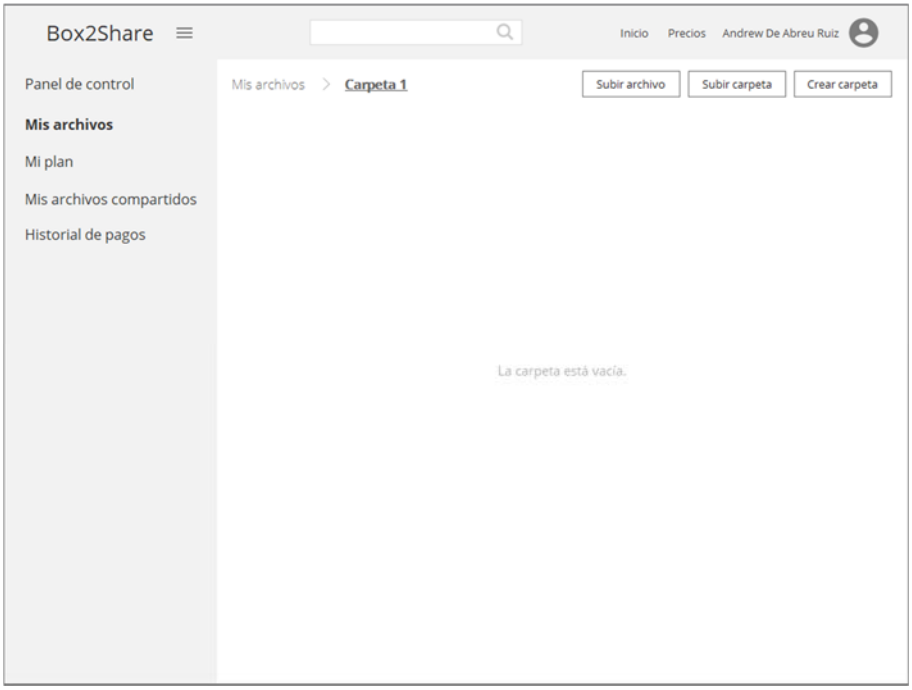

Figura 10-21. Vista de un carpeta

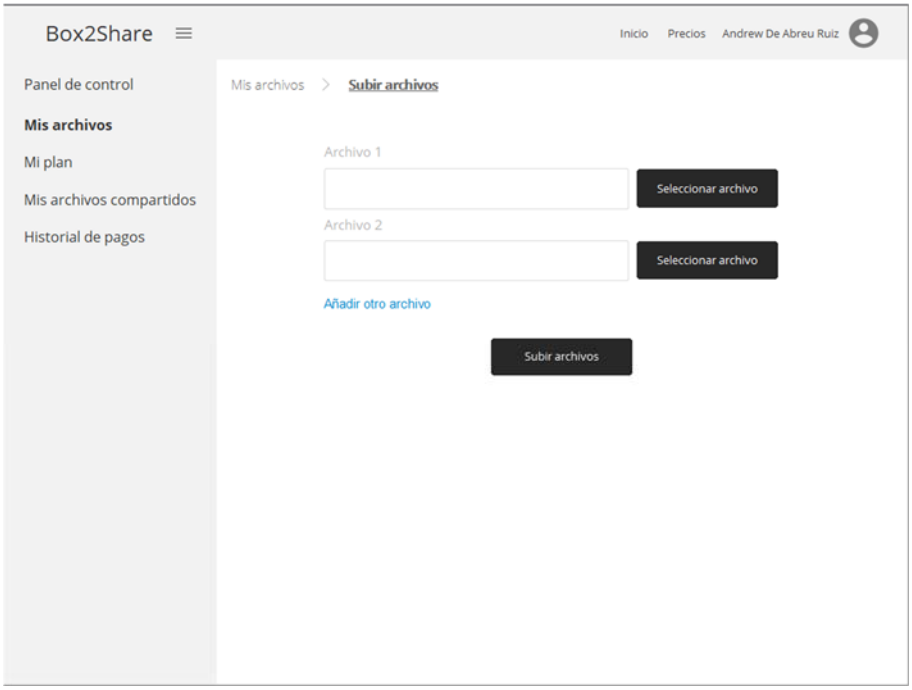

Figura 10-22. Vista del formulario para subir un archivo

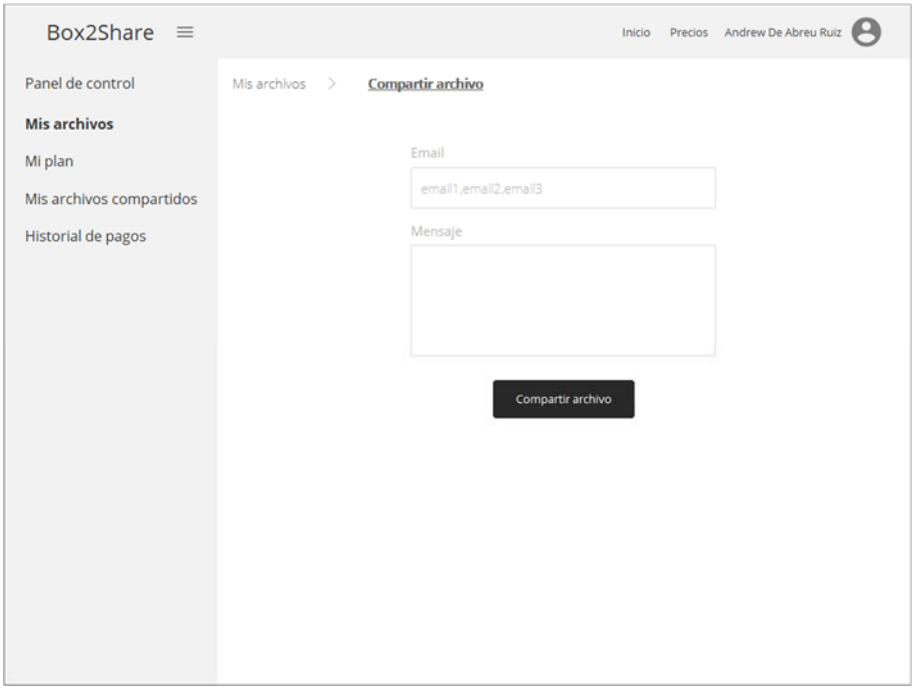

Figura 10-23. Vista del formulario para compartir un archivo

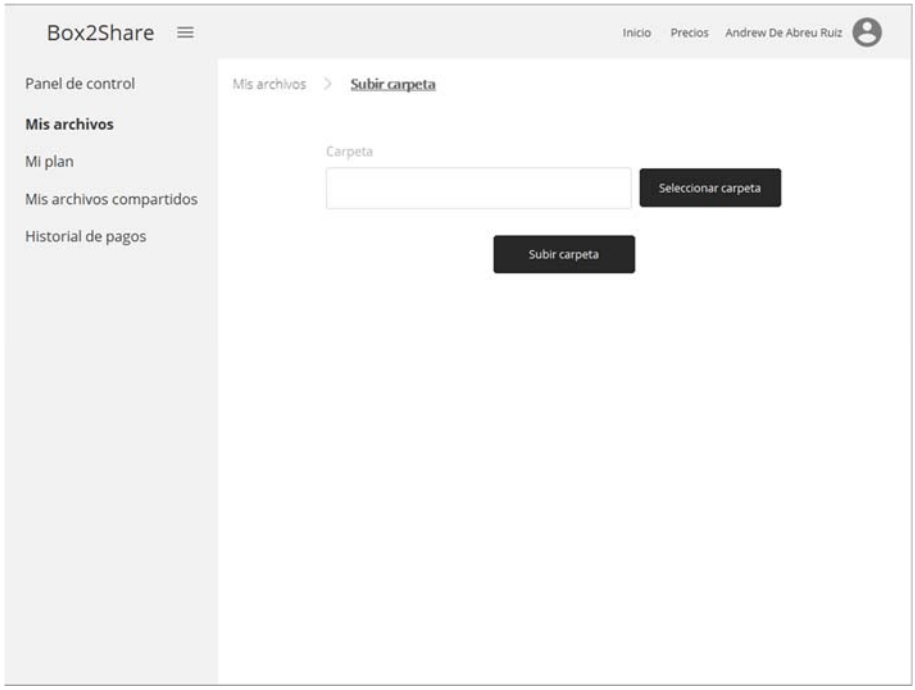

Figura 10-24. Vista del formulario para subir una carpeta

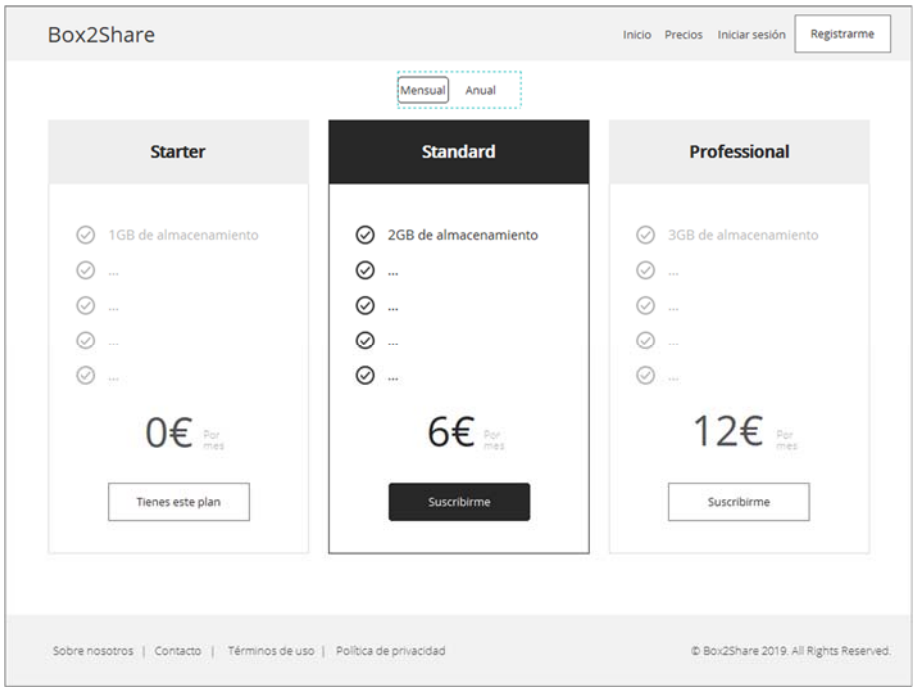

Figura 10-25. Vista de la tabla de precios

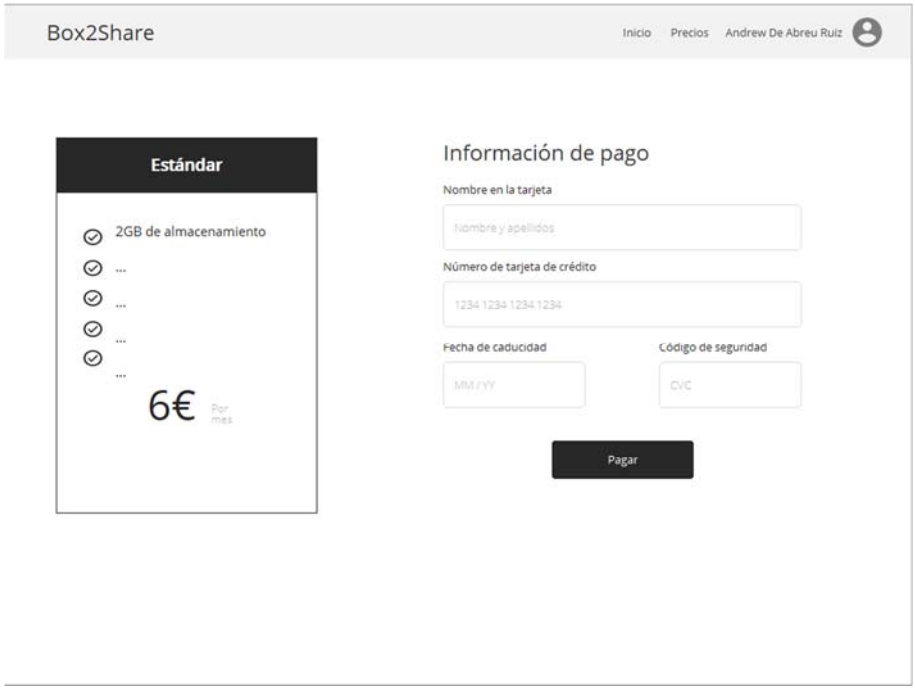

Figura 10-26. Vista del formulario de pago

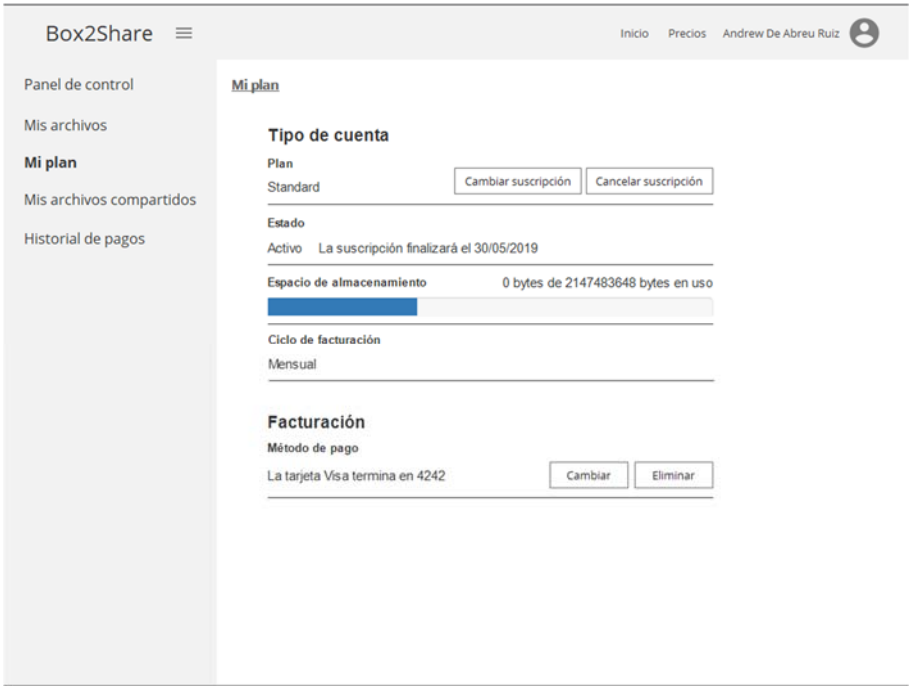

Figura 10-27. Vista del plan actual

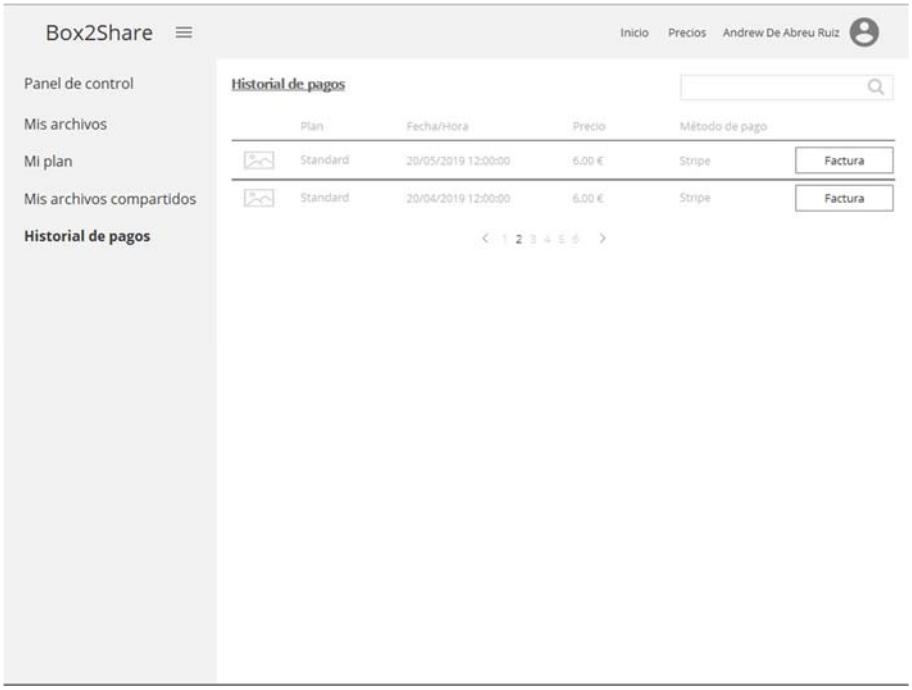

Figura 10-28. Vista del historial de pagos

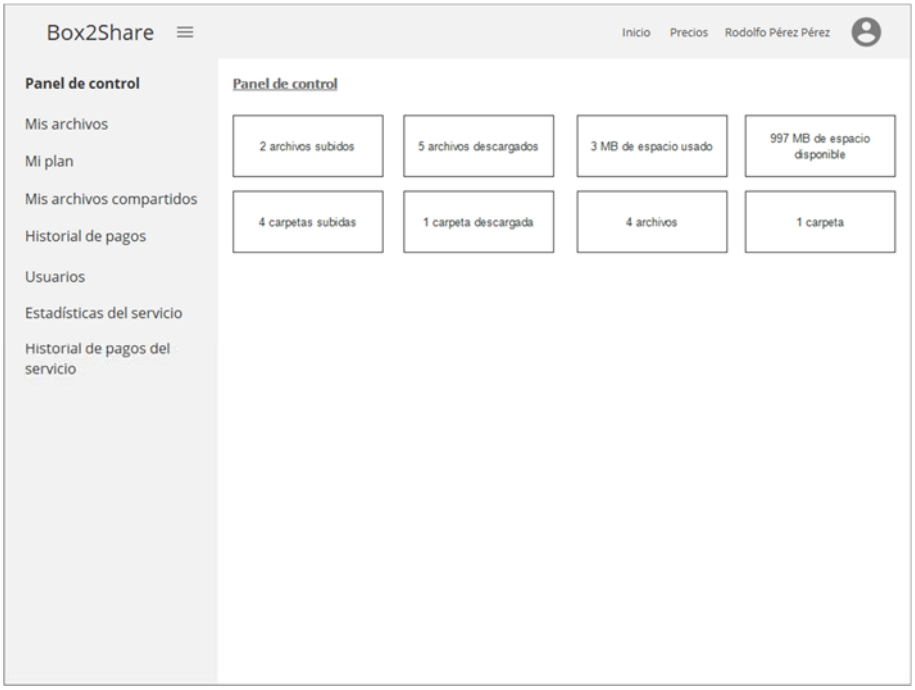

Figura 10-29. Vista de la página de inicio para un usuario administrador

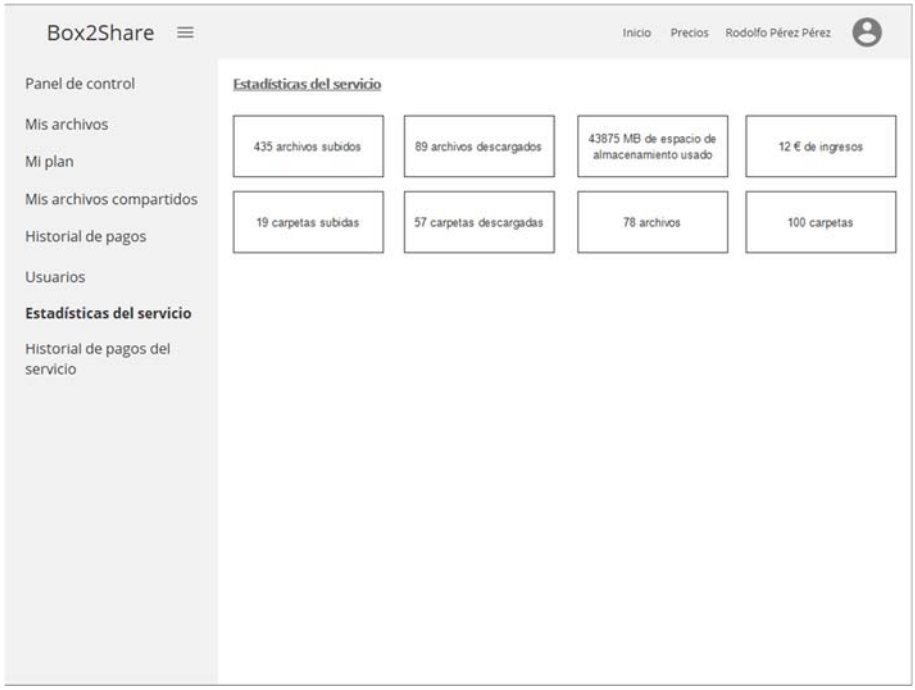

Figura 10-30. Vista de las estadísticas del servicio

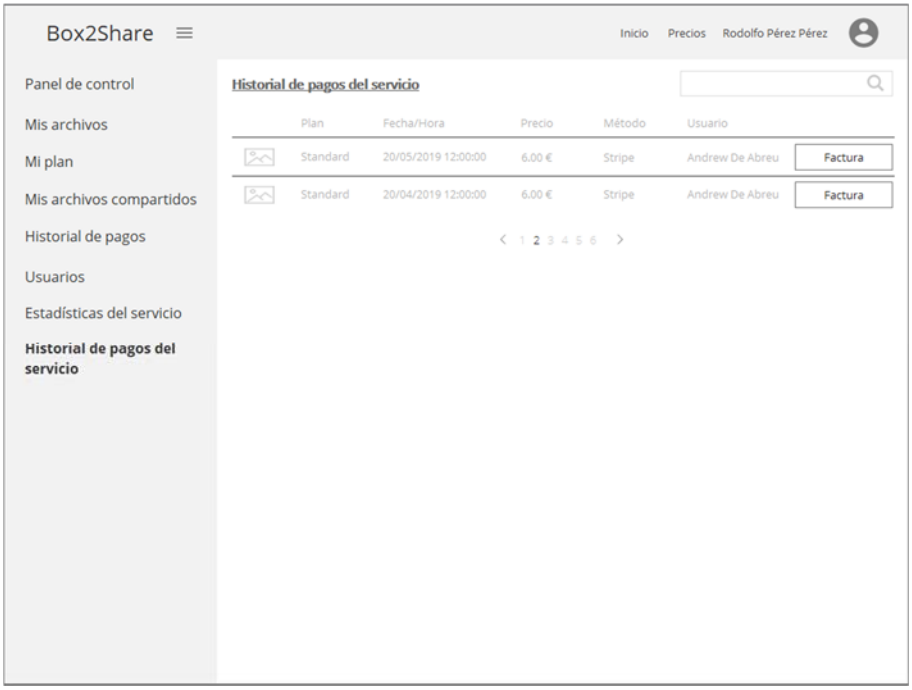

Figura 10-31. Vista del historial de pagos del servicio

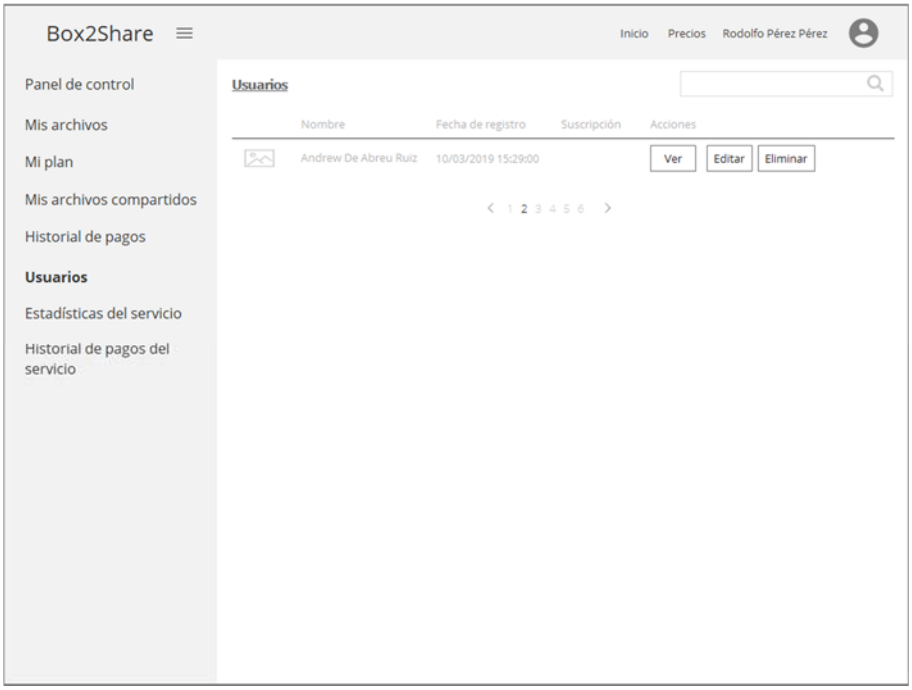

Figura 10-32. Vista de los usuarios del servicio

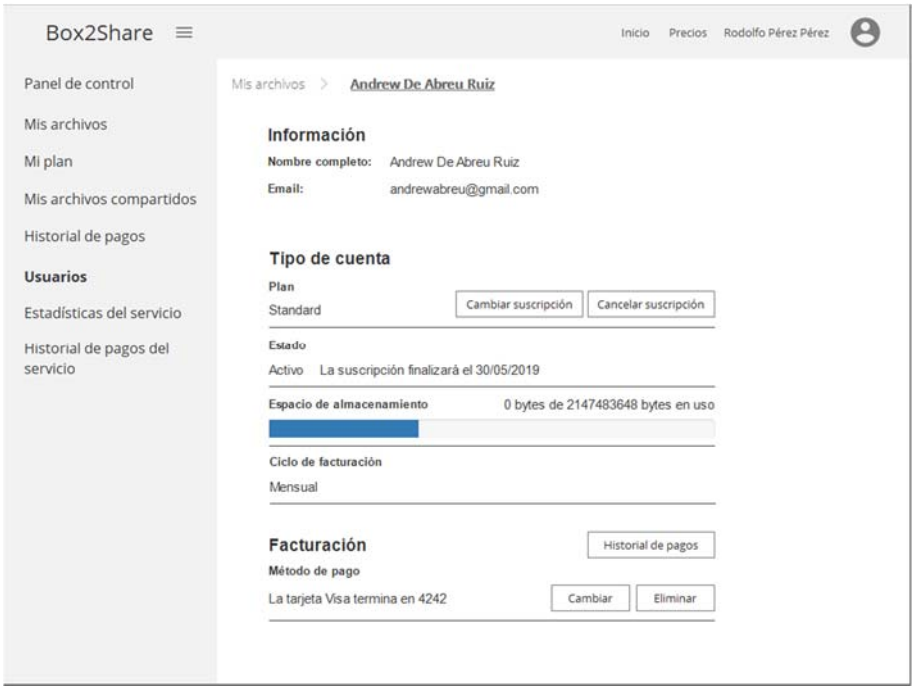

Figura 10-33. Vista de la información de un usuario del servicio

10.4. Diagrama entidad-relación completo

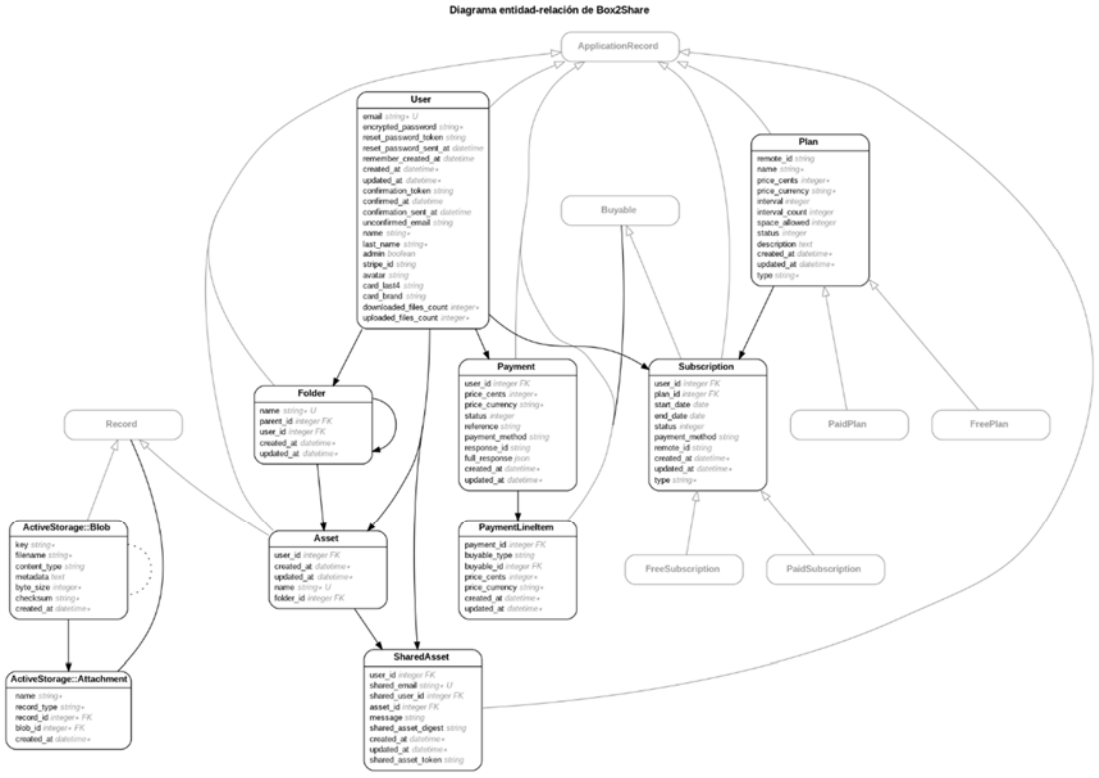

Figura 10-34. Diagrama entidad-relación completo

10.4.1. Enlace al repositorio del trabajo

El código fuente de la aplicación Ruby on Rails se encuentra alojado en un repositorio en GitHub. Se puede acceder a él a través del siguiente enlace:

https://github.com/andrewabreur/box2share

10.4.2. Manual de usuario

En esta sección se muestra el manual de usuario para hacer uso de la aplicación. A continuación, explicamos paso a paso las funcionalidades más relevantes que se pueden realizar.

A. Usuario registrado (cliente)

Panel de administración

Desde el panel de administración podemos acceder a los siguientes funcionalidades:

- Consultar el panel de control *("Panel de control")* en donde se muestran todas las estadísticas de uso de nuestra cuenta de usuario.
- Consultar el directorio raíz del sistema de archivo o carpetas *("Mis archivos")* de nuestra cuenta de usuario.
- Consultar los archivos que nos han compartido *("Compartiendo conmigo")*.
- Consultar nuestro plan actual *("Mi plan").*
- Consultar nuestro historial de pagos *("Historial de pagos").*

Adquirir una suscripción

Para adquirir una suscripción a un plan de pago debemos hacer clic en el enlace de *"Precios"* ubicado en la cabecera de la aplicación el cual nos llevará a la vista de precios en donde se muestran los tres planes que se ofrecen: el plan *Básico*, el plan *Estándar* y el plan *Profesional*. Dado que ahora estamos suscritos al plan *Básico*, el botón que tiene asociado está deshabilitado y tiene como texto *"Estás suscrito"* mientras que el botones del resto de planes están habilitados y tienen como texto *"Suscribirte"*.

Tras seleccionar el ciclo de facturación que deseemos, hacemos clic en el botón de *"Suscribirte"* del plan escogido el cual nos llevará a la vista con el formulario de pago. Aquí debemos introducir el número de tarjeta de crédito que ofrece Stripe para realizar un pago válido junto con cualquier fecha de caducidad y código de seguridad. Por ejemplo: *"4242 4242 4242 4242"*, *"12 / 20"* y *"322"* respectivamente.

Si los datos introducidos son correctos, al pulsar el botón de *"Comprar suscripción"* se nos redirige a la ruta raíz de la aplicación y se nos muestra un aviso indicándonos que nuestra suscripción se ha realizado correctamente. Además, en nuestra dirección de correo electrónico recibiremos dos mensajes. El primero es para indicarnos que la suscripción se ha realizado correctamente y el segundo es para notificarnos de que el pago de la suscripción se ha realizado correctamente, de cuando se realizará el próximo y proporcionarnos el enlace para acceder a la factura.

Por otra parte, si accedemos a la vista del plan actual a través del enlace de *"Mi plan"* del panel de administración podemos observar cómo ha pasado de tener los datos del plan *Básico* a tener los del plan adquirido.

Historial de pagos y facturas

Para consultar el historial de pagos debemos hacer clic en el enlace de *"Historial de pagos"* ubicado en el menú lateral del panel de administración el cual nos llevará a la vista donde se muestran todos nuestros pagos realizados en la aplicación. En este caso se mostrará un único pago procedente del proceso de adquirir suscripción realizado en el paso anterior. Para cada pago se muestra el código de referencia, el nombre del plan adquirido, la fecha y hora de realización, el precio pagado, el método de pago y el estado.

Si queremos consultar la factura asociada a un pago, debemos hacer clic en el botón de *"Factura"* el cual mostrará la factura en una nueva pestaña de nuestro navegador. Aquí podremos descargar la factura en formato PDF.

Cambiar método de pago

Para cambiar nuestro método de pago debemos acceder a la vista del plan actual y hacer clic en el botón *"Actualizar"* de la sección de *"Facturación"* el cual nos llevará a la vista con el formulario para actualizar nuestro método de pago. Este es idéntico al formulario de pago para adquirir una suscripción asi que debemos introducir el número de tarjeta de crédito, la fecha de caducidad y el código de seguridad.

Si los datos introducidos son correctos, al pulsar el botón de *"Actualizar método de pago"* se nos redirige a la vista del plan actual y se nos muestra un avisto indicándonos que nuestro método de pago ha sido actualizado correctamente.

Cambiar una suscripción

Para cambiar una suscripción a otro plan de pago debemos hacer clic en el botón de *"Cambiar suscripción"* ubicado en la sección de *"Tipo de cuenta"* de la vista del plan actual. Este nos redirige a la vista con los planes disponibles a los que podemos cambiarnos.

Tras seleccionar el ciclo de facturación que deseemos, basta con hacer clic en el botón de *"Cambiar"* asociado al plan elegido. Si todo sale bien, se nos redirige a la vista del plan actual y se nos muestra un aviso indicándonos que el cambio de la suscripción se ha realizado correctamente. También recibiremos un mensaje en nuestra dirección de correo electrónico indicándonos los anterior.

Cancelar una suscripción

Para cancelar una suscripción a un plan de pago debemos hacer clic en el botón de *"Cancelar suscripción"* ubicado en la sección de *"Tipo de cuenta"* de la vista del plan actual. Se nos mostrará un mensaje de confirmación indicándonos si estamos seguros de cancelar la suscripción. Al aceptar, se nos notificará que la suscripción ha sido cancelada con éxito. Además, la vista del plan actual se actualiza con los datos del plan *Básico*.

En la dirección de correo electrónico recibiremos un mensaje notificándonos también de lo anterior.

Subir un archivo

Para subir un archivo debemos hacer clic en el botón de *"Subir archivo"* ubicado en la vista del directorio raíz a la cual podemos acceder a través del enlace de *"Mis archivos"* del menú lateral del panel de administración. Este nos llevará a la vista con el formulario para subir un archivo. Aquí debemos seleccionar el archivo que deseemos y hacer clic en el botón de *"Subir archivo"*. Si todo es correcto, el archivo no ha sido subido anteriormente y disponemos de espacio de almacenamiento en nuestra cuenta, se nos redirige a la vista anterior y se nos muestra un aviso indicándonos que el archivo ha sido subido con éxito.

NOTA: El proceso anterior es el mismo para subir un archivo a una carpeta o subcarpeta.

Ver detalles de un archivo

Para ver los detalles de un archivo podemos hacer clic en el enlace con el nombre de este o pulsar en la opción de *"Detalles"* de su menú desplegable. Ambas posibilidades nos llevarán a la vista con los detalles del archivo. En esta vista se nos mostrará a quién pertenece el archivo, la fecha en la que se subió, la última fecha en la que se modificó, el tamaño, el tipo y su extensión. Además, si se trata de una imagen, documento PDF o video se mostrará una vista previa del mismo.

Descargar un archivo

Para descargar un archivo debemos pulsar en la opción de "Descargar" de su menú desplegable.

Renombrar un archivo

Para renombrar un archivo debemos pulsar en la opción de *"Renombrar"* de su menú desplegable la cual nos llevará a la vista con el formulario para renombrarlo. En él aparecerá un campo de texto que tendrá como valor el nombre del archivo. Aquí debemos reemplazar este valor con el nuevo nombre para el archivo.

Si el dato introducido es correcto, al pulsar el botón de *"Renombrar archivo"* se nos redirige a la vista anterior y se nos muestra un mensaje de aviso indicándonos que el archivo ha sido renombrado correctamente.

Eliminar un archivo

Para eliminar un archivo debemos pulsar en la opción de *"Eliminar"* de su menú desplegable. Se nos mostrará un mensaje de confirmación indicándonos si deseamos eliminar el archivo. Al aceptar, se nos muestra un aviso indicándonos que el archivo se ha eliminado correctamente.

Compartir un archivo

Para compartir un archivo debemos pulsar en la opción de *"Comparte"* de su menú desplegable el cual nos llevará a la vista con el formulario para compartir el archivo. Aquí debemos introducir obligatoriamente la dirección de correo electrónico del usuario destinatario del archivo y opcionalmente un mensaje.

Si los datos introducidos son correctos, no hemos compartido el archivo con esa dirección anteriormente, la dirección de correo electrónico no está en blanco y tiene un formato válido,

al pulsar el botón de *"Comparte"* se nos redirige a la vista anterior y se nos muestra un aviso indicándonos que el archivo se ha compartido correctamente. Además, en la columna *"Miembros"* del archivo se actualiza al número de miembros con el que lo hemos compartido.

Desde el punto de vista del usuario destinario, este recibirá en su cuenta de correo electrónico un mensaje con el enlace para acceder al archivo y el mensaje opcional si se introdujo.

Este usuario deberá autenticarse en la aplicación para acceder al archivo a través del enlace. Si no dispone de cuenta deberá registrarse previamente y luego volver a acceder al enlace o a la vista de los archivos compartido con él del panel de administración. Una vez acceda se le mostrará los detalles del archivo. Además, podrá descargarlo.

Dejar de compartir un archivo

Para dejar de compartir un archivo con un usuario debemos pulsar en la opción de *"Miembros"* de su menú desplegable el cual nos llevará a la vista en la que se muestran todos los usuarios con el que hemos compartido el archivo en la aplicación. Algunos de ellos puede que aún no estén registrados en ella.

En la vista debemos de pulsar en el botón de *"Dejar de compartir"* del usuario al que queremos quitarle el acceso al archivo. Se nos mostrará un mensaje de confirmación indicándonos si estamos seguros de dejar de compartir el archivo con ese usuario. Al aceptar, se nos redirige a la vista anterior y aparecerá un avisto indicándonos que se ha dejado de compartir el archivo correctamente con ese usuario.

Crear un carpeta

Para crear una carpeta debemos hacer clic en el botón de *"Nueva carpeta"* ubicado en la vista del directorio raíz a la cual podemos acceder a través del enlace de *"Mis archivos"* del menú lateral del panel de administración. Este nos llevará a la vista con el formulario para crear una carpeta. Aquí debemos introducir el nombre de la carpeta que deseemos y hacer clic en el botón de *"Crear carpeta"*. Si todo es correcto, el nombre de la carpeta no está en blanco ni en uso en la ubicación actual, se nos redirige a la vista anterior y se nos muestra un aviso indicándonos que la carpeta ha sido creada correctamente.

NOTA: El proceso anterior es el mismo para crear una subcarpeta dentro de una carpeta o subcarpeta.

Consultar una carpeta

Para consultar una carpeta debemos hacer clic en el enlace con el nombre de la carpeta el cual nos llevará a la vista con su contenido.

B. Usuario administrador

Listar usuarios

Para listar los usuarios registrados en la aplicación debemos hace clic en el enlace de *"Usuarios"* ubicado en el menú lateral del panel de administración del usuario administrador. Este nos llevará a la vista con todos los usuarios y se muestra para cada uno su avatar, nombre, apellidos, email y fecha de registro.

Consultar detalles de un usuario

Para consultar los detalles de un usuario debemos pulsar en la opción de *"Detalles"* de su menú desplegable el cual nos llevará a la vista en la que se muestran todos los detalles del usuario seleccionado.

Consultar las estadísticas de un usuario

Para consultar las estadísticas de un usuario debemos pulsar la opción de *"Estadísticas"* de su menú desplegable el cual nos llevará a la vista en la que se muestran las secciones de *"Tipo de cuenta"* y *"Estadísticas del usuario"*.

En la primera sección se incluye el plan al que está suscrito el usuario, el estado de su suscripción, el estado de almacenamiento y el ciclo de renovación (si la suscripción es de pago).

En la segunda sección se incluye todas las estadísticas que se presentan al usuario en su panel de control.

Consultar el historial de pagos del servicio

Para listar los pagos del servicio debemos hace clic en el enlace de *"Historial de pagos del servicio"* ubicado en el menú lateral del panel de administración del usuario administrador. Este nos llevará a la vista con todos los pagos del servicio y se muestra, para cada uno, su referencia, el nombre y apellidos del usuario que realizó el pago, el plan asociado al pago, la fecha/hora del pago, el método de pago y el estado de este.