UNIVERSIDAD CARLOS III

Análisis y diseño de la aplicación SaaS para gestión de proyectos SimpleDesk

Proyecto fin de carrera – Ingenieria Informatica

Autor: Arranz Mateo, Marcos
Tutor : Rafael Sotomayor Fernández
Director: Luis Miguel Sánchez García

"Lo que no puedo crear, no lo entiendo"

- Richard Feynman

En memoria de mis abuelos, Adoración y Patricio, que gracias a una vida de esfuerzo me dieron la posibilidad de dedicarme a algo que me apasiona.

AGRADECIMIENTOS

Este proyecto fin de Carrera es un punto y seguido en el viaje que empecé hace años al introducirme en el mundo de la informática, las matemáticas y la ciencia en general, que espero que dure muchos años.

Este viaje no habría sido posible sin el incondicional apoyo de mis padres, Remedios y Pedro, que me han animado y dado la oportunidad de estudiar, han sido pacientes cuando las cosas no salían como se esperaba y han compartido mi alegría cuando si lo hacían. Gracias a mi hermano, Arturo, por dejar que le contagiase el entusiasmo por la ciencia y la tecnología y por estar siempre ahí, aconsejando y deseándome lo mejor. Agradezco a toda mi familia por su apoyo.

Gracias a mis amigos de facultad Aarón, Alberto, Alfredo, Apa, David Cacho, David Barroso e Isma por todos los momentos pasados en esas primeras clases cuando todo era nuevo, en tardes interminables en la biblioteca, por compartir los nervios antes de los exámenes y sobretodo, en esos descansos en la cafetería. Gracias a mis amigos de Erasmus, por compartir esa experiencia de la cual nos quedarán recuerdos imborrables para siempre.

Agradezco a mi tutor, la dedicación y paciencia a la hora de realizar este proyecto y su buen humor. Quiero dar las gracias a todos los buenos profesores durante la carrera que se preocuparon porque aprendiese.

RESUMEN

El presente documento tiene como objetivo exponer el Proyecto Fin de Carrera para la universidad Carlos III de Madrid, describiendo el desarrollo de la aplicación web denominada SimpleDesk.

La aplicación SimpleDesk pretende ser una aplicación web que integre diversas herramientas tradicionales de productividad como la toma de notas, alojamiento de ficheros en la nube o creación de tareas. Estas herramientas están diseñadas para ayudar a recopilar, compartir y llevar a cabo proyectos personales o compartidos con otros usuarios de tal forma que pueda sustituir al uso tradicional del correo electrónico en diversos escenarios. La aplicación está basada en el modelo de negocio SaaS (*Software as a Service*), en las últimas tecnologías en el desarrollo *Front-end* y en la interoperabilidad entre servicios a través de APIs (*Application Programming Interface*).

A lo largo del documento se recorrerán las distintas etapas del ciclo de vida de un producto de software. Esto incluye tanto un estudio previo del estado actual de aplicaciones similares y de las herramientas con que desarrollarlo; las actividades de gestión y planificación del proyecto, como los aspectos técnicos del análisis, diseño y desarrollo de la aplicación.

La lectura de este documento servirá al lector recorrer el proyecto desde su concepción hasta su desarrollo final.

Palabras clave

Front-end, Productividad, SaaS, API, Backbone.js, Single-Page Application, MongoDB, NoSQL, Django

ABSTRACT

The purpose of this paper is the presentation of the Final Thesis to Carlos III Madrid University, describing the development of the web application **SimpleDesk**.

The aim of the **SimpleDesk** application is to amalgamate several traditional *Product*ivity tools such as note-taking, file-hosting in the cloud and task management. These tools are designed to help in the collecting, sharing and completing of personal and shared tasks in such a way that replaces the use of e-mail in various scenarios. The application is based on the business model **SaaS** (*Software as a Service*), on the most recent *front-end* development technology and on the interoperability among services through **APIs** (*Application Programming Interfaces*).

The different life-cycle phases of a software *Product* will be covered throughout this paper. Included will be a previous study on the current profile of similar applications and development tools; project planning and management tasks; as well as the technical aspects of the analysis, design and development of the application.

Reading this document will allow the reader to follow the project step-by-step from its conception to its final development.

Key words

Front-end, *Product*ivity, SaaS, API, **Backbone.js**, Single-Page Application, **MongoDB**, NoSQL, **Django**

Tabla de contenido

1 INTRODUCCIÓN

Desde el comienzo de Internet el uso del correo electrónico ha sido el principal actor en la comunicación entre usuarios, tanto en entornos profesionales como familiares o de amigos. Sin embargo, el correo electrónico es una tecnología presente desde hace mucho tiempo, y aun siendo ampliamente usado tiene limitaciones. El ámbito de la aplicación a desarrollar, será crear una aplicación que mantenga la sencillez y eficacia del correo electrónico, que tan popular le ha hecho, combinado con nuevas características acordes a los nuevos tiempos.

En el entorno tecnológico actual, se ha visto alterado en los últimos años por la aparición de dispositivos móviles con acceso a internet. En la actualidad, la mayoría de las aplicaciones desarrolladas para estos dispositivos son aplicaciones nativas, es decir, desarrolladas para una plataforma en concreto, ya sea Android, iOS o Windows Phone por citar unos ejemplos. Debido a este crecimiento en uso de los dispositivos móviles las aplicaciones móviles puede parecer una buena idea crear una aplicación de este tipo, sin embargo se ha optado por crear una aplicación web. Las razones para esto están basadas en increíble desarrollo en los últimos años de los lenguajes y herramientas para el desarrollo web. Paulatinamente las aplicaciones tradicionales de escritorio han sido migradas a versiones web, existiendo una enorme variedad en estas. Al igual que ha pasado esto en las aplicaciones de escritorio es muy probable que pase igual para las aplicaciones móviles, debido a sus ventajas como la gran capacidad de distribución sin necesidad de actualizaciones, el almacenamiento en la nube o la interoperabilidad entre sistemas. Por tanto es probable que los lenguajes que dominan el desarrollo web lo hagan en el desarrollo móvil, y es una motivación dominar y aprender las nuevas tecnologías y mejores prácticas en su desarrollo.

1.1 OBJETIVOS

1.1.1 Objetivos del Proyecto Final de Carrera

Es un problema común a la hora de crear contenido compartido entre varios usuarios que la información se disperse usándose diversas herramientas para esto, como el correo electrónico, servicios de alojamiento de ficheros y servicios de mensajería instantánea. Por tanto el objetivo de este proyecto es crear una aplicación web que permita gestionar pequeños proyectos de forma individual o junto a otras personas de una forma fácil e intuitiva, siendo capaz de compartir tanto archivos multimedia, como notas, tareas o conversaciones.

Siendo el objetivo principal el mencionado en el primer párrafo la aplicación ha de cumplir una serie de sub-objetivos que se lista a continuación:

- Sencillez de uso: Al ser una aplicación orientada a un público no profesional, la sencillez de uso pasa a ser un objetivo prioritario. La interfaz ha de ser capaz de ser usada por un usuario medio sin necesidad de explicación extra.
- Rapidez: Ha de ser una aplicación fluida donde no exista recargos completos de página y ofrezca una experiencia similar a una aplicación de escritorio.
- Integración con servicios externos: Debido que ya existen servicios muy capaces en la toma de notas y en alojamiento de ficheros, se hará uso de ellos mediante el uso de APIs.

1.2 ESTRUCTURA DEL DOCUMENTO

Este documento está estructurado en cuatro capítulos principales. En el primero de ellos se explica el contexto y motivación de realizar esta aplicación, y cuáles son los objetivos a cumplir.

En el segundo capítulo se hace un análisis del estado actual y evolución de las aplicaciones con similares propósitos, tales como herramientas de productividad actuales, o de aplicaciones sociales. A continuación se expone las herramientas actuales a disposición del desarrollo de la aplicación, incluyendo protocolos, patrones, arquitecturas, *frameworks*, etc.

En el tercer punto, se realizara un análisis de los objetivos a realizar, como se ha diseñado la aplicación para satisfacer estos objetivos y a continuación como se han implementado.

1.3 FASES DEL DESARROLLO

El trabajo se ha dividió en 2 fases principales. En la primera de ellas se hace una evaluación del estado del arte y una prueba de concepto de lo que se quiere construir, y cuáles son las tecnologías disponibles para realizar la implementación. La segunda fase consiste en la implementación iterativa del concepto a desarrollar.

1.4 ÍNDICE DE ILUSTRACIONES

1.5 ÍNDICE DE TABLAS

2 ESTADO DEL ARTE

En esta sección se describirá el estado tecnológico actual y de los productos con funcionalidades similares.

2.1 INTRODUCCIÓN

Antes de comenzar cualquier desarrollo es importante conocer el estado actual de posibles competidores del producto a desarrollar, sus puntos fuertes, sus puntos débiles y los posibles nichos de mercado. Una vez analizado una variedad representativa de las soluciones actuales al problema a resolver se está en mejor situación para analizar las características futuras de la aplicación, adaptando las que funcionen en otros escenarios, eliminando las innecesarias y añadiendo las que se consideren oportunas.

Por otra parte es importante analizar los conceptos, herramientas, procesos y tecnologías disponibles para el desarrollo del proyecto a realizar. De esta forma se tendrá una mejor visión de los riesgos que entraña el desarrollo y las mejores soluciones disponibles.

Teniendo en cuenta estos puntos se va a realizar una introducción explicando los puntos y conceptos más importantes dentro del desarrollo para a continuación describir las principales herramientas utilizadas. A continuación se explicaran las metodologías agiles que describirán como realizar el desarrollo. Por último se realizara una comparativa de algunas aplicaciones con objetivos similares, analizando sus virtudes y defectos.

2.2 CONTEXTO TECNOLÓGICO

En este apartado se va a describir los conceptos más importantes relacionados con el desarrollo de aplicaciones web. Esto incluye en primer lugar los nuevos modelos de distribución del software, conocido popularmente como Cloud Computing. A continuación se describirá la historia de las aplicaciones web, y el estado actual de las distintas herramientas involucradas en el desarrollo de estas. En este apartado también se describirá brevemente el estado de las aplicaciones web sociales y las relacionadas con la productividad, y las herramientas utilizadas para su desarrollo.

2.2.1 Cloud Computing

Cloud Computing se refiere a un modelo que habilita el consumo cómodo, ubicuo y bajo demanda de servicios desde un conjunto de recursos informáticos (ej. redes, almacenamiento, servidores, servicios) que puedan ser aprovisionados y desplegados con un esfuerzo de administración mínimo. [1]

Este modelo está compuesto de cinco características principales:

- Servicio bajo demanda: Un consumidor puede aprovisionarse unilateralmente de recursos computacionales, como tiempo de servidor, sin necesidad de una interacción humana.
- Acceso amplio a través de red: Los servicios se ofrecen a través de una red y accedidos mediante una mezcla heterogénea de diversos clientes (ej. Teléfonos móviles, *tablets*, portátiles y estaciones de trabajo).
- Combinación de recursos: El proveedor de los recursos informáticos los combina para servir a múltiples consumidores, con diversos recursos físicos y virtuales, asignados y reasignados

dinámicamente. Existe una sensación de independencia de la localización, en la cual el consumidor no tiene control ni conocimiento de la localización exacta de los recursos.

- Elasticidad: Las capacidades informáticas pueden ser reasignadas de una forma rápida, de forma que escale rápidamente hacia el exterior. Esto hace que para el usuario parezca que las capacidades sean ilimitadas.
- Servicio monitorizado: Los sistemas son controlados y optimizados utilizando un balanceo de la carga según la métrica apropiada para el sistema (ej. Almacenamiento, procesamiento y ancho de banda). Los recursos pueden ser monitorizados, controlados y reportados, proveyendo transparencia para el proveedor y para el consumidor.

El *Cloud Computing* se desarrolló desde principios de los años 2000, por parte de las principales empresas de Internet, como **Google**, **Microsoft** o **Amazon**. Especialmente esta última empresa fue el mayor impulsor de este modelo, gracias a que en 2006 lanzo **AWS**(***Amazon*** *Web Services*). Este servicio surgió de una modernización interna de sus centros de datos, ya que estos como la mayoría de centros de datos estaban siendo utilizados al diez por ciento como norma general, dejando el resto de capacidad para picos extraordinarios.

El uso flexible de los recursos computacionales, ha sido la principal ventaja que ha ofrecido el Cloud Computing, ya que permite pagar por el recurso que se esté utilizando. Además de esta ventaja, los proveedores de *Cloud Computing*, añaden otras ventajas como reducción de costes a largo plazo al evitar los costes de infraestructura, poder dedicar mayores esfuerzos en las aplicaciones frente a la infraestructura, mayor velocidad en el desarrollo e implementación de aplicaciones y un alcance global. [2]

En esta figura podemos ver un diagrama general del modelo *Cloud Computing*.

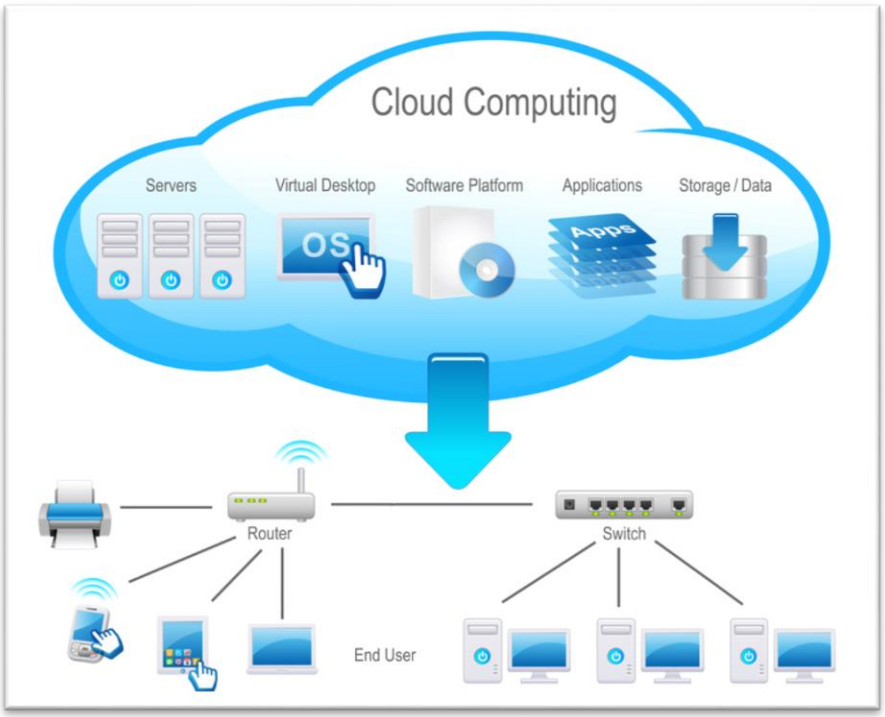


Ilustración 1: Diagrama Cloud Computing (http://www.cloudcomputinginindia.in/images/cloud-computing.jpg)

En esta figura podemos ver como en la nube, se ofrecen distintos servicios, que son entregados al usuario final a través de la red.

MODELOS DE SERVICIO

IaaS

Se define IaaS como el servicio son recursos informáticos fundamentales tales como procesamiento, almacenamiento o ancho de banda. El consumidor no tiene control sobre los recursos subyacentes pero si sobre los sistemas operativos, almacenamiento y aplicaciones desplegadas. [1]

Estos servicios están orientados principalmente a administradores de sistemas, los cuales obtienen ciertas ventajas sobre el modelo tradicional, ya que mantienen un control sobre los el software básico, como sistema operativo, bases de datos o aplicaciones instaladas, pero abstrayendo el manejo de los recursos de hardware. Esto se realiza ofreciendo una instancia de una máquina virtual, la cual se ejecuta en grandes sistemas de recursos distribuidos horizontalmente. Estas máquinas virtuales son configuradas y optimizadas por el proveedor del servicio.

Algunos ejemplos de proveedores son Amazon EC2, DynDNS, HP Cloud y Google Compute Engine.

Amazon EC2

Amazon Elastic Compute Cloud (EC2) constituye una parte central de *Amazon Web Services*, dedicada a ofrecer computo como servicio. [3]

El servicio fue estrenado por Amazon.com, Inc. en Agosto del 2006, desarrollado por un equipo en Sudáfrica dirigido por Chris Pinkham. Desde su lanzamiento el servicio ha sido complementado con nuevas funcionalidades. En Octubre del 2007, se añadieron un nuevo tipo de instancias *High-CPU medium* y *High-CPU extra-large*. Actualmente existen doce tipos de instancias. En la primera mitad del 2008 se añadieron la capacidad de IP estáticas, distintos tipos de zonas y la posibilidad de elegir el *kernel*. En Agosto de 2008 se añadió la opción de almacenamiento al introducir EBS (*Elastic Block Storage*). A finales de 2008 se anunció un acuerdo con Microsoft para ofrecer Windows y SQL server en beta. Además se anunciaron planes para añadir balanceo de carga, auto escalado y monitorización. [4]

En Noviembre de 2010, la web de la propia Amazon fue migrada a Amazon EC2 y AWS.

Amazon S3

Amazon S3 es un servicio web de almacenamiento online. Su objetivo es ofrecer escalabilidad, alta disponibilidad, baja latencia a precios asequibles. Los ficheros se organizan en lo que Amazon llama Bucket. En ellos se puede almacenar cualquier tipo de archivo de hasta un total de 5TB, junto a un fichero de metadatos de 2KB. El acceso a estos ficheros se realiza mediante una interfaz REST o SOAP, o mediante el protocolo BitTorrent.

Amazon S3 ha tenido una gran aceptación, alojando a fecha de Junio 2012 más de un billón de objetos [5]. Muchos servicios comerciales tan importantes como Dropbox [6] o Ubuntu One [7] usan S3 como servicio de alojamiento.

PaaS

Se define como el servicio ofrecido es la capacidad de desplegar aplicaciones en la nube. Estas aplicaciones han sido creadas usando lenguajes de programación, librerías, servicios y herramientas

soportadas por el proveedor. El consumidor no controla la infraestructura pero si tiene control sobre el despliegue de aplicaciones y su configuración. [1]

El servicio está dirigido a desarrolladores o equipos de desarrollo, ya que permite desplegar aplicaciones de una forma sencilla evitando las tareas de configuración y administración del entorno de producción. Las ventajas principales son la facilidad de uso, la simplicidad, la automatización y la integración con otros servicios web mediante interfaces REST o SOAP.

Algunos de los ejemplos más conocidos son: AWS Elastic Beanstalk, Heroku, Windows Azure Cloud Services o Google App Engine.

Heroku

Heroku es un servicio de PaaS que actualmente soporta la ejecución de varios lenguajes de programación haciendo uso de Amazon AWS. Se estrenó en Junio de 2007 como una de las primeras plataformas como servicio. En ese momento soportaba únicamente Ruby pero desde entonces se han añadido Java, Scala, Clojure, Node.js, Python y PHP (no documentado). La infraestructura de Heroku se ejecuta en Amazon Web Services. La principal diferencia respecto a servicios IaaS como Amazon EC2 es que en Heroku la mayoría de servicios de administración y componentes necesarios para el desarrollo vienen incluidos y el desarrollador solo ha de preocuparse por el desarrollo de la aplicación. A Junio de 2013, ejecuta más de tres millones de aplicaciones. [8]

Las características principales de Heroku son:

- Ofrece su servicio en función de procesos web, llamados *dynos*, ofreciendo el primero de maneras gratuita.
- Adicionalmente se pueden añadir procesos trabajadores, para ejecutar tareas rutinarias.
- No se permiten realizar operaciones de escritura sobre la instancia, por lo que si se requiere escribir se ha de hacer en un servicio externo.
- El tamaño máximo del repositorio es de 100 MB.
- La base de datos estándar es PostgreSQL, aunque se pueden utilizar otras gracias a addons.
- Para desplegar una aplicación se hace uso de Git.

Google App Engine

Google App Engine es el servicio PaaS ofrecido por Google. Se liberó en 2008 como versión beta, de la que salió en Septiembre de 2011. GAE permite la ejecución de aplicaciones escritas en Java, Python, Go y PHP. Hasta cierto nivel de recursos es gratuito, que una vez superados se cobrara en base a tiempo de procesamiento, uso del ancho de banda o almacenamiento. Para Python soporta gran cantidad de *frameworks* como Django, CherryPy, Pyramic, Flask, web2py y webapp2 y el *framework* propio de Google. A diferencia de otros proveedores la base de datos ofrecida por defecto no es una base de datos relacional, sino que es BigTable, la base de datos no relacional de Google.

Windows Azure

Windows Azure es el servicio de plataforma e infraestructura creado por Microsoft y lanzado en Febrero del 2010. Permite el despliegue y ejecución de aplicaciones creadas en ASP.NET, PHP y Node.js. Para su despliegue se pueden utilizar mediante FTP, Git, o Team Fundation Server.

Amazon Elastic Beanstalk

Es el servicio PaaS de Amazon, que permite la ejecución de aplicaciones en diversos servicios de AWS como EC2 o S3. Permite una gran variedad de entornos de ejecución entre los que se encuentran:

- Ruby, PHP y Python en servidores HTTP Apache.
- .NET en IIS 7.5
- Java en Apache Tomcat.
- Node.js

Para el despliegue se puede utilizar Git y en el caso de Java ficheros WAR.

SaaS

Cuando recurso ofrecido por el proveedor es una aplicación que se ejecuta en una infraestructura en la nube. Estas aplicaciones pueden ser accedidas desde diversos clientes. El consumidor no controla los recursos subyacentes, como la red, servidores o sistemas operativos. [1]

Los servicios SaaS ofrecen aplicaciones que se ejecutan sobre una infraestructura en la nube, estando enfocadas al usuario final. Las aplicaciones son accesibles desde cualquier cliente ligero, ya sea un móvil, una Tablet o un navegador, por ejemplo. SaaS se ha convertido en un modelo común para muchas aplicaciones empresariales, gracias a las ventajas que proporciona como puede ser reducir los costes de mantenimiento del departamento IT o actualizaciones más sencillas del software. Aunque existe una gran variedad de servicios SaaS, hay algunas características comunes a la mayoría de ellas son:

- Personalización: La mayoría de aplicaciones SaaS permiten adaptar la interfaz para adaptarse a los colores y logo de la empresa cliente.
- Entrega de funcionalidades acelerada: La entrega de nuevas funcionalidades se realiza en ciclos más cortos que el software tradicional debido a esta hospedada de forma central por los cambios son realizados por el proveedor.
- Integración con protocolos abiertos: Debido a la necesidad de acceder a datos de sistemas internos se ofrece integración mediante APIs, normalmente sobre protocolos HTTP, SOAP o REST.
- Colaboración: debido al estar accedidas por red e inspiradas por las nuevas redes sociales y web 2.0, muchas aplicaciones SaaS ofrecen muchas características de colaboración entre usuarios.

Actualmente existe una gran variedad de aplicaciones ofrecidas en SaaS, desde software ofimático, email o aplicaciones empresariales como CRM, ERP, software de gestión de proyectos. Algunos de los ejemplos más conocidos son Google Apps, Microsoft Office 365 o Salesforce CRM.

Servicios de alojamiento de archivos

Una mención aparte dentro de los servicios SaaS, son los servicios de alojamiento de archivos o también llamados servicios de alojamiento en la nube. Estos consisten en servicios que dan la capacidad a los usuarios de subir ficheros a la 'nube' para más tarde ser accedidos desde otros dispositivos.

En los últimos años el uso de este tipo de aplicaciones ha aumentado considerablemente y una gran variedad de servicios son ofrecidos por las compañías más destacadas. Los servicios más populares son Dropbox, Box, Microsoft SkyDrive, Google Drive, Apple iCloud y Amazon Cloud Drive.

Dropbox

Dropbox fue fundad por dos estudiantes del **MIT**, Drew Houston y Arash Ferdowsi, después que el primero de ellos olvidase repetidas veces su pen drive. En Junio de 2007 **Dropbox** Inc., fue fundado y al poco tiempo recibió financiación por parte de **Y Combinator**.

Desde el punto de vista tecnológico, el cliente de escritorio de **Dropbox** está escrito en **Python**, haciendo uso de las librerías **wxWidgets** y **Cocoa**. Para ahorrar en el ancho de banda en las sincronizaciones hace uso de codificación delta, que permite actualizar tan solo los fragmentos del fichero modificados. Para el almacenamiento hace uso de los servicios **Amazon** S3 bajo un encriptamiento AES-256, y las transmisiones son hechas a través de SSL.

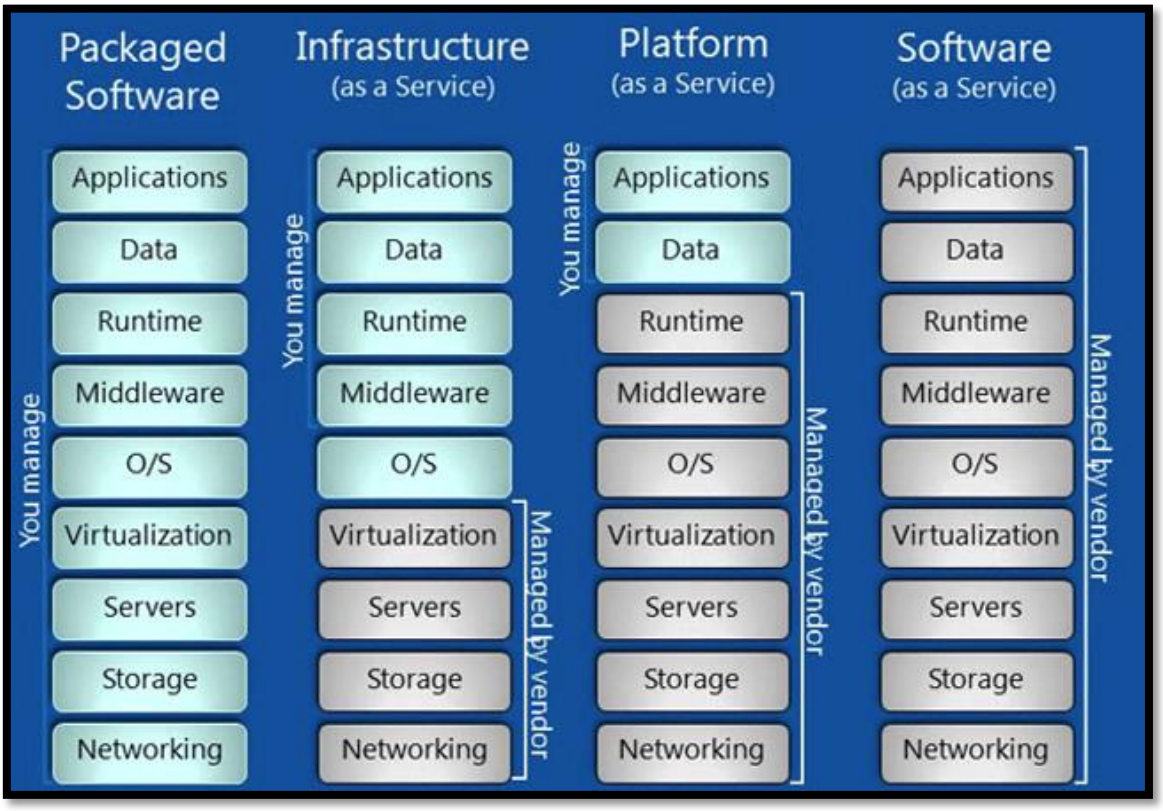


Ilustración 2: Modelos servicio Cloud Computing (http://venturebeat.files.wordpress.com/2011/11/iaas-paas-saas.jpg?w=1072&h=736)

Otros

Otros modelos mencionados habitualmente son los siguientes:

NaaS

Network as a Service es un servicio que ofrece la capacidad de transporte y la conexión entre nubes. **NaaS** incluye la optimización de recursos considerando las redes y los recursos de computación como un todo unificado.

Los modelos más comunes incluyen redes privadas virtuales (**VPN**), Ancho de banda en demanda (**BoD**) y redes móviles virtuales.

DaaS

La virtualización de los escritorios significa que se ofrece como servicio toda la responsabilidad *back-end*, como el almacenamiento, copias de seguridad y actualizaciones.

BaaS

BaaS es un servicio que provee de a los desarrolladores de aplicaciones web y móviles servicios para persistir los datos. Adicionalmente suelen ofrecer otro tipo de funcionalidades como el tratamiento de usuarios, notificaciones push e integración con redes sociales. Los servicios se proveen mediante un SDK o una API. El desarrollo de BaaS es muy reciente, datándose la mayoría de empresas a partir de 2011.

MODELOS DE DESPLIEGUE

Nube privada

La infraestructura es ofrecida para el uso exclusivo de una organización. Puede ser propiedad, manejada y operada por la organización, por servicio de terceros, o por una combinación de ambos. [1]

Nube pública

La infraestructura esta ofrecida para el uso general por parte del público. Puede ser propiedad, manejada y operada por empresas, organizaciones académicas, gubernamentales o combinación de ellas. [1]

Nube comunitaria

La infraestructura esta ofrecida para el uso exclusivo de una comunidad especifica de consumidores con intereses comunes (ej. Requisitos de seguridad). Puede ser propiedad, manejada y operada por una o más organizaciones de la comunidad, por un servicio de terceros o por combinación de ambas. [1]

Nube hibrida

Una nube hibrida es una composición de dos o más nubes de distinta infraestructura que se mantienen como distintas entidades, pero que están unidas por medio de la estandarización o por el uso de tecnología propietaria que permite la portabilidad de datos y aplicaciones. [1]

2.2.2 Aplicaciones web

A continuación se describe la historia y estado actual del desarrollo de aplicaciones web.

INTRODUCCIÓN

En este apartado se va a describir las aplicaciones web, explicando que son y de donde surgen. A continuación se describirán las tecnologías usadas en la actualidad para su desarrollo. Por último se comentara el estado del arte de los dos grupos de aplicaciones web relacionadas con la aplicación a desarrollar: las redes sociales y las aplicaciones de productividad.

DEFINICIÓN

Las aplicaciones web son programas en el cual los usuarios acceden mediante la red, ya sea Internet o una intranet. Estas aplicaciones son programadas en lenguajes soportados por un navegador web, como puede ser JavaScript. Las aplicaciones web son actualmente muy populares, gracias a la ubicuidad de los navegadores web, lo que permite acceder a una aplicación web desde diversos dispositivos.

HISTORIA

En los comienzos de internet las páginas web eran estáticas y la interactividad era muy reducida. En 1995 Netscape introdujo un lenguaje script que se ejecutaba dentro de su navegador. Este lenguaje era JavaScript, y permitía añadir elementos dinámicos a la interfaz de usuario. A partir de 1996 Macromedia lanzo Flash, un re*Product*or de animación, que se añadía como *plugin* al navegador, el cual posibilitó una interactividad mucho mayor de la que anteriormente. En 2005 se produjo un avance importante debido a la introducción de la tecnología AJAX. Esta tecnología combinaba el uso de JavaScript con XML, para realizar llamadas al servidor y refrescar parcialmente la página web sin necesidad de hacer una descarga completa de esta. Adicionalmente actualmente está en desarrollo la última revisión de HTML, HTML5, la cual hace énfasis en las nuevas capacidades interactivas sin necesidad de *plugin*s.

Desde el punto de vista de negocio la tendencia actual de las compañías es proveer aplicaciones mediante la red, las cuales anteriormente eran distribuidas como aplicaciones de escritorio. Las razones de esta tendencia se encuentran principalmente en la facilidad de distribución ante nuevas actualizaciones del software, ya que solo es necesario actualizar el software en el servidor y no en cada equipo. Además los usuarios solo necesitan un equipo ligero para la ejecución de la aplicación, siendo el consumo de espacio mínimo. También es importante señalar que, aunque existan algunas diferencias entre los distintos navegadores, el código del servidor es único y no es necesario tener una versión distinta para cada sistema operativo, como si pasa en aplicaciones de escritorio. En el modelo de Cloud Computing estas aplicaciones se conocen como SaaS (Software as a Service).

Actualmente existe una variedad muy amplia de aplicaciones web: desde aplicaciones de oficina como procesadores de texto, hojas de cálculo, herramientas de presentación a clientes de correo, pasando por editores de video, herramientas de gestión empresarial, gestión de proyectos, etc.

TECNOLOGÍAS WEB

En este apartado se describirán las principales tecnologías, patrones y arquitecturas usadas en el desarrollo web en la actualidad.

Arquitecturas web

En un principio la interactividad de las aplicaciones era muy reducida: en el lado del cliente algún pequeño script JavaScript y en el lado del servidor pequeños formularios y su persistencia. Ante tan poca complejidad, modelos como el que ofrecía PHP, triunfaron rápidamente, gracias a una extrema sencillez. El comportamiento consistía en una página HTML en la cual se abrían unas etiquetas especiales, para indicar que lo que iba entre ellas era código. Dentro de este código se escribía un código PHP, que recuperaba la petición del cliente, la trataba y la persistía si era necesario. Una vez hecho esto se devolvía la página demandada.

A medida que la complejidad crecía, se constató que este modelo era muy limitado, pues pronto derivaba en código repetido y por tanto poca reusabilidad, poca separación de responsabilidades y en definitiva un código de muy baja calidad. Ante esto Microsoft público en 2002, ASP.NET Webforms. Microsoft intento trasladar una experiencia similar a la programación de escritorio a la web. Esto consistía en una programación basada en eventos. En Webforms la página se dividía en dos: una presentación, donde se escribía HTML junto con etiquetas de ASP, utilizadas para incrustar variables o componentes. Además, existía lo que se llama *code-behind* en el cual se escribía código de servidor. Los componentes de las vistas generan código ejecutable en el cliente. Estos controles, reaccionan a eventos en el cliente, los cuales generan una llamada al servidor como si de una aplicación de escritorio se tratase.

Tanto el código en la misma página como Webforms siguen siendo soluciones ampliamente usadas, pero su falta de separación de responsabilidades, la opacidad de Webforms y la dificultad a la hora de testear la aplicación han dado paso a la adopción del patrón MVC (modelo-vista-controlador) de una forma mayoritaria por los desarrolladores y el uso de los *frameworks* que lo implementan. Este modelo propone una separación de responsabilidades en, como su nombre indica, un modelo, una vista y un controlador. Las responsabilidades de cada uno se resumirían así:

- El modelo: Es la representación específica de la información que se quiere tratar. Su responsabilidad será la extracción y persistencia de los datos, así como la lógica de negocio.
- El controlador: Su responsabilidad es responder a las peticiones realizadas. El controlador escogerá el modelo/s necesarios para completar la petición y enviara los datos a la vista correspondiente. Es un intermediario entre el modelo y la vista.
- La vista: Presenta los datos del modelo en el formato adecuado.

La interacción usual es que el usuario interaccione con la interfaz, resultando en una petición al servidor. El controlador capturara la petición, accediendo al modelo para extraer información o modificarla. El controlador pasa el objeto modelo a la vista, el cual representara esta información de una manera adecuada.

Aunque en un principio este patrón fue desarrollado para aplicaciones de escritorio, ha sido en las aplicaciones web donde mayor éxito ha tenido. Actualmente existen una gran variedad de *frameworks* que lo implementan, tales como:

- Ruby On Rails: creado para Ruby en el año 2005 por David Heinemeier Hansson.
- ASP.NET MVC: respuesta de Microsoft al patrón MVC.
- Django: *framework* similar a Ruby On Rails para Python.
- Symfony: *framework* MVC para PHP

- Grails: *framework* MVC para Groovy
- Spring MVC: *framework* MVC para Java.

En un principio la mayoría de *frameworks* se decantaban por lo que se denomina cliente ligero, en el cual la gran mayoría de funcionalidades son controladas por el código que se ejecuta en el servidor. De esta forma, la vista se genera en el servidor y este envía la página de HTML, con los recursos correspondientes (hoja de estilos, código script, imágenes, etc.) al cliente. Esta página tendrá una interactividad limitada, y para la gran mayoría de interacciones se requiriera enviar una petición al servidor el cual devolverá otra página entera. La tendencia actual es dotar de mayor dinamismo a las vistas, cambiando el estado de la página mediante llamadas AJAX, las cuales tan solo actualizan una parte de la página. Cuando la gran mayoría de la interactividad y el código se realizan en el cliente se denomina servidor ligero.

En el caso de que la totalidad de la página exista en una página web se denomina SPI (single page application). En este caso el cliente hará una carga inicial de todo el HTML, CSS y JavaScript necesario para ejecutar la página. A partir de ese momento, no se realizara ninguna recarga total de la página. En este caso la tendencia es a que el servidor provea una interfaz, normalmente REST, donde el cliente realice las peticiones mediante AJAX. Estas peticiones serán devueltas al cliente en forma de datos (JSON o XML, normalmente), los cuales serán presentados en la interfaz por el código del cliente. Adicionalmente el estado del historial se ira modificando gracias a la manipulación de la URL, bien con el símbolo #, o gracias a HTML5 del estado pushHistory, lo cual permite hacer uso del botón atrás de una forma consistente.

Servicios Web

Se define como un Sistema software diseñado para permitir interacciones interoperables entre maquinas sobre una red. Tiene una interfaz descrita en un formato que puede ser procesado por una maquina (WSDL específicamente). Otros sistemas interactúan con el servicio web de la manera especificada en la descripción mediante mensajes SOAP, típicamente a través de HTTP con una serialización XML en conjunto con otros estándares Web. [9]

W3C también añade: se pueden identificar dos clases de servicios web:

- Servicios conformes a REST, en los cuales el primer propósito del servicio es manipular representaciones XML de un recurso Web utilizando un conjunto de operaciones sin estado.
- Servicios web arbitrarios, en los cuales el servicio puede estar expuesto a un conjunto arbitrario de operaciones. [10]

Los servicios web tienen un papel muy importante en la comunicación entre diversas aplicaciones e infraestructuras dentro de la web, ya que los consumidores tan solo tienen que conocer el formato de las peticiones y el de las respuestas, sin importar el lenguaje o tecnología del proveedor.

Algunas de las ventajas que aportan se pueden resumir en los siguientes puntos:

- Integración de aplicaciones y datos: Facilita la interoperabilidad gracias al uso del protocolo HTTP, el cual está presente en muchos escenarios y al uso de mensajes en texto plano.
- Versatilidad: Los servicios web pueden ser consumidos por cualquier tipo de aplicación o humanos vía el navegador.
- Reutilización de código: Un servicio puede ser utilizado por múltiples clientes, lo que evita tener que crear un módulo para esa funcionalidad.

- Ahorro de costes: La interoperabilidad permite crear aplicaciones que integren datos de diversas procedencias a un coste muy bajo, gracias entre otras cosas a que los protocolos son abiertos.

Por el contrario, estas ventajas se consiguen a costa de un mayor coste en el rendimiento tanto en tiempo de procesamiento como en ancho de banda, pues los archivos XML o JSON utilizados para la comunicación son más grandes que uno binario.

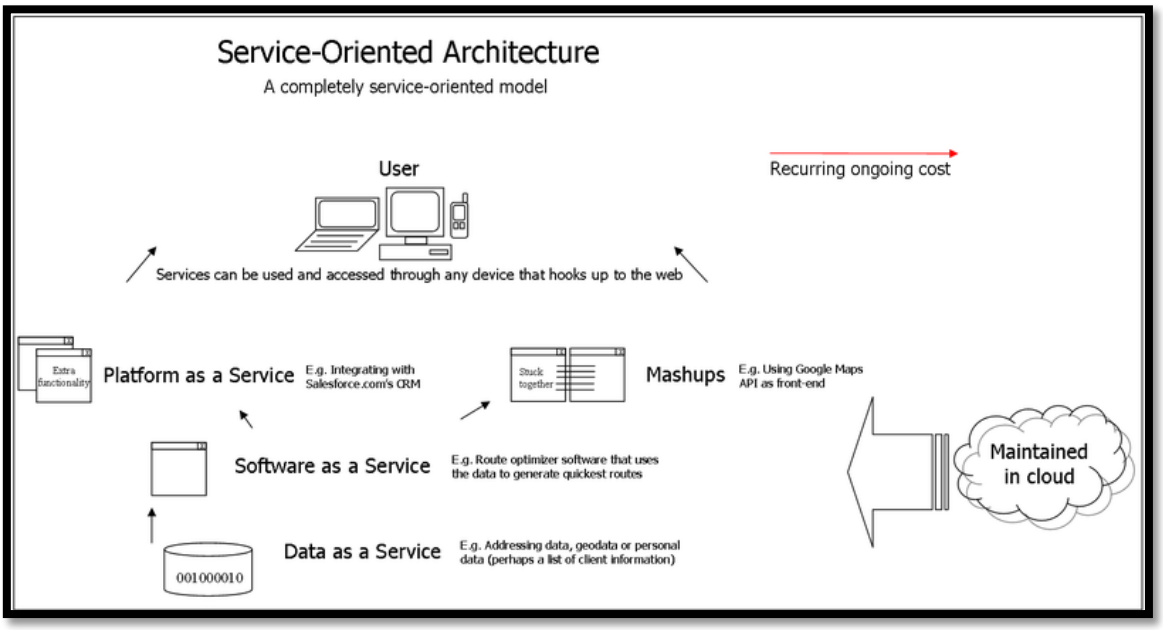


Ilustración 3: Modelo SOA (http://upload.wikimedia.org/wikipedia/commons/0/0b/SOA_Detailed_Diagram.PNG)

SOAP

Simple Object Access Protocol (SOAP) es un protocolo que especifica cómo se comunican dos objetos en diferentes entornos de ejecución por medio de XML. Depende de otros protocolos de aplicación para su transmisión, normalmente HTTP y SMTP.

Consta de tres partes: la envoltura, que especifica que es el mensaje y como se procesa; la cabecera que expresa los tipos de datos y el cuerpo que especifica el procedimiento de las llamadas y sus respuestas.

Las tres características básicas de SOAP son la extensibilidad, permitiendo su desarrollo; la neutralidad, siendo capaz de transmitirse bajo diferentes protocolos y su independencia, al ser posible utilizar cualquier lenguaje de programación.

REST

REST es un patrón arquitectónico desarrollado por Roy Fielding en 2000. Actualmente es la interfaz web predominante.

El comportamiento consiste en llamadas entre un cliente y un servidor. El cliente es el responsable de iniciar la llamada que será procesada y contestada por el servidor. La llamada se hará sobre un recurso. Este recurso puede ser cualquier cosa que tenga un significado coherente. A cada recurso le corresponde una dirección única. A diferencia de otros patrones, REST hace énfasis en una lectura sencilla basándose en el uso de nombres y de unos pocos verbos. A diferencia de SOAP, REST tiene un tipado más débil y no requiere XML.

Existen cinco características para que un servicio se considere RESTful. Estas características son las siguientes [11]:

- Arquitectura cliente servidor: Debe existir una división clara entre el cliente y el servidor, los cuales se comunicaran mediante una interfaz bien definida. Esta separación mejora la escalabilidad y permite remplazar tanto el cliente como el servidor mientras que se respete la interfaz.
- Sateless: es decir, sin estado. Al igual que el protocolo HTTP la comunicación debe ser sin independiente del contexto. Cada petición debe contener toda la información necesaria para ser procesada.
- Cacheable: Las respuestas se han de definir cacheables o no, de una forma explícita o implícita.
- Sistema por capas: el cliente no debe poder distinguir si se conecta al servidor final, tan solo conoce la dirección del recurso. Esto permite introducir servidores intermedios para mejorar el rendimiento o la seguridad.
- Interfaz uniforme: la petición ha de identificar inequívocamente el recurso que se desear acceder, utilizando por ejemplo un sistema URI. Este recurso será devuelto de acorde a su representación, es decir, para una entrada en una base de datos, la respuesta podría ser su representación en JSON.

Tecnologías del lado del cliente

Como se ha explicado las páginas y aplicaciones web han sufrido grandes cambios desde su comienzo. De las páginas estáticas de un comienzo, se ha llegado a páginas con gran interactividad. Entre medias del proceso tecnologías como Flash o Applets se han usado para proveer de interactividad a las páginas. Actualmente estas tecnologías están en decadencia en favor de estándares abiertos. A continuación se detallan las tecnologías más extendidas en el desarrollo moderno en la parte del cliente.

Lenguaje de marcado

El único lenguaje que comprenden los navegadores a la hora de renderizar es HTML (*HyperText Markup Language*), el cual fue creado en 1990. Desde su comienzo a experimentado una evolución importante. En un principio se ideo para representar información, pero a medida que se necesitaban más elementos visuales, este lenguaje cuya principal función es definir la estructura se mezcló con elementos de presentación. A partir del surgimiento de las hojas de estilos CSS, y su adopción, el consorcio W3C, el encargado de definir los estándares web, ha ido eliminando elementos de presentación. La revisión más extendida en la actualidad es la 4.01 y XHTML, un subconjunto compatible con XML. Desde hace pocos años se está adoptando la última revisión HTML5, la cual intenta dar respuesta a las aplicaciones de interfaces ricas.

Hojas de estilo

Como se ha comentado anteriormente, a medida que eran necesarios más elementos visuales en HTML, la complejidad del lenguaje creció y además se hizo un mal uso de HTML al mezclar distintas responsabilidades. Ante esto en 1996, CSS 1 se convirtió en la primera revisión de hojas de estilos en ser oficial por parte de W3C. Las hojas de estilos, y en concreto CSS, sirven para separar la presentación de la estructura. Con CSS es sencillo reutilizar estilos en base a la estructura de HTML. Es

posible especificar un estilo para todo un sitio web para todos los elementos de cabecera, por ejemplo. O extraer la hoja de estilos y reutilizarla para otro sitio web.

A pesar de las constantes revisiones, y adopción masiva, CSS no está exento de problemas, ya que no permite reglas anidadas, ni variables, lo cual da a una difícil reutilización del código. Ante esto han surgido lenguajes similares a CSS, pero que aportan funcionalidades añadidas. El programador ha de codificar y después, compilar el código a CSS. Las soluciones más populares son LESS, SASS y Stylus.

Un problema recurrente a la hora de desarrollar una página web o una aplicación web es crear un estilo común y consistente a lo largo de toda la página. Lo común es desarrollarla desde cero, lo cual requiere un esfuerzo importante y los conocimientos necesarios, o comprar una plantilla, la dará efectivamente un estilo común. Desde hace unos pocos años se ha visto una adopción masiva de diversos dispositivos tecnológicos con acceso a internet y a las páginas web. Esto ha supuesto un reto a los desarrolladores y diseñadores, pues las páginas han de ser capaces de mostrarse de una forma correcta en resoluciones y tamaños muy diferentes. Para solucionar este problema existen recientemente varios *frameworks front-end* que permiten lo que se conoce como responsive design, es decir, una respuesta distinta dependiendo del medio donde se va a mostrar. Además estos *frameworks* proveen una estructura y un estilo consistente.

Los *frameworks* más populares son Twitter Bootstrap y Zurb Fundation.

Lenguajes script

A la hora de hablar de lenguajes de script en el lado del cliente, se ha de hablar de JavaScript, el cual es interpretado por todos los navegadores modernos. JavaScript fue desarrollado en 1996 para el navegador Netscape. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. JavaScript interacciona con el documento del navegador a través del DOM, el cual es una interfaz para interactuar con los documentos HTML. Esta interfaz a pesar de existir una recomendación oficial por parte de W3C, es implementada de forma diferente por cada uno de los navegadores, lo cual da a problemas de compatibilidad entre navegadores.

Al ser un lenguaje creado en un breve espacio de tiempo, JavaScript fue considerado un lenguaje problemático durante mucho tiempo, especialmente por ciertas características que se consideran que tienen una implementación pobre. Además es un lenguaje verboso, es decir, que se ha de escribir mucho para realizar una funcionalidad comparada con otros lenguajes modernos similares, como por ejemplo Ruby o Python. Ante estos problemas, y la necesidad creciente de grandes proyectos en JavaScript, se han propuesto algunos lenguajes para sustituir a JavaScript, como Dart por parte de Google. Sin embargo, la práctica totalidad de las páginas funcionan gracias a JavaScript y es el único lenguaje que funciona en todos los navegadores modernos, por lo cual ninguna alternativa ha tenido una acogida importante. Al igual que CSS, han surgido lenguajes que compilan a JavaScript. Google desarrollo GWT, el cual permite programar en Java y obtener un código JavaScript, Microsoft ha creado el lenguaje TypeScript el cual extiende JavaScript o Coffescript el cual mejora la legibilidad y brevedad del código. Existen infinidad de *frameworks* en JavaScript que, gracias a la dinamicidad de JavaScript, extienden el lenguaje soportando nuevas funcionalidades. Algunos de los más populares es Prototype.js o Underscore.js

Como se ha mencionado anteriormente, la compatibilidad entre navegadores ha sido un problema desde el mismo surgimiento de las páginas web. Las diferentes implementaciones hacen que el código que funciona en un navegador puede que no lo haga en otro, o un elemento se renderice de una forma distinta. Para facilitar esta interacción con el DOM, han surgido varios *frameworks*, entre

los que destaca sobre todos ellos jQuery. Además de jQuery, MooTools o YUI, también intentan facilitar esto.

Aunque jQuery y soluciones similares han facilitado mucho la tarea de los desarrolladores, la tendentica comentada anteriormente de clientes más complejos frente a servidores más ligeros, ha hecho insuficiente el desarrollo exclusivo con estos *frameworks* y tal y como ya pasó en el lado del servidor han surgido *frameworks* MVC. Probablemente el más conocido de ellos es Backbone.js, a la cual han seguido una explosión de *frameworks* en un tiempo muy corto. Otras soluciones son JavascriptMVC, Spine.js, Ember.js, AngularJS, Batman.js, KnockoutJS, etc. Todas estas soluciones proveen de una estructura al código JavaScript, aunque no son puramente MVC, pues el controlador no siempre está presente por lo que se llaman MV*. El funcionamiento habitual es tener un modelo, el cual interactúa con el servidor mediante llamadas AJAX a una interfaz RESTful. Observando este modelo pueden existir una o varias vistas, las cuales serán encargadas de actualizar la plantilla correspondiente de HTML. Estas plantillas están fuera del *framework*, por lo cual se hace uso de uno de los varios *plugin*s existentes, tales como Underscore.js, Mustache.js, Handlebars.js, etc.

Tecnologías del lado del servidor

Servidor

Un servidor es una máquina que escucha peticiones y el software que escucha estas peticiones se denomina Servidor Web. Las peticiones a un servidor web se realizan mediante el protocolo HTTP. Un servidor web por si solo puede servir ficheros estáticos, es decir, paginas HTML, ficheros JavaScript, CSS, imágenes, etc.; pero no puede ejecutar una aplicación. Para ello en un principio se creó el protocolo CGI, el cual especifica de qué forma se ha de comunicar un servidor y la aplicación. Otras soluciones fueron añadir módulos al servidor de forma que este fuese capaz de interpretar el código de la aplicación. Quizás el ejemplo más significativo es el módulo mod_PHP para Apache. En entornos empresariales se popularizo lo que se denomina servidor de aplicaciones, especialmente para *frameworks* como JEE o .NET. Estos servidores ejecutan la aplicación para la que están diseñados, como puede ser WebLogic de Oracle, o Websphere de IBM para Java y IIS para .NET

Otro tipo de servidores son los reverse proxy. Estos servidores tienen como función redirigir las entradas a otros servidores, debido a múltiples razones como pueden ser el balanceo de carga, reducción de la carga de los servidores, seguridad, etc. De esta forma la configuración para un entorno Ruby On Rails podría ser tener un servidor reverse proxy como NGINX que redirija el tráfico a una maquina clúster, que ejecute un servidor web específico para Ruby, como por ejemplo Unicorn.

Lenguajes y frameworks

Los lenguajes de programación son los idiomas artificiales para crear programas. Cuando se habla de lenguajes de programación web se habla normalmente de los lenguajes que se van a ejecutar en el servidor web. En los comienzos del desarrollo web Perl fue muy popular para su programación con interfaces CGI. Alrededor de 1994 se lanzó PHP, el cual gano mucha popularidad en el desarrollo rápido de páginas web. Paralelamente se creó Java, un lenguaje inicialmente creado para el desarrollo en dispositivos electrónicos. Posteriormente se reoriento hacia la incipiente Web. Rápidamente se popularizo en diversos usos, como las Applets, pequeños programas incrustados en las páginas web que se ejecutaba en el navegador o en el móvil. Con la llegada de Java2, y sus versiones J2EE, orientada al entorno empresarial, su uso en empresas se extendió. Desde entonces se han creado muy diversos lenguajes de programación: desde entornos propietarios como C# por parte de Microsoft o Coldfusion por parte de Adobe, o desde el Open Source como Python o Ruby. Se ha de reseñar que muchos de estos lenguajes no son orientados a la Web per se, C#, por ejemplo es

utilizado en diversos ámbitos, es el *framework* el que propicia su orientación a la web, como en este caso sería ASP.NET.

En 2005, Ruby On Rails fue lanzado. Desde entonces su popularidad en el desarrollo rápido de aplicaciones web ha sido creciente. Este *framework* popularizo el uso del patrón MVC, que ha sido imitado en diversos entornos como Django para Python, Grails para Groovy o ASP.NET MVC. La tendencia general es utilizar lenguajes script modernos, gracias a su mayor productividad.

Entre las últimas tendencias, se encuentran los lenguajes que son capaces de ejecutarse en la máquina virtual de Java, ya que de este modo son capaces de aprovechar la tremenda popularidad de Java y su extenso número de librerías, eliminando algunos de los fallos de Java. Entre estos lenguajes se encuentran Groovy, Scala y Clojure. Además, un lenguaje de creciente popularidad es Node.js, que consiste en JavaScript ejecutado en el servidor. Gran parte de esta popularidad se debe a que a diferencia de otros *frameworks* que crean un nuevo hilo por cada petición del servidor, Node.js tiene un único hilo que funciona mediante eventos. Con esto logra un gran rendimiento en aplicaciones con una gran tasa de pequeñas operaciones I/O.

Base de datos

La persistencia de datos ha sido desde el principio de la informática un tema central. Si bien en un principio se almacenaban en ficheros de texto, esto pronto se vio insuficiente para tratar grandes cantidades de datos.

Desde 1970, el paradigma predominante ha sido el de las base de datos relacionales. Estas bases de datos, proporcionan una forma sencilla de almacenar los datos a la vez que ofrecen operaciones ACID (atomicidad, consistencia, asilamiento, durabilidad) y un lenguaje para hacer consultas completas (SQL).

Hasta los años 2000, ningún otro modelo ha sido capaz de popularizarse, a pesar de intentos como bases de datos orientadas a objetos. Sin embargo desde unos pocos años atrás, el uso de un conjunto de bases de datos que no hacen uso de SQL, se ha popularizado bajo el nombre noSQL. Esta popularidad se ha sido gracias a la facilidad de escalar horizontalmente, su rapidez y su capacidad de almacenar datos desestructurados; debido al crecimiento exponencial de los datos generados. Sin embargo esta denominación agrupa a bases de datos muy diversas como basadas en valor-llave como Redis, basada en documentos como MongoDB o en grafos como Neo4j. Actualmente existe una variedad muy grande entre estos *Product*os, como los mencionados Redis, MongoDB, Neo4j o Cassandra, HBase, Voldemort, DynamoDB, SimpleDB, CouchBase, Cloudant, etc.

Otros

Control de versión de código

En el desarrollo moderno de software se suele hacer uso de un sistema de control de versiones. Estos sistemas aportan un sistema de revisión de cada versión del software y de sus cambios.

Actualmente existen dos grandes grupos de sistema de control de versiones: por una parte los sistemas centralizados y por otra parte los sistemas descentralizados. Entre los primeros se encuentran CVS o Subversion y entre los últimos Git o Mercurial. Los últimos tienen una serie de ventajas que hacen que estén ganando más usuarios actualmente.

- Al ser clonado el repositorio entero, se puede seguir trabajando sin conexión.
- Realizar la mayoría de operaciones es más rápido.

- Al estar más replicado el repositorio no es tan necesario hacer backups.
- Es más sencillo crear y mezclar ramas.

Por el contrario, los sistemas distribuidos tienen un aprendizaje más lento y puede ser difícil de comprender para un usuario acostumbrado a los sistemas de control centralizados.

RED SOCIAL

Definición

Se puede definir una red social como un servicio *online* que se centra en ofrecer una relación social entre usuarios que compartan intereses, hobbies o vínculos en la vida real.

Las principales características que definen a una red social serian:

- Creación de un perfil público o semi-publico
- Lista de contactos con los que comparte una conexión
- Capacidad de ver las conexiones de otros usuarios

Cada usuario es representado con un perfil, donde muestra cierta información al resto de usuarios, tal como su nombre, edad, gustos, fotos, etc.

A diferencia de las comunidades online, que están centradas en comunidades, las redes sociales se centran en las interacciones entre individuos.

Historia

El comienzo de los servicios de intercambio nació con los *Bulletin Board System* finales de los años 70. Los BBS permitían la comunicación con un sistema central donde los usuarios podían bajar archivos o mandar mensajes a otros usuarios. Los usuarios principales de los BBS eran entusiastas de la tecnología y habitualmente el tema principal de las discusiones o proyectos era la tecnología.

A principios de los años 90 aparecieron diversos servicios con características sociales tales como, America Online, CompuServe, Prodigy o The Well en un primer intento de llevar el acceso a internet a un público no técnico. CompuServe fue el primer servicio en ofrecer un chat en su programa, mientras que Prodigy facilito el acceso con unas tarifas más accesibles y una interfaz más amigable. AOL proporcionó un acceso mayoritario al público estadunidense a internet. Paralelamente se creó que protocolo IRC, el cual es el padre de la mensajería instantánea.

En la década de los 90, se crearon lo que se puede conocer como primeras redes sociales como los sitios de citas o los foros. Gracias a plataformas como vBulletin o PHPBB, la creación de foros se extendió rápidamente. En 1997 se lanzó el servicio SixDegrees, el cual se considera la primera red social moderna, ya que permitía crear un perfil y conectar con otros amigos. A pesar de no estar en funcionamiento, en su momento cumbre logro más de un millón de usuarios. A partir de los años 2000, han sido muy diversas las redes sociales surgidas. Una de las pioneras fue Friendster, la cual sigue en funcionamiento con más de 90 millones de usuarios registrados. A partir de 2003, se lanzaron dos redes sociales con gran popularidad: LinkedIn y MySpace. La primera de ellas, fue y sigue siendo la principal red social en el entorno empresarial. MySpace tuvo un éxito espectacular y para 2006, era la red social más activa del mundo. Sin embargo, su uso ha decaído considerablemente ante el surgimiento de la red social Facebook. Esta, creada en 2004, por un estudiante de Harvard, Mark Zuckerberg, es a día de hoy la red social más grande por número de usuarios, alrededor de 1000 millones de personas registradas.

Principales redes sociales

En este apartado se van a describir las principales redes sociales que existen en la actualidad, en función de la cantidad de usuarios en su nicho de mercado. Se explicara la historia brevemente, las funciones que ofrece y las tecnologías usadas para su desarrollo y funcionamiento.

Facebook

Como se ha mencionado es la red social con más usuarios registrados. A fecha de Marzo de 2013 son 1.100 millones [12] los usuarios registrados y es el segundo sitio de internet más visitado mensualmente según el ranking Alexa. Creada por un estudiante de Harvard, Mark Zuckenberg, en 2004 empezó como una aplicación web de uso exclusivo por la comunidad universitaria. Sin embargo desde su creación la adopción ha sido notable, especialmente desde 2008 hasta a la actualidad donde paso de 100 millones de usuarios a los 1.100 millones mencionados, como se puede apreciar en la siguiente gráfica.

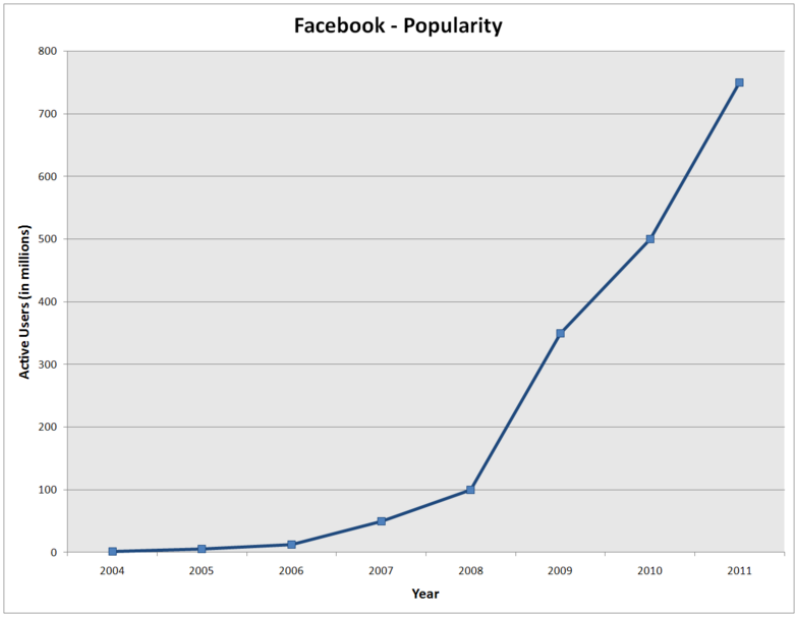


Ilustración 4: Popularidad de usuarios de Facebook
(http://maitemallen.files.wordpress.com/2012/05/facebook_popularity.png)

Como todas las redes sociales los usuarios han de registrarse para usarlo, y una vez hecho esto pueden agregar a otras personas a su lista de amigos siempre que estos acepten la solicitud de amistad. Las actividades centrales son compartir fotos, videos y estados personales con tu grupo de amigos, pero estos servicios se han ampliado con mensajería instantánea, llamadas, aplicación móvil, páginas de marcas, aplicaciones dentro de la plataforma de Facebook o creación de grupos afines. Adicionalmente Facebook ofrece integración de sus servicios a *webmaster* y desarrolladores, de forma que se puede añadir módulos para registrarse o para compartir en tu muro, por ejemplo.

Desde el punto de vista técnico, el código de Facebook está escrito en PHP, el cual es transformado a C++, lo que permite una reducción del 50% en el consumo de los servidores. Para la persistencia de datos se usa un sistema basado en la base de datos noSQL HBase almacenado en un sistema distribuido de máquinas.

Twitter

Twitter es una red social de *microblogging* creada en Marzo de 2006. Desde su creación ha obtenido una gran popularidad, siendo en 2012 más 500 millones [13] los usuarios registrados, con una gran de personalidades siendo usuarios activos.

Al igual que otras redes sociales, te has de registrar y crear un perfil. Una vez registrado puedes seguir a otros usuarios, para poder leer los mensajes que escriban. Estos mensajes pueden ser creados por cualquier usuario a condición de ser un mensaje menor de 140 caracteres. Con los datos recopilados se ofrece una lista con los temas más actuales llamado *Trending Topic*, que son ajustados localmente según la procedencia de cada mensaje. Al igual que Facebook, ofrece *plugin*s para integrar en una página web para funciones como registros o ver los mensajes escritos sobre un determinado tema. Adicionalmente ofrece una API para la creación de aplicaciones de terceros.

Desde el punto de vista técnico, la interfaz web esta creada en Ruby On Rails. Hasta 2008 los mensajes eran manejados por un servidor de colas escrito en Ruby, el cual ha sido migrado progresivamente a software escrito en Scala. El almacenamiento se hace usando MySQL. Además, Twitter contribuye a la comunidad de software abierto, con proyectos como Twitter Bootstrap el cual es el repositorio más popular en GitHub.

LinkedIn

LinkedIn es una red social orientada al entorno empresarial. Lanzada en 2003, la convierte en una de las redes sociales activas más veteranas. Tiene 225 millones de usuarios registrados, lo que la convierte en la red social de trabajo más usada del mundo.

Dentro de LinkedIn puedes crear un perfil profesional donde se agregan los estudios, habilidades y trabajos, que podrán ser consultados por empresas interesadas en contratar nuevo personal. Existen grupos profesionales, por diversas temáticas. Al igual que Facebook ofrece un servicio de actualizaciones de tus contactos.

La mayor parte de LinkedIn está construida en Java usando el *framework* Spring, aunque se utilizan numerosas tecnologías como Grails, Node.js, Git, JRuby, Scala, Ruby On Rails, Backbone.js, MySQL, Oracle, Hadoop, etc. Entre otros métodos de persistencia, LinkedIn hace uso de una base de datos noSQL creada por la propia LinkedIn llamada Voldemort. Es una base de datos clave-valor, de baja latencia escalable horizontalmente. LinkedIn ha liberado varios proyectos como código libre, como Dust.js o Venus.js

Google+

Google+ es un servicio lanzado por Google, Inc. en Junio de 2011. Actualmente es la segunda red social por número de usuarios activos con más de 343 millones de usuarios.

Google+ se ha dirigido a un público generalista con el objetivo de unificar las características sociales de muchos de sus servicios. Frene a redes como Facebook, ha introducido nuevas características como:

- Círculos: En vez de tener una lista unificada de amigos con quien compartir tu información, se han creado grupos con los que el usuario controla el nivel de información compartida.
- Hangouts: videochats grupales entre usuarios.
- Carga instantánea: una función exclusiva de la aplicación móvil, con la que se realiza un backup de las fotos realizadas con el móvil y listas para ser compartidas.
- Grupos: Posibilidad de unirse a grupos de interés.

Google+ está construido al igual que la mayoría de aplicaciones de Google, usando Java servlets en el lado del servidor y JavaScript para la funcionalidad de la interfaz junto con el *framework* propio de Google, Closure. Para la persistencia de datos hace uso de la base de datos clave-valor de Google Big Table.

Pinterest

Pinterest es un servicio lanzado en 2010, orientado a compartir imágenes. Es una de las redes sociales con mayor crecimiento actual, consiguiendo en Diciembre de 2011, poco más de un año después de su creación, ser una de las diez redes sociales con más usuarios activos.

El servicio está orientado a compartir tableros con imágenes, organizados por temas en común, como por ejemplo, decoración, naturaleza, moda, etc. Al igual que Facebook o Google+ ofrece la posibilidad de crear perfiles corporativos. Los usuarios pueden añadir las imágenes fácilmente gracias a una extensión para el navegador.

El *back-end* está construido en Python, junto con una versión modificada de Django y Tornado como servidor web. Para la persistencia de datos se usa MySQL, mientras que el almacenamiento se hace usando Amazon S3. El código *back-end* se ejecuta en instancias Amazon EC2.

GitHub

GitHub es un servicio de hosting de código para proyectos que usan Git como sistema de control de versiones, con muchas características sociales. A Enero de 2013 cuenta con más de 3 millones de usuarios y aloja más de 5 millones de repositorios. [14]

Además del hosting de código, GitHub permite seguir a otros usuarios o repositorios y obtener actualizaciones de la actividad de estos usuarios o repositorios.

El código está construido en Ruby On Rails y en Erlang. Para la persistencia se usan entre otras opciones, Redis, una base de datos clave-valor.

SOFTWARE DE PRODUCTIVIDAD

Desde el comienzo del desarrollo del software se ha creado software de productividad, esto es software que su objetivo sea ayudar a realizar una tarea. La productividad sin embargo es un concepto muy amplio, en el que se pueden incluir hojas de cálculo, procesadores de texto, calendarios, herramientas de comunicación, correo electrónico, etc. Para acotar el dominio del tema a tratar se explicaran cuatro grupos de software que corresponden con las cuatro funciones principales de la aplicación: la comunicación, la gestión de tareas, la toma de notas y la gestión de información personal.

Software colaborativo

Se denomina software colaborativo a herramientas de software que facilitan el trabajo en grupo, mejorando su rendimiento o posibilitando que personas en distintos puntos geográficos colaboren.

Dependiendo del nivel de colaboración el software colaborativo se puede dividir en tres grupos:

Comunicación

Son herramientas para enviar mensajes, ficheros, datos o documentos entre gente para compartir la información de manera desestructurada. Algunos ejemplos son wikis, publicaciones web o el correo electrónico.

El correo electrónico merece mención especial ya que ha sido la principal herramienta de comunicación desde el surgimiento de internet, gracias a una facilidad de uso, su bajo coste y su ubicuidad. Sin embargo el correo electrónico, desde el punto de vista *Product*ivo y de comunicación no está exento de problemas y adolece de los siguientes problemas:

- Perdida de contexto en cadenas de correo muy largas
- Exceso de información: Al ser el emisor el que controla a quien se envía el mensaje el receptor se puede ver saturado con información innecesaria.
- Inconsistencia en la información: al haber grandes cadenas de correo la información puede ser duplicada por los equipos de trabajo.

Conferencia

También se utilizan para compartir la información pero de una manera más interactiva. Algunos ejemplos podrían ser: foros de internet, video conferencia o mensajería instantánea.

Un ejemplo popular de este tipo de herramientas es Skype. Esta aplicación se creó en 2003, y ofrece principalmente mensajería instantánea y conferencia de video y voz a través de VoIP.

Coordinación

Son herramientas complejas para manejar grupos de personas hacia un determinado objetivo, en el cual cada persona puede tener un rol diferente. Los ejemplos más representativos son las herramientas de administración de proyectos, pero existen más como calendarios colaborativos, sistemas de manejo de información, hojas de cálculo online, etc.

Estas herramientas se centran en el manejo de actividades grupales, como pueden ser los calendarios, software social o software de administración.

Software de administración de tareas

El software de administración de tareas comprende el proceso de creación y administración de tareas. Estas tareas sirven a una persona o grupo para llevar a cabo un objetivo. Para un manejo eficaz de las tareas se requiere el manejo del estado de la tarea, la prioridad, las personas asignadas y notificaciones. El manejo de las tareas puede formar parte esencial de la administración de proyectos.

Existen multitud de aplicaciones para el manejo de tareas. Los objetivos de estas son diversos, habiendo muchas aplicaciones de uso gratuito que funcionan como simples listas, a aplicaciones empresariales de pago con multitud de funciones. Algunas de las más relevantes son:

- Google tasks: Es el servicio de tareas de Google. Permite la integración con Google Calendar.
- Wunderlist: Usada por más de 3.5 millones de usuarios, y presente en los dispositivos más importantes. Orientada para un público informal, funciona bajo servicio *Freemium*, obteniendo funcionalidad extra a cambio de una suscripción mensual.
- Any.do: Competencia de Wunderlist, es otra aplicación simple de gestión de tareas. Está disponible para Android, iOS y como extensión para el navegador Google Chrome.
- Trello: Gestor de tareas más completo que los anteriores, está basado en el método Kanban. Los proyectos se representan como tableros con listas que a su vez contienen tareas.
- Asana: Es una herramienta creada originalmente por dos ingenieros de Facebook para mejorar la productividad de los empleados. Entre las características que ofrece se encuentran espacios de trabajo, proyectos, tareas, notas y comentarios. En Junio de 2012, se añadió la funcionalidad de bandeja de entrada, con el objetivo de minimizar la comunicación por email.
- *Producteev*: Una aplicación de gestión de tareas orientada al mundo empresarial, competencia de Asana.

Toma de notas

La toma de notas es un proceso por el cual se captura información de una fuente. El autor de la nota intenta condensar la información más importante, liberando la mente de recordar esta información.

La toma de notas es un proceso central en el comportamiento humano a la hora de manejar información participando diversos procesos mentales y la interacción con otras funciones cognitivas. La persona que toma las notas ha de filtrar, organizar y estructurar la información original, escribiendo la interpretación de este.

Existen diversas aplicaciones para capturar notas:

- Evernote: El servicio de toma de notas más popular actualmente. Con clientes para Android, iOS, Windows Phone, Windows, aplicación web y extensión para el navegador está presente en una amplia gama de dispositivitos. Permite capturar notas directamente desde el navegador, escribirlas manualmente, grabar notas de voz, notas con la cámara, integración con el correo electrónico, con Twitter, etc.
- Google Keep: Servicio lanzado por Google Inc. en Marzo de 2013. Actualmente disponible para Android y como aplicación web en Google Drive.
- Springpad: Una aplicación similar a las anteriormente mencionadas. Disponible para Android, iOS y aplicación web.

Manager de información personal

Un manager de información personal es un tipo de software que funciona como un organizador personal. Como un tipo de herramienta para el tratamiento de información el objetivo es facilitar el seguimiento, captura y manejo de información personal. Los managers de información personal suelen tener todos o algunos de estas herramientas:

- Archivos personales: Documentos, Música, fotos, videos o similares
- Notas
- Direcciones
- Calendario
- Recordatorios
- RSS *feeds*
- Emails

2.3 MODELOS ÁGILES

Los modelos agiles son un subconjunto de la metodología de desarrollo incremental pero con un enfoque distinto, cuya idea principal es anteponer la adaptabilidad a la predicción.

Las metodologías clásicas, son tradicionalmente predictivas, con procesos rígidos y formales donde desde un principio se planifica con gran detalle el producto. Por esa razón estas metodologías se resisten al cambio. Por el contrario, las metodologías agiles incorporan desarrollos iterativos cortos con re-planificaciones continuas, documentación ligera y una comunicación continuada con los clientes.

2.3.1 Origen

El origen del desarrollo ágil se puede encontrar en las metodologías desarrolladas en los años noventa, como reacción a los métodos rígidos usados por aquella época. En el año 2001, un grupo de desarrolladores publicó lo que se conoce como el manifiesto ágil. El texto de este manifiesto dice de este modo:

"Estamos descubriendo formas mejores de desarrollar software tanto por nuestra propia experiencia como ayudando a terceros. A través de este trabajo hemos aprendido a valorar:

Individuos e interacciones sobre procesos y herramientas
Software funcionando sobre documentación extensiva
Colaboración con el cliente sobre negociación contractual
Respuesta ante el cambio sobre seguir un plan

Esto es, aunque valoramos los elementos de la derecha, valoramos más los de la izquierda."

Tal y como dice el manifiesto, no se renuncia a los elementos de la derecha, pero consideran más importante los siguientes puntos:

- Individuos y sus interacciones: la auto organización y motivación es importante, al igual que lo son la colaboración in situ.
- Software funcionando: Se valora más una parte funcional del programa que una presentación de lo que va a ser.
- Colaboración con el cliente: Los requisitos no pueden ser recogidos de una forma completa al principio del desarrollo, por lo que la colaboración con el cliente resulta muy importante.
- Responder a los cambios: Es más importante ser capaces de responder a los cambios que detallar un plan rígido.

El manifiesto se basa en doce principios [15]:

1. Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
2. Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
3. Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.

4. Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.
5. Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
6. El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
7. El software funcionando es la medida principal de progreso.
8. Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.
9. La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.
10. La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
11. Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.
12. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

Existen una serie de características que son comunes a casi todas las metodologías agiles. Todas las metodologías dividen el trabajo en pequeños incrementos que supongan una planificación mínima. Estos incrementos se planifican dentro de iteraciones de una duración de entre dos semanas y dos meses. Cada iteración completa todas las etapas de un ciclo de vida de un producto: análisis, diseño, codificación, testeo y aceptación. Un representante de los interesados en el desarrollo del producto es el encargado de valorar cual serán las funcionalidades que dan más valor.

Existen además ciertas prácticas o herramientas asociadas a las metodologías agiles, con el objetivo principal de mejorar la calidad del software. Entre estas se pueden encontrar el testeo unitario, el desarrollo dirigido por testeo (TDD), desarrollo dirigido por comportamiento (BDD), integración continua, refactorización del código, uso de patrones de diseño, programación por pares, etc.

Como se ha mencionado antes, la principal diferencia entre las metodologías agiles y las metodologías tradicionales es su enfoque, mientras que las primeras tienen como objetivo la adaptabilidad, reconociendo la incertidumbre ligada a la recogida de requisitos; las metodologías clásicas realizan una predicción completa del desarrollo futuro. Teniendo esto en mente, los métodos agiles inciden en ofrecer una satisfacción del cliente mayor gracias a una entrega temprana de software funcional. En la tabla a continuación podemos ver un resumen de las principales diferencias [15]:

	ÁGIL	TRADICIONAL
REQUERIMIENTOS DE USUARIO	Recogida iterativa	Definidos detalladamente antes del desarrollo
COSTE REHACER	Baja	Alta
DIRECCIÓN DEL DESARROLLO	Preparada para el cambio	Rígida
TESTEO	En cada iteración	Al finalizar el desarrollo
PARTICIPACIÓN DEL CLIENTE	Alta	Baja
HABILIDADES EXTRA POR PARTE DE LOS DESARROLLADORES	Habilidades interpersonales y conocimiento del dominio de negocio	Ninguna en particular
ADECUACIÓN DEL TAMAÑO DEL PROYECTO	Tamaños pequeños a medios	Tamaños altos

Tabla 1: Comparativa Metodologías agiles y tradicionales

2.3.2 Metodologías agiles

Existen multitud de metodologías agiles; algunas de las más representativas son Programación Extrema, Scrum y Kanban. A continuación se describen de forma resumida las características de cada una de ellas.

PROGRAMACIÓN EXTREMA (XP)

La programación extrema es una metodología que fue creada por Kent Beck durante su trabajo en Chrysler en 1996. El nombre se refiere a llevar al extremo las buenas prácticas.

Incide en la satisfacción del cliente, y como otros métodos agiles, favorece entregas frecuentes frente a una sola entrega. XP proclama mejorar el software a través de cinco valores: simplicidad, comunicación, *feedback*, respeto y valor. La simplicidad se refiere a realizar tan solo lo que se necesita en el momento, evitando crear código que puede que no se llegue a utilizar nunca. Si finalmente se necesita, se realizaran los cambios necesarios en el código, aunque esto suponga una pérdida de eficiencia, que XP afirma acaba compensándose al no utilizar recursos en grandes diseños. La comunicación implica el trabajo conjunto en el mismo espacio físico. El *feedback* hace referencia a crear software que funcione y pueda ser entregado al cliente, el cual dará información en retorno. El respeto implica tanto el respeto a uno mismo como al resto del equipo, asumiendo la responsabilidad y recibir autoría del código creado. Por último, el valor se refiere a informar honestamente del progreso sin tener miedo al fracaso. [16] [17]

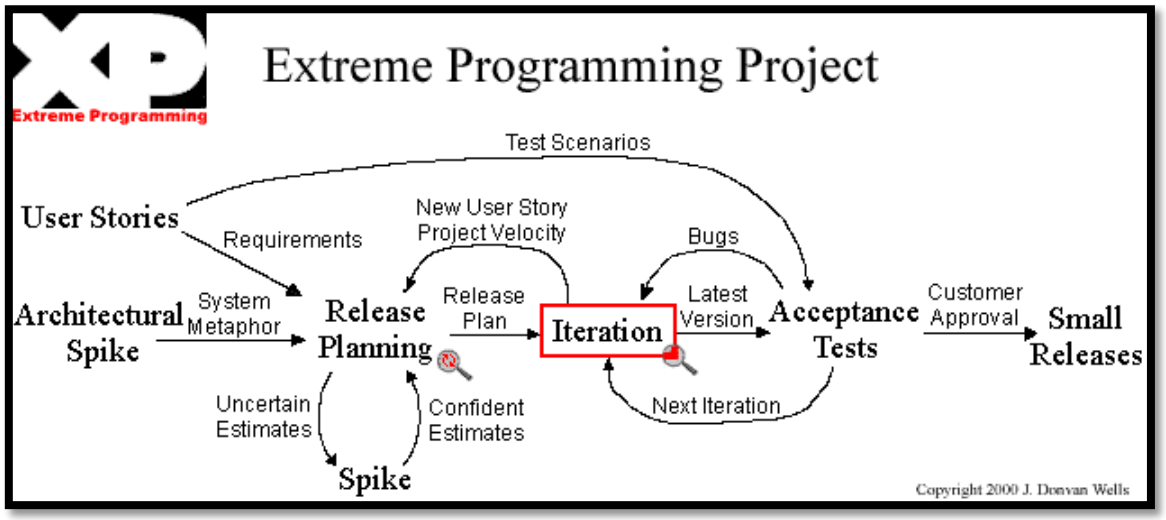


Ilustración 5: Desarrollo Extreme Programming (http://3.bp.blogspot.com/-M7sKIJdAfMs/T-QdacxYSNI/AAAAAAAAAIU/3I7v2-OcE78/s640/xp-project.gif)

XP propone una serie de reglas para llevar a cabo los proyectos. Se pueden resumir de la siguiente forma [16]:

Planificación

- Las historias de usuario se escriben.
- Realizar pequeñas entregas frecuentemente.
- El proyecto es dividido en iteraciones.
- La planificación de la iteración se realiza al principio de la iteración.

Administración

- El equipo trabajara en un entorno abierto.
- Crear un ritmo sostenible.
- Pequeñas reuniones cada día.
- El ritmo de la iteración será medido.
- Distribuir a la gente dentro del proyecto.
- Cambiar XP cuando falle.

Diseño

- Simplicidad.
- Uso de tarjetas CRC.
- Hacer uso de pruebas de concepto.
- Ninguna funcionalidad es añadida sin ser necesaria.
- Refactorizar siempre que sea posible.

Implementación

- El cliente siempre ha de estar disponible.
- El código tiene que adecuarse a los estándares.
- Programa primero el test unitario.
- Todo el código de producción es programado en pares.
- Tan solo una pareja integra el código a un tiempo dado.
- Integrar el código frecuentemente
- Disponer de un servidor de integración.

Testing

- Todo el código tiene que tener testeo unitario.
- Todo el código ha de pasar los tests unitarios antes de ser pasado a producción.
- Cuando se encuentra un bug se crea un test.
- Test de aceptación son ejecutados frecuentemente y el resultado publicado.

SCRUM

Scrum es una metodología donde un equipo trabaja de forma holística para conseguir un objetivo, en comparación con una forma secuencial en las metodologías tradicionales. El nombre viene de un término del rugby en el cual todo un equipo lucha para conseguir la posesión de la pelota. Actualmente se considera la metodología ágil más popular. [18]

En Scrum existen una serie de roles propios que son diferentes al de las metodologías clásicas. Los roles son los siguientes:

- *Product Owner*: Representa al cliente, y es el que tiene la perspectiva de negocio. Es el encargado de escribir las historias de usuario y de priorizarlas.
- Equipo de desarrollo: Es el encargado de llevar a cabo la funcionalidad requerida. Está compuesto por un equipo pequeño con diversas habilidades.
- Scrum Master: Es el encargado de hacer cumplir las reglas de Scrum, y de evitar cualquier distracción al equipo.

El proceso de Scrum consiste en divisiones de tiempo llamadas Sprints. Estos Sprints tienen una duración entre 1 semana y 2 meses, cuya duración permanece fija. Al comienzo de cada sprint se realiza una reunión para la planificación de las funcionalidades y su estimación. Al finalizar el sprint

todas las tareas programadas para ese sprint han de estar finalizadas y se ha de poder entregar un producto funcional. Estas funcionalidades o también llamadas historias de usuario, ha sido previamente almacenada en el *Product Backlog*. El *Product Owner* indicará cuales son las historias de usuario que quiere incluir en el Sprint, es decir las que más valor le reportan. Al comenzar cada día de trabajo se ha de realizar una mini reunión en el que cada miembro del equipo explicara al resto cuál será su trabajo ese día. Al finalizar el Sprint se realizara una reunión de retro inspección para examinar las estimaciones realizadas con la cantidad de trabajo realizado para reajustar la capacidad de trabajo del equipo, y aprender nuevas lecciones sobre el proceso de desarrollo.

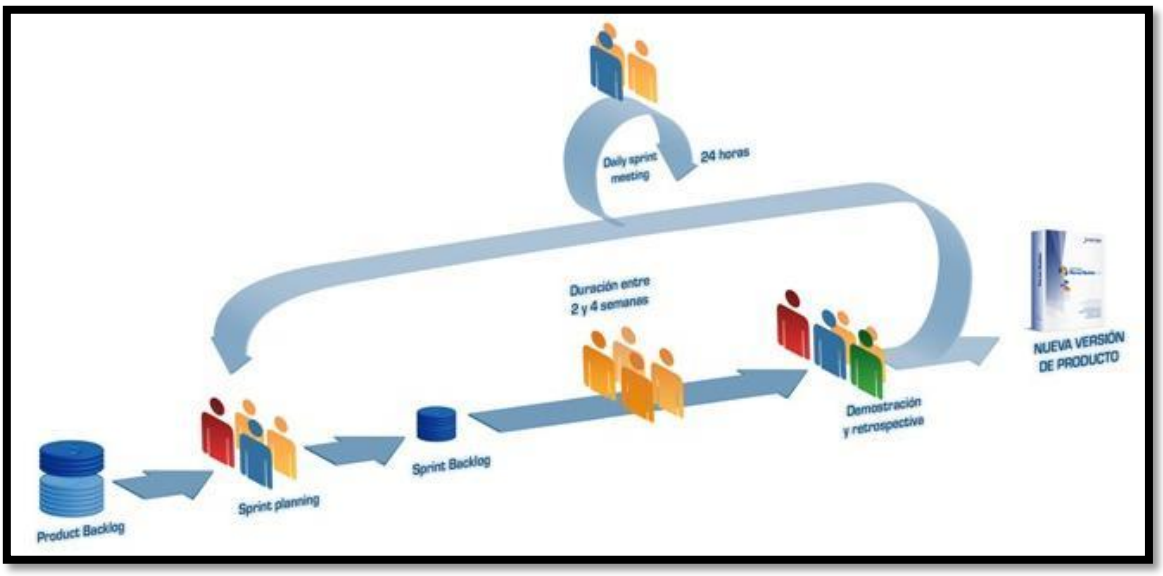


Ilustración 6: Proceso de desarrollo **Scrum** *(https://encrypted-tbn1.gstatic.com/images?q=tbn:ANd9GcR5KAbc0orJPi3YvQ4aVqaqzMirv8ABH9uarCPK6ry8iSqyO8_pfA)*

KANBAN

El método **Kanban** que proviene de las practicas JIT (just-in-time) de Toyota. El proceso ha sido adaptado para el desarrollo software haciendo énfasis en la mejora continua.

El proceso se basa en limitar la capacidad de trabajo en donde se van asignando los trabajos de una pila de tareas. Los principios son los siguientes:

- Comenzar con lo que conoces: **Kanban** no prescribe una serie de roles o procesos, sino que propone seguir con el proceso que se esté usando hasta la fecha, estimulando la mejora continua.
- Cambio evolutivo: El equipo ha de estar de acuerdo en adoptar pequeños cambios. Aunque grandes cambios puedan parecer más efectivos pueden tener una gran cantidad de probabilidades de fallo.
- Mantener los roles: Mantener los roles y títulos ayuda a eliminar algunos de los miedos y dificultades iniciales.
- Actos de liderazgo: Los actos de liderazgo en todos los niveles deberían ser promovidos.

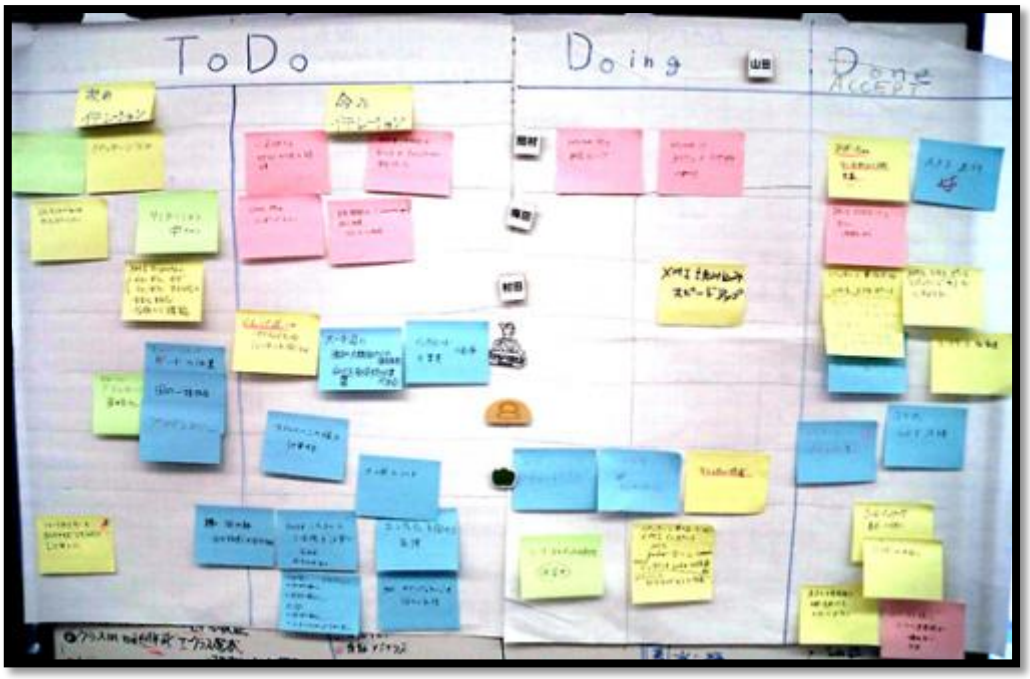


Ilustración 7: Ejemplo de un Tablero ***Kanban***
(http://t3.gstatic.com/images?q=tbn:ANd9GcR7GT_iWDP_Yc2btwfy1grLaAVefEeACuosZk7xL9kdoHuxyVYLWw)

Kanban se sustenta en seis prácticas, las cuales son las siguientes:

- Visualizar: El flujo de trabajo ha de ser visible. De esta forma se puede comprender de una mejor forma la metodología de trabajo.
- Limitar el trabajo en progreso: Ha de existir un repositorio de trabajo, del que se extraen las tareas a realizar.
- Manejar el flujo de trabajo: El flujo de trabajo ha de ser medido y registrado. Ser medido es la base para evaluar las mejoras en los procesos.
- Hacer las políticas explicitas: Sin una política clara de trabajo es difícil mejorar o discutir cambios sobre ella.
- *Feedback*: Implementar políticas de *feedback* para revisar el trabajo.
- Mejorar de forma colaborativa: Se alienta al uso de métodos científicos para mejorar los procesos.

2.4 ANÁLISIS DE APLICACIONES RELACIONADAS

A continuación se exponen aplicaciones con un ámbito y funcionalidad parecida a la aplicación desarrollada.

2.4.1 Apache Wave

Google Wave fue una aplicación web creada por Google y lanzada al público en 2009. Se liberó gran parte del código y se permitió crear extensiones, ya que la intención era convertir a Wave en un protocolo que sustituyese el email como forma generalizada de comunicación en Internet, siendo Google uno de los proveedores de protocolo Wave. Sin embargo, en Agosto de 2010, Google anuncio que cerraría el servicio debido a su baja adopción. El proyecto desde entonces ha sido mantenido por la fundación Apache.

El uso de Wave es muy similar al uso del correo electrónico, pero a diferencia de este no es necesario enviar todos los mensajes anteriores en cada respuesta, sino que el documento, llamado Wave, es almacenado en un servidor central. Cada usuario de este documento es capaz de modificar este documento, mientras que el resto de usuarios podrán visualizar los cambios en tiempo real y hacer ellos cambios de forma concurrente. De esta forma toma un funcionamiento con características de mensajería instantánea, email, wiki o foros.

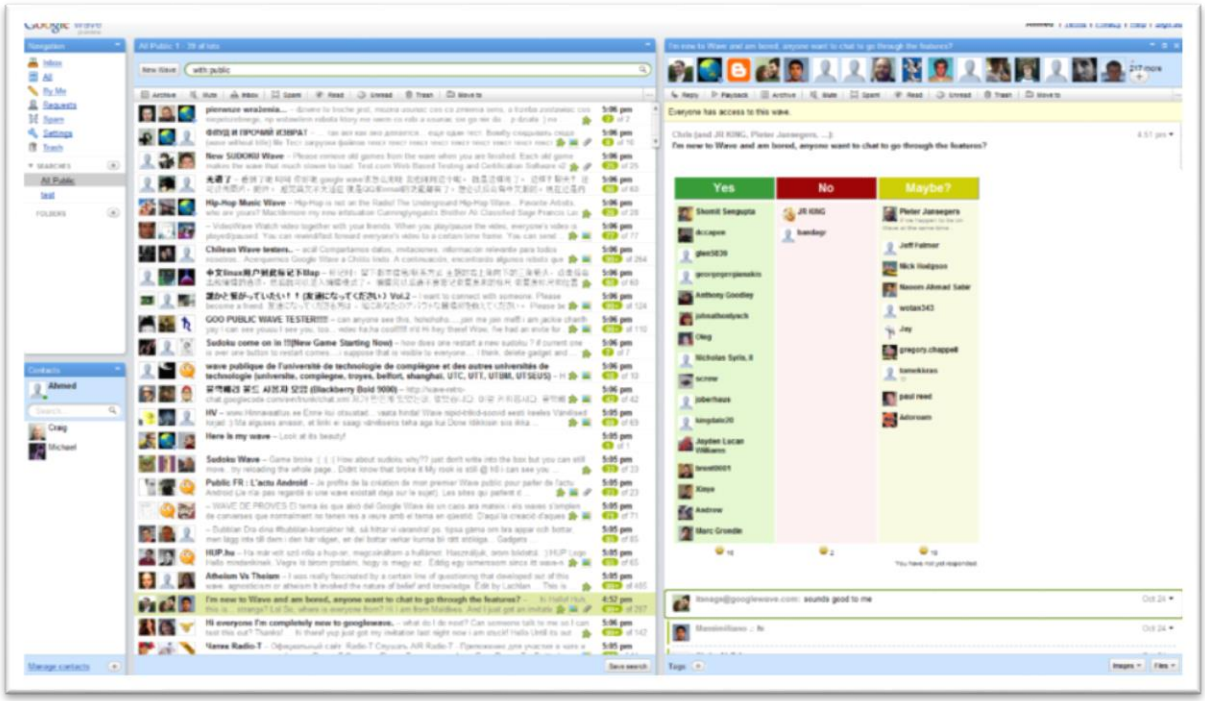


*Ilustración 8: **Google** Wave (https://encrypted-tbn3.gstatic.com/images?q=tbn:ANd9GcR7-szpKbfCPEqWwm9cuNAi2YJURRrqnb6ibQj0NEe8ekavxkS2)*

2.4.2 Dispatch

Dispatch es una aplicación web enfocada a la administración de proyectos web, con la intención de sustituir o reducir el uso del email entre grupos de trabajo.

Intenta funcionar de una manera similar al correo electrónico, para entornos de trabajo, pero reduciendo el ruido y facilitando la capacidad de referenciar y encontrar la información. Permite además integración con servicios de terceros, concretamente **Evernote**, **Google** Drive, **Dropbox** y Box. [19]

Para registrarnos, nos da la posibilidad de hacerlo mediante nuestra cuenta de **Google** o añadiendo un email.

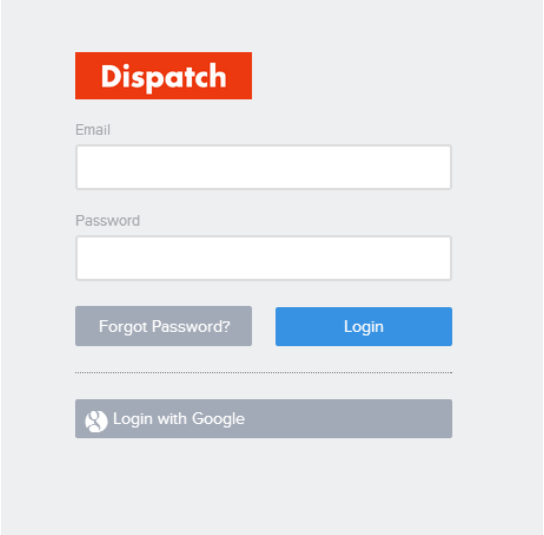


Ilustración 9: Login en Dispatch

Una vez creada una cuenta el funcionamiento consiste en crear un proyecto, llamado Dispatch, el cual tendrá una dirección de email propia.

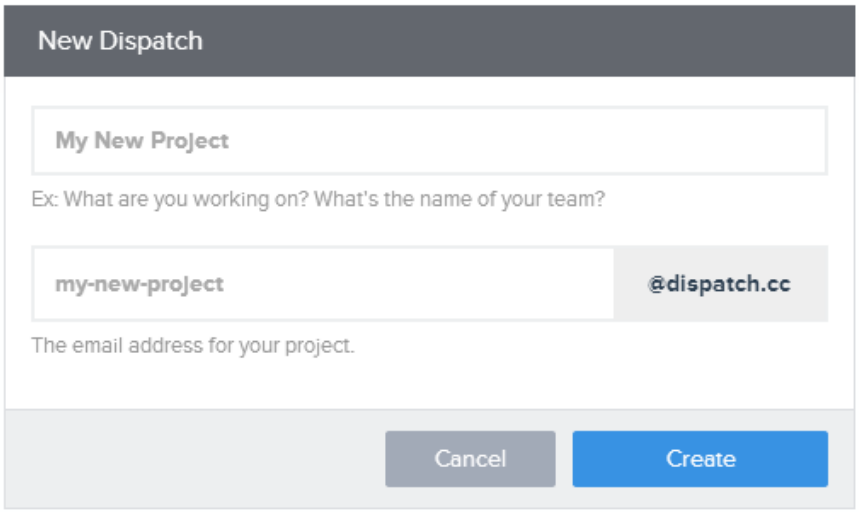


Ilustración 10: Creación de proyecto en Dispatch

A este proyecto se podrán suscribir nuevos usuarios, mediante invitación. La vista principal muestra un flujo de publicaciones. Estas publicaciones pueden ser añadidas por cualquier miembro del equipo, y consisten en un mensaje de texto, un link, un fichero o un link a uno de los servicios de terceros.

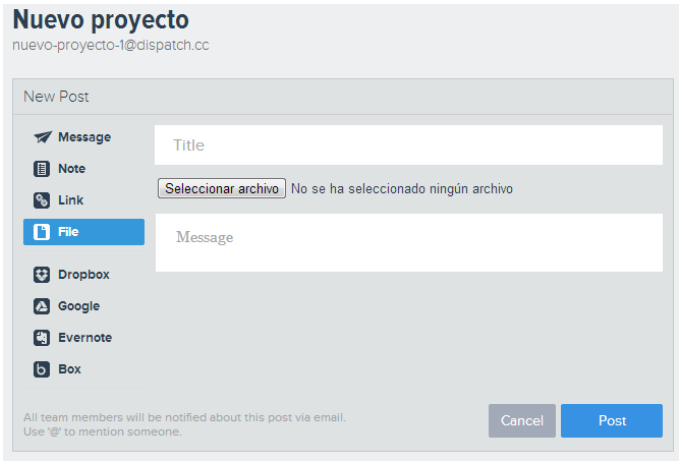


Ilustración 11: Creación de nueva entrada en Dispatch

Una vez añadida la publicación se mostrara en el muro del proyecto.

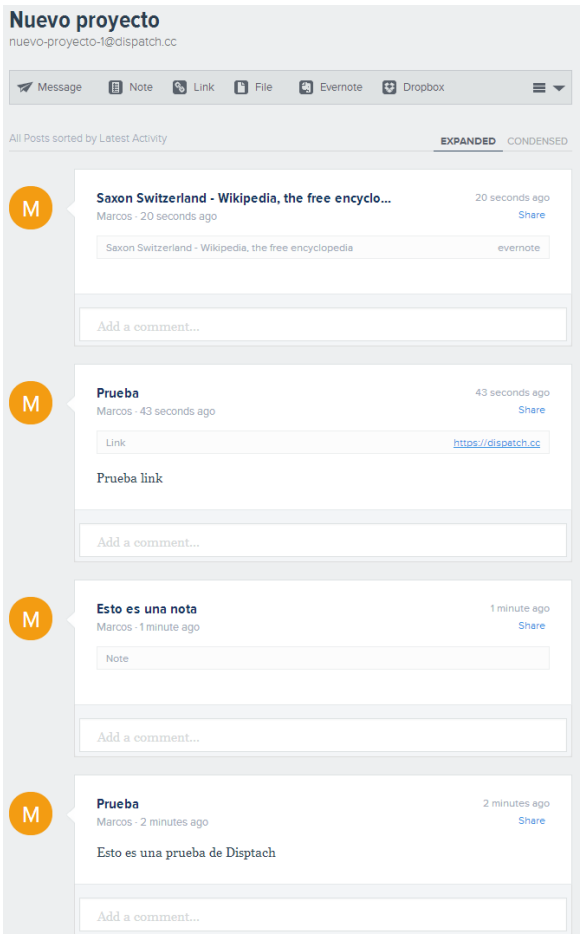


Ilustración 12: Feed de Dispatch

Cada entrada podrá ser distribuida por un link concreto o ser editada.

2.4.3 TeamBox

TeamBox es otra aplicación para crear proyectos con enfoque empresarial. Al igual que otras aplicaciones se puede acceder mediante el uso de cuenta de Google, Facebook o Twitter.

Entre las funcionalidades que ofrece se encuentra crear, manejar y crear informes sobre tareas. También ofrece administración de archivos en la nube, ya sea en sus servidores o integrando con servicios de terceros como Dropbox, Box o Google Drive. Para la comunicación dispone de una vista de conversación y video conferencia integrado. Además se ofrece la aplicación en varios dispositivos. El servicio se ofrece como modalidad Freemium, es decir, gratuita para la versión básica y de pago para funcionalidades más avanzadas.

Una vez creada una cuenta, el usuario obtiene una vista general de todos los proyectos a los que esta suscritos con sus novedades. En el lateral se encuentran más opciones, entre la que se encuentra crear un nuevo proyecto.

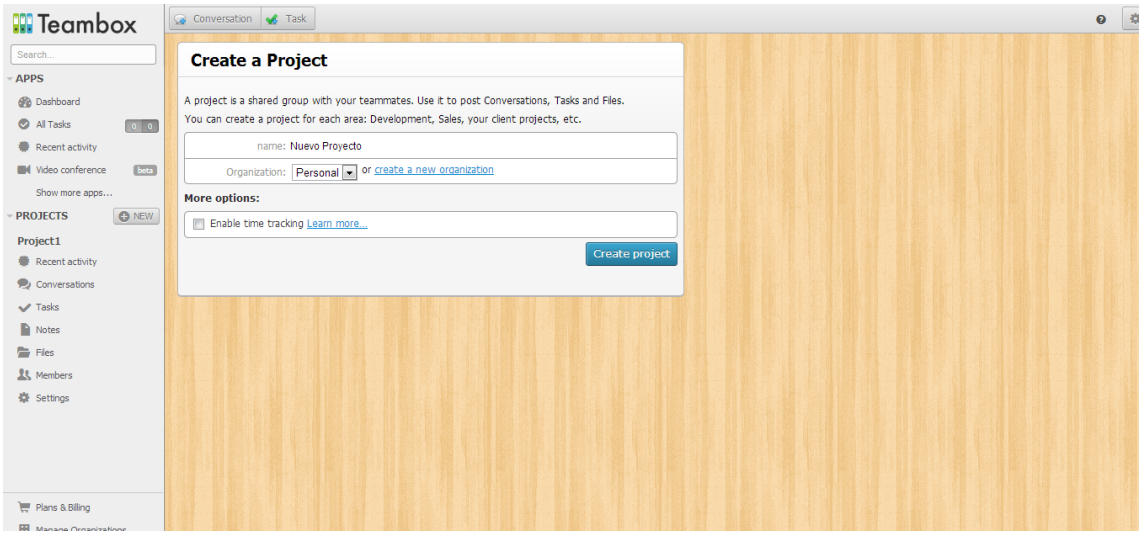


Ilustración 13: Creación proyecto TeamBox

En la ventana de creación, pregunta por la organización a la que se va añadir y si se va a registrar tiempos. Una vez creado el proyecto preguntara con quieres compartirlo, dando posibilidad de introducir las direcciones de correo. A partir de disponer de un nuevo proyecto, se dispone de cuatro vistas principales. En la primera se pueden añadir discusiones sobre un tema concreto.

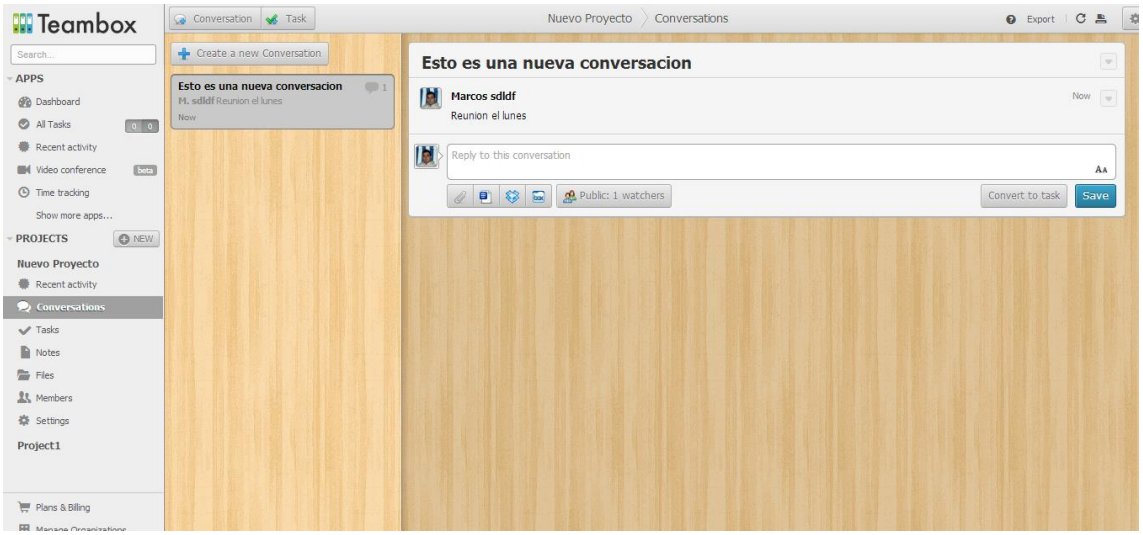


Ilustración 14: Creación de conversación en TeamBox

En la segunda vista se pueden añadir y gestionar listas de tareas con sus correspondientes tareas.

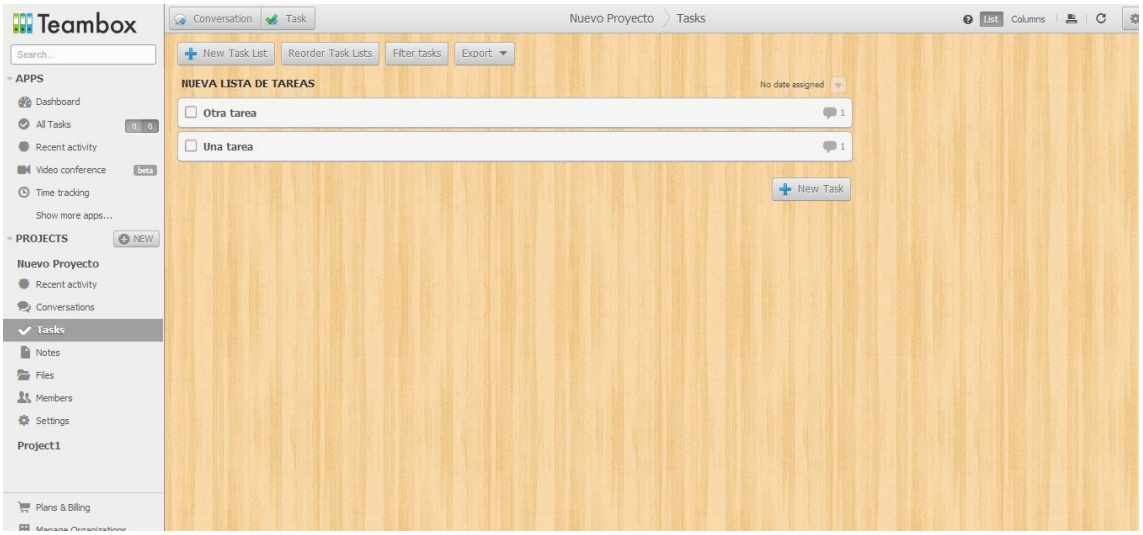


Ilustración 15: Vista tareas TeamBox

En la tercera vista se pueden añadir notas, gracias a un editor de texto integrado.

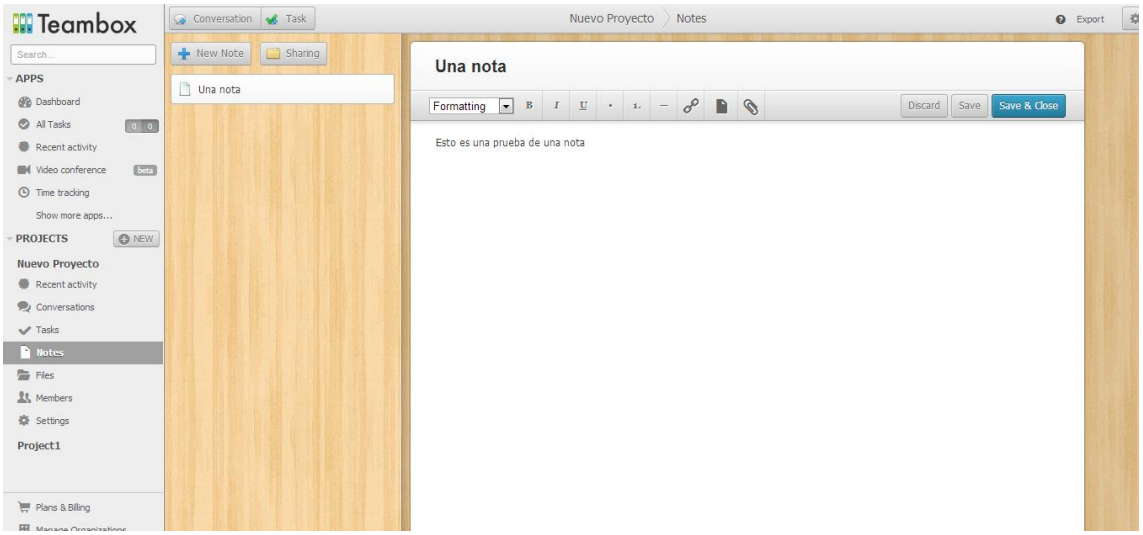


Ilustración 16: Creación nota TeamBox

Por último se ofrece una gestión de archivos.

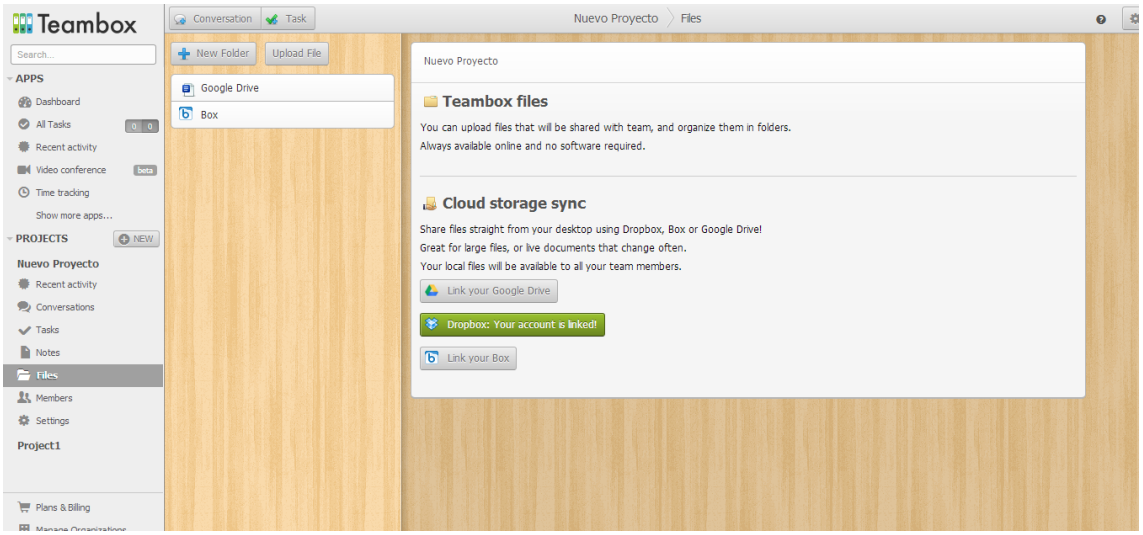


Ilustración 17: Archivos en la nube TeamBox

2.4.4 Do

Al igual que las anteriores aplicaciones analizadas Do es una aplicación de gestión de proyectos enfocada a entornos de trabajo.

Como anuncian en su página web Do, ofrece gestión de tareas, organización de proyectos, toma de notas, gestión de contactos y de tratos, grupos, conversaciones, envío de tareas por email y disponibilidad en varios dispositivos. Al igual que TeamBox, se ofrece en modalidad Freemium.

Para registrarse en la aplicación es posible crear una cuenta mediante la cuenta de **Google** o dando un email propio. Una vez que se accede a la aplicación principal, se pueden ver a los grupos a los que perteneces. Por cada grupo se pueden crear nuevos proyectos.

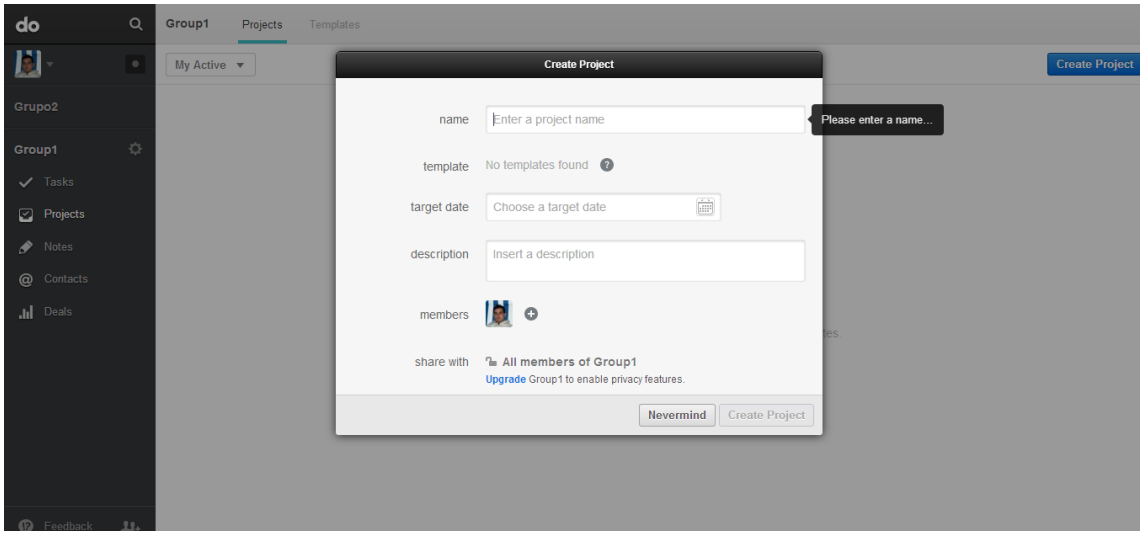


Ilustración 18: Nuevos Proyecto Do

Por cada proyecto, se podrá crear tareas en las cuales se pueden añadir discusiones o archivos procedentes de **Dropbox** o **Google** Drive.

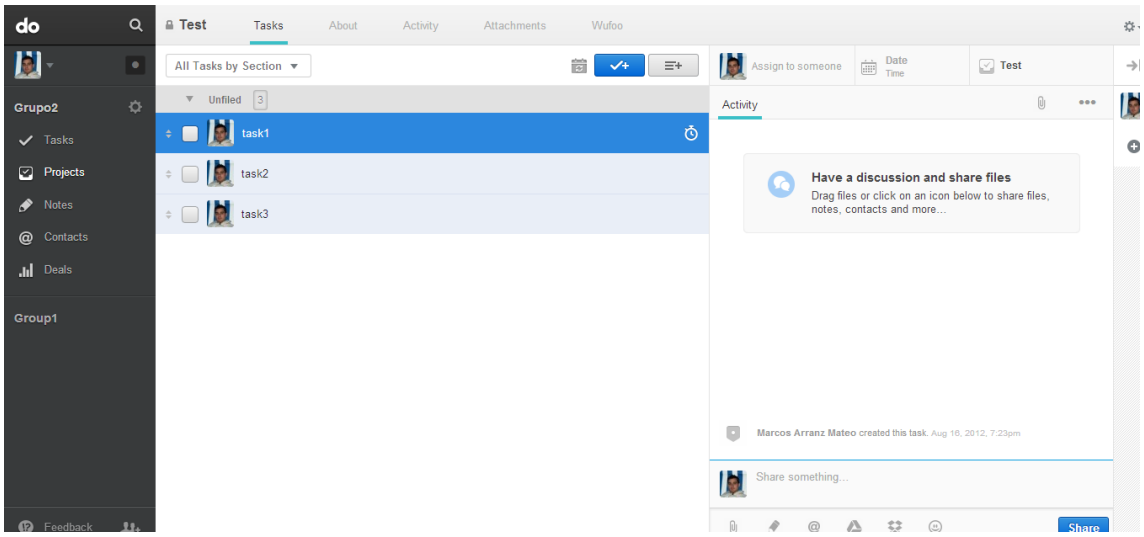


Ilustración 19: Tareas de un proyecto Do

Al igual que las anteriores aplicaciones se pueden añadir notas.

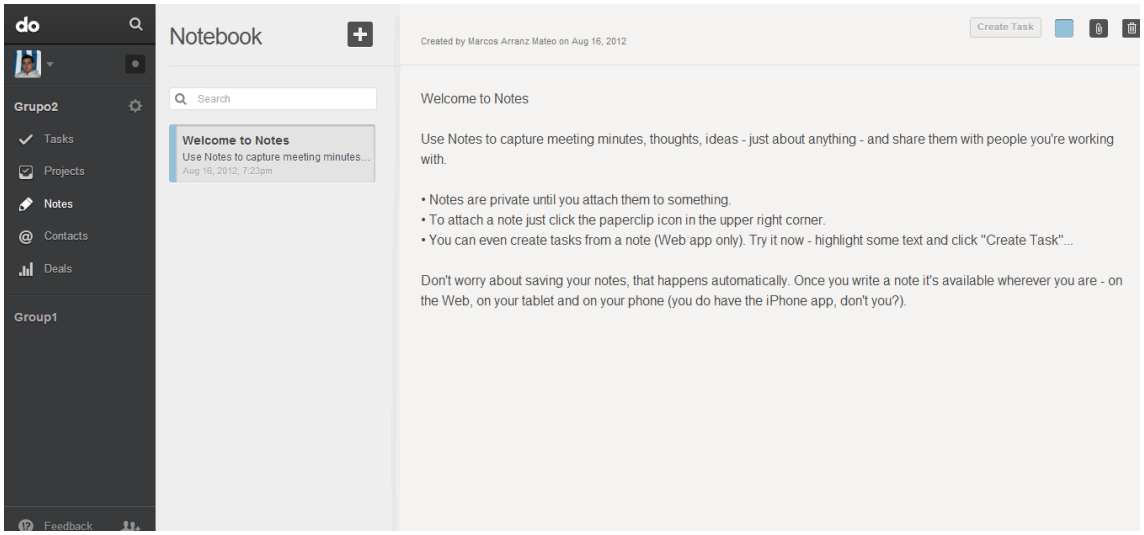


Ilustración 20: Notas en Do

Adicionalmente, ofrece la gestión de contactos y tratos:

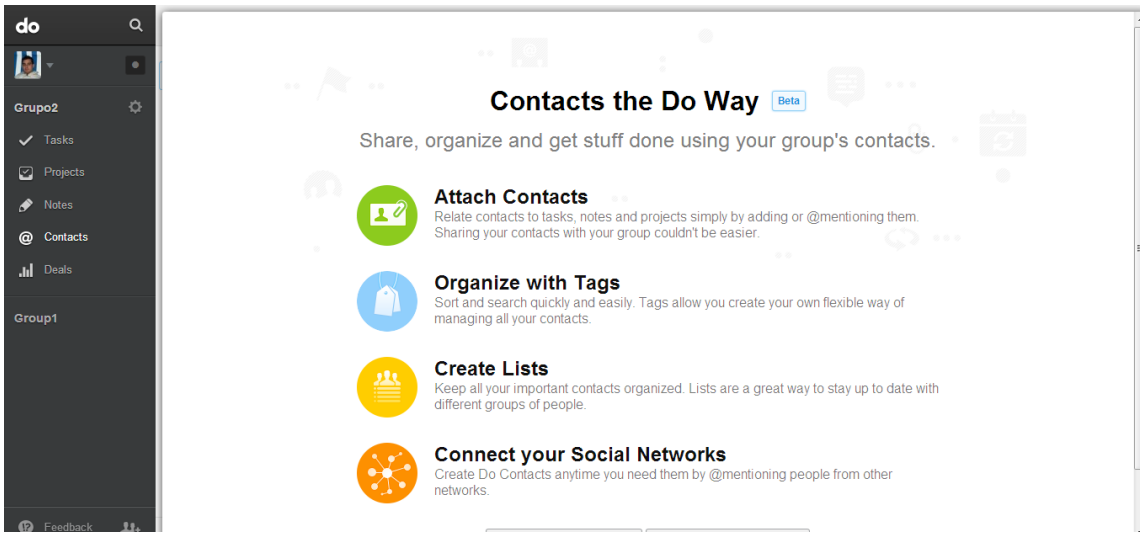


Ilustración 21: Contactos en Do

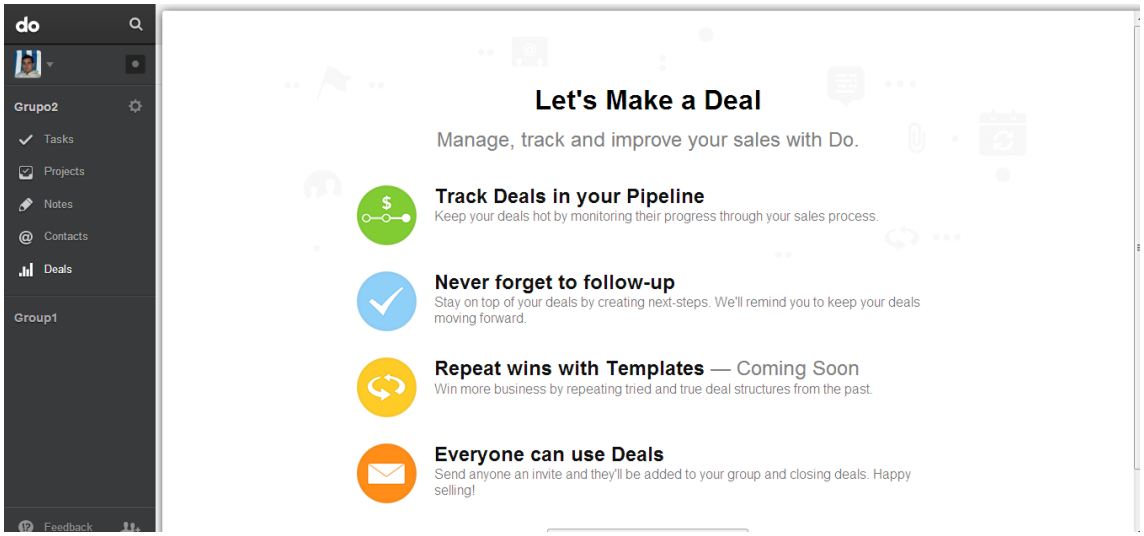


Ilustración 22: Deals en Do

2.4.5 Conclusión del análisis

Después de haber analizado diversas aplicaciones, se puede constatar que todas intentar sustituir o minimizar el uso del correo electrónico. A pesar de las grandes ventajas del correo tiende a introducir mucho ruido, siendo difícil coordinar el trabajo entre muchas personas.

En el caso de Wave, la intención es sustituir el email introduciendo un nuevo protocolo, de manera que la información esté disponible en un solo lugar y sea más dinámica. Sin embargo a pesar de la excelencia técnica de Wave, no logro el éxito esperado debido entre otras cosas a ser demasiado novedoso y no lo suficientemente intuitivo.

En el caso de las otras tres aplicaciones, el enfoque es parecido, pues están orientadas al entorno empresarial. El objetivo principal es llevar a realizar exitosamente un proyecto por parte de un equipo. Para ello, inciden en las funcionalidades para captar la información y la colaboración.

La aplicación que se pretende desarrollar no tiene como objetivo el entorno empresarial por lo que es más importante la sencillez y una interfaz intuitiva. El principal objetivo es poder compartir información de una forma sencilla, por encima de llevar a cabo algún trabajo. A diferencia de las redes sociales tradicionales, que están orientadas a crear redes o grupos de personas, en este caso lo que se pretende realizar estar orientados a algún hobby, afición o interés en común y compartir información sobre ello. Para poder visualizar la diferencia, en una red social como Google+, se podría crear un grupo “Amigos” en las que podrías compartir información con un grupo de amigos cercanos. El problema es si, por ejemplo, quieres organizar un viaje en el que no todos van a participar. Organizar un viaje no es una actividad profesional, pero requiere cierta dosis de organización.

3 ANÁLISIS, DISEÑO Y DESARROLLO.

En este capitulo se describe las distintas fases llevadas a cabo para realizar la aplicación.

3.1 INTRODUCCIÓN

En este documento se da una descripción del sistema que se desarrolla. La metodología de desarrollo escogida es Scrum, englobada dentro de las conocidas como metodologías ágiles. La razón de esta es elección es que las metodologías ágiles proveen de una flexibilidad mucho mayor, pues su enfoque es más adaptativo que predictivo en contraste con las metodologías convencionales. En la mayoría de los proyectos y en este en particular, los objetivos no siempre están claros y a medida que el producto se desarrolla algunas funcionalidades pasan a tener más importancia, y otras menos. Las metodologías ágiles esperan una planificación constante y además se adaptan mucho mejor a equipos pequeños, como es el caso.

Scrum por su parte, está especialmente indicada para proyectos en entornos complejos donde los requisitos son cambiantes o poco definidos, donde la innovación, la competitividad, la flexibilidad y la productividad son fundamentales.

El proceso de desarrollo de Scrum consiste en bloques temporales fijos y cortos, al final de los cuales se ha de entregarse un resultado completo. Este resultado ha de ser capaz de ser entregado sin mucha dificultad al cliente. En cada bloque temporal se seleccionan unas funcionalidades a realizar, basándose en las que más valor den al cliente. Por cada funcionalidad se realizará un ciclo completo incluyendo el análisis, diseño, implementación y comprobación.

Sin embargo Scrum, no provee una especificación de la planificación inicial por lo que se realiza una Iteración 0, donde se realizará un análisis ligero previo para comenzar a tener una lista con las funcionalidades priorizadas y ordenadas. También se realizará un diseño de ciertas características comunes a todas las funcionalidades, como es la elección de patrones de arquitectura, las tecnologías a ser usadas, etc.

3.1.1 Descripción general

El objetivo principal de la aplicación es ofrecer un servicio para la organización de proyectos tanto en solitario como compartidos con otras personas. A diferencia de otras aplicaciones, el objetivo no es ofrecer un sistema rígido orientado a proyectos profesionales sino un sistema directo donde puedas organizar y compartir información.

Los objetivos serán proveer un sistema para poder añadir nuevos proyectos a tu cuenta. Cada proyecto, tendrá diversas vistas para separar y organizar la información. La vista principal será una vista donde se puede actualizar el estado e introducir nueva información. Esta información puede ser simplemente texto o un link web. Esta información puede ser comentada por otros usuarios o por el mismo usuario. Además de esto, la información contendrá un menú contextual donde se pueden añadir tanto ficheros, como tareas o notas asociadas a esa información.

Una segunda vista mostrará las tareas del proyecto. En esta vista habrá tareas finalizadas y por finalizar. Una tarea representa una actividad que se debe llevar a cabo, con unas características asociadas tales como lugar, fecha, descripción, etc. La tercera vista mostrará una vista con notas asociadas al proyecto. Estas notas servirán para anotar de un método detallado información

relacionada con el proyecto. Además se ofrecerá la posibilidad de sincronizar las notas con el popular servicio **Evernote**, especializado en la gestión y captura de notas.

La última vista permite unir el servicio con la aplicación **Dropbox**. Una vez que se ha concedido el permiso a **Dropbox**, se podrán subir archivos y compartirlos con otros usuarios.

Los usuarios potenciales de este producto serán personas de edad joven o media, con experiencia en el uso de herramientas online pero no necesariamente con conocimientos técnicos. Al ser un proyecto libre, no existe en principio ninguna restricción que podría ser impuesta por el cliente, y es el equipo de desarrollo el que elegirá las herramientas que mejor se adapten. La única dependencia impuesta por los requisitos será el hacer uso de un servicio externo de alojamiento de archivos, pues el alojamiento en servidores propios puede ser costoso.

Todo el código se almacenara en **Github en el repositorio: https://github.com/Markinhos/Drawer y se mantendrá una version online en el dominio https://simpledesk.herokuapp.com/.**

3.2 ANÁLISIS

A continuación se describe el análisis de los requisitos necesarios para llevar a cabo satisfactoriamente la aplicación.

3.2.1 Introducción

El objetivo del análisis previo es identificar los objetivos de negocio principales. A diferencia con las metodologías tradicionales, se hace una visión a muy alto nivel con no excesiva documentación. Una vez tengamos los objetivos principales se realizará una lista de historias de usuario, es decir, objetivos que dan valor al cliente, con el que rellenar nuestro *Product Backlog*. Previamente estas historias de usuario se han priorizado y estimado de tal forma que las primeras historias en ser realizadas son las que den más valor. De este modo, el producto avanza desde más valor a menos. En cualquier caso, este *Product Backlog* inicial es tan solo el punto de partida, pues se ha de estar preparado para modificaciones de requisitos, nuevas funcionalidades, re priorizaciones, etc.

Epics

En primer lugar vamos se recogen los objetivos a alto nivel, llamados también Epics en la metodología Scrum. Para cada uno de estos Epics se va a crear un caso de uso sencillo, en el que se identifiquen las actividades más generales.

Acceso a la aplicación

- Usuario se registra en la aplicación
- Usuario accede a la aplicación con sus credenciales

Gestión de tarea

- Usuario añadir tarea
- Usuario edita tarea
- Usuario borra tarea
- Usuario completa una tarea

Gestión de estados

- Usuario añade estado
- Sistema identifica el link
- Usuario borra estado
- Usuario añade un comentario

Gestión de notas

- Usuario crea una nota
- Usuario edita una nota
- Usuario borra tarea

Gestión de ficheros

- Usuario añade un archivo
- Usuario borra un archivo
- Usuario refresca
- Usuario sube varios archivos a la vez.

Opciones de usuario

- Usuario cambia contraseña
- Usuario revoca acceso de servicios de terceros
- Añadir avatar a perfil

Compartir proyecto

- Usuario comparte proyecto con otro usuario

Sincronizar notas con servicio terceros

- Crea una nota en el servicio de terceros
- Sistema comprueba actualizaciones
- Botón refrescar
- Sistema comprueba de forma periódica actualizaciones
- Al borrar nota se borra en servicio externo

3.2.2 Historias de usuario

A partir de los hitos se van a crear las historias de usuario iniciales para empezar a construir el *Product Backlog*. Por el momento se van a crear historias de usuario informales, donde tan solo se especifique el objetivo de la funcionalidad, una estimación de la complejidad y un valor para el cliente. De esta forma siguiendo los principios agiles, aplazamos el análisis pormenorizado para cuando el desarrollo este más cercano.

- Registrar en la aplicación
- Acceso a la aplicación
- Añadir proyecto
- Añadir tarea
- Editar tarea
- Borrar tarea
- Completar tarea
- Añadir estado
- Borrar estado
- Añadir comentario
- Añadir nota
- Editar nota
- Borrar nota
- Añadir fichero a la nube
- Borrar fichero de la nube
- Refrescar lista de ficheros en la nube
- Cambiar contraseña
- Usuario revoca acceso de servicios de terceros
- Añadir avatar
- Compartir proyecto
- Sincronizar nota al crearse
- Sincronizar nota al editarse
- Sincronizar nota al borrarse
- Refrescar las notas con los cambios.
- Sincronizar archivo al crearse
- Sincronizar archivo al borrarse

A continuación se va a estimar y priorizar las historias de usuario. Para la estimación se va a utilizar una serie de puntos, de la serie de Fibonnaci: 1, 2, 3, 5, 8. Se ha escogido esta escala para que no se pierda demasiado tiempo en realizar estimaciones excesivamente precisas. Para la prioridad se utilizara el sistema MoSCoW (*Must*, *Should*, *Could*, *Won't*). Las historias de usuario son priorizadas por el cliente, pues es el que conoce mejor que funcionalidades le dan más valor. Por tanto una historia el valor seria el siguiente:

- *Must Haves*: son funcionalidades que han de incluirse antes de que el producto sea entregado.
- *Should Haves*: son funcionalidades que no son críticas pero aportan un valor alto.
- *Could Haves*: son funcionalidades que se han de incluir si no incurren en un esfuerzo muy alto. En caso de problemas con la fecha límite, son las primeras funcionalidades en ser eliminadas
- *Won't Haves*: son funcionalidades que se han requerido pero que no se incluirán en la entrega pero se añadirán a lo largo de un desarrollo posterior.

Nombre	Prioridad	Complejidad
Registro aplicación	*Must*	5
Acceso aplicación	*Must*	5
Añadir proyecto	*Must*	5
Añadir tarea	*Must*	5
Editar tarea	*Should*	3
Completar tarea	*Must*	2
Añadir estado	*Must*	5
Borrar estado	*Could*	1
Añadir comentario	*Could*	5
Añadir nota	*Must*	5
Editar nota	*Should*	3
Borrar nota	*Should*	1
Añadir fichero	*Must*	8
Borrar fichero	*Could*	2
Subir varios archivos a la vez	*Should*	8
Refrescar lista de ficheros en la nube	*Should*	5
Cambiar contraseña	*Should*	2
Usuario revoca acceso de servicios de terceros	*Won't*	3
Añadir avatar	*Won't*	5
Compartir proyecto	*Should*	5
Refrescar proyecto	*Should*	5
Sincronizar nota al crearse	*Could*	8
Sincronizar nota al editarse	*Could*	3
Sincronizar nota al borrarse	*Won't*	2
Refrescar las notas con los cambios	*Could*	5

Tabla 2: Product Backlog sin priorizar

El *Product Backlog* nos quedaría de esta forma:

Nombre	Prioridad	Complejidad
Registro Aplicación	*Must*	5
Acceso aplicación	*Must*	5
Añadir proyecto	*Must*	5
Añadir tarea	*Must*	5
Añadir estado	*Must*	5
Añadir nota	*Must*	5
Añadir fichero	*Must*	8
Borrar tarea	*Should*	1
Borrar nota	*Should*	1
Completar tarea	*Should*	2
Cambiar contraseña	*Should*	2
Editar tarea	*Should*	3
Editar nota	*Should*	3
Refrescar lista de ficheros en la nube	*Should*	5
Compartir proyecto	*Should*	5
Refrescar proyecto	*Should*	5
Subir varios archivos a la vez	*Should*	8
Borrar estado	*Could*	1
Borrar fichero	*Could*	2
Sincronizar nota al editarse	*Could*	3
Añadir comentario	*Could*	5
Sincronizar nota al crearse	*Could*	8
Sincronizar nota al borrarse	*Won't*	2
Usuario revoca acceso de servicios de terceros	*Won't*	3
Añadir avatar	*Won't*	5

Tabla 3: Product Backlog priorizado

Se ha estimado el tamaño de cada Scrum en 1 mes y la velocidad relativa serán 14 puntos de historia. Estos puntos no tienen una correspondencia directa con horas, sin embargo son a medida que se avancen en las iteraciones se obtendrá una capacidad más precisa de la capacidad de trabajo. La razón de dar tan solo 14 puntos se debe a una baja disponibilidad en horas, la poca familiaridad con las tecnologías a ser usadas y lo novedoso de estas, lo que puede dar problemas en forma de bugs.

Iteración 1

Registro Aplicación	4	5
Acceso aplicación	4	5

Iteración 2

Añadir proyecto	4	5
Añadir tarea	4	5

Iteración 3

Añadir estado	4	5
Añadir nota	4	5

Iteración 4

Añadir fichero	4	8
Borrar tarea	3	1
Borrar nota	3	1
Completar tarea	3	2
Cambiar contraseña	3	2

Iteración 5

Editar tarea	3	3
Editar nota	3	3
Refrescar lista de ficheros en la nube	3	5

Iteración 6

Compartir proyecto	3	5
Subir varios archivos a la vez	3	8
Borrar estado	2	1

Iteración 7

Borrar fichero	2	2
Sincronizar nota al editarse	2	3
Añadir comentario	2	5

Iteración 8

Refrescar las notas con los cambios.	2	5
Sincronizar nota al crearse	2	8
Usuario revoca acceso de servicios de terceros	1	3

Iteración 9

Sincronizar nota al borrarse	1	2
Añadir avatar	1	5

Por tanto, según la planificación inicial tendríamos que el proyecto finalizaría en 9 meses.

Se harán 3 entregas:

1. La primera de ellas será al finalizar la iteración 4, y comprenderá las funcionalidades relacionadas con la gestión básica de estados, tareas, notas y ficheros.
2. La segunda incluirá la posibilidad de compartir el proyecto con otros usuarios. Se completara al finalizar la iteración 6.
3. La última entrega incluirá las funcionalidades relacionadas con la sincronización de servicios. Se entregara en la última iteración.

Una vez que tenemos recopilados los requisitos de la aplicación se puede hacer una estimación tanto en tiempo como en presupuesto necesario para llevar a cabo el proyecto. En los anexos II y IV se puede ver el presupuesto necesario y el diagrama de Gantt del proyecto.

3.3 DISEÑO

A continuación se describe el proceso de diseño de la aplicación.

3.3.1 Introducción

En esta fase se va a realizar un diseño de los conceptos a más alto nivel de la aplicación. EL objetivo no será hacer un diseño pormenorizado sino diseñar las características principales que nos permitan tomar las decisiones más acertadas en lo referente a aspectos que afecten a todo el desarrollo posterior como puede ser la elección de las tecnologías y herramientas a usar o la arquitectura. Siguiendo la filosofía ágil, el diseño pormenorizado se retrasara lo máximo posible, evitando riesgos, ya que se conocerá mejor los detalles necesarios.

3.3.2 Prototipos

Para tener un punto de partida sobre la interfaz de usuario se va a realizar una serie de prototipos de diferentes partes de la interfaz. El objetivo es realizar una serie de bocetos a bajo nivel para tener claro las distintas vistas. Se han realizado cuatro bocetos, correspondientes a las principales vistas.

La primera vista corresponde al panel de estados. En esta vista se puede ver como existe una serie de proyectos que el usuario ha creado previamente y dentro de uno de los proyectos, en la sección de actividad cada usuario ha escrito un estado o un comentario.

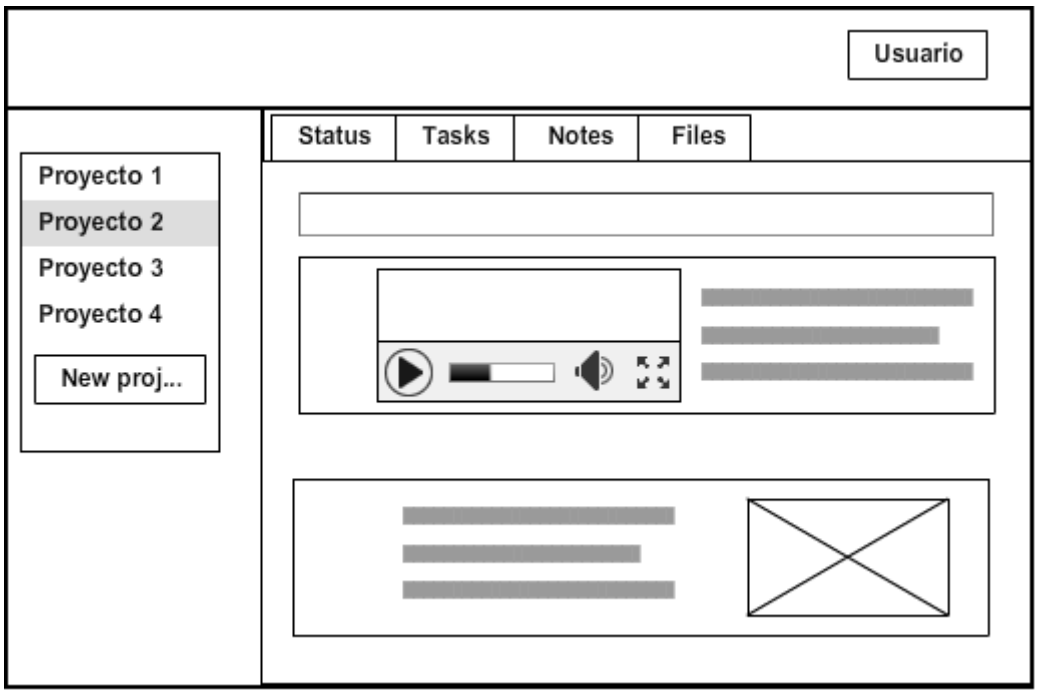


Ilustración 23: Vista Estados

En la segunda vista se puede ver la sección de tareas. En ella se ve una lista de tareas por hacerse y unas tareas ya completadas.

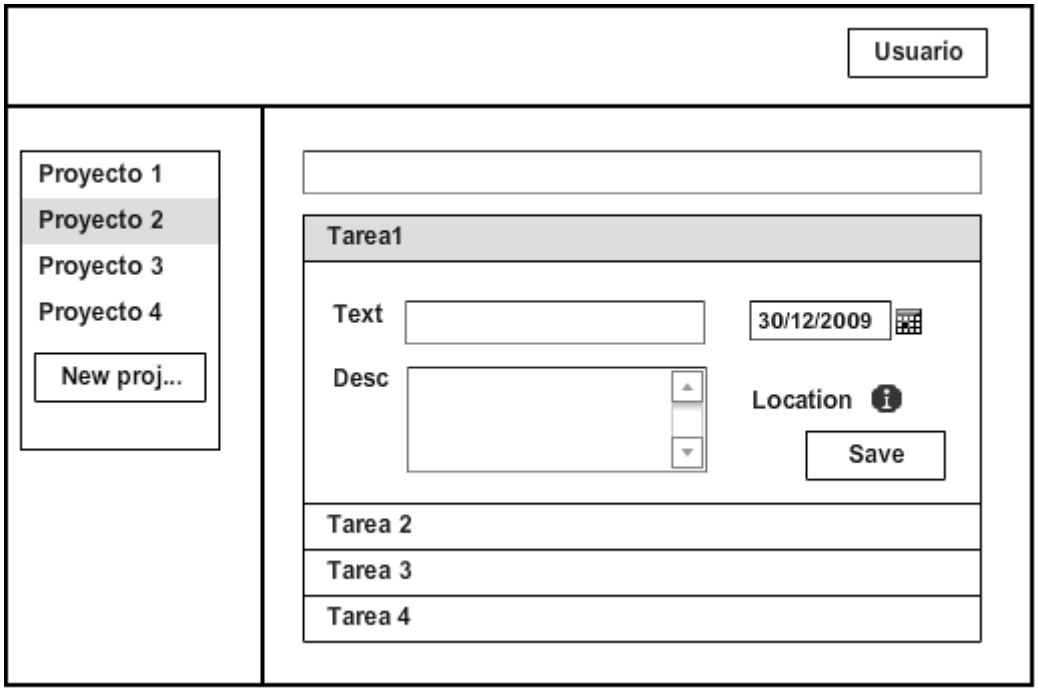


Ilustración 24: Vista Tareas

En la tercera vista, se puede ver una serie de notas, junto con un editor. Este editor servirá para crear nuevas notas o editar las ya existentes.

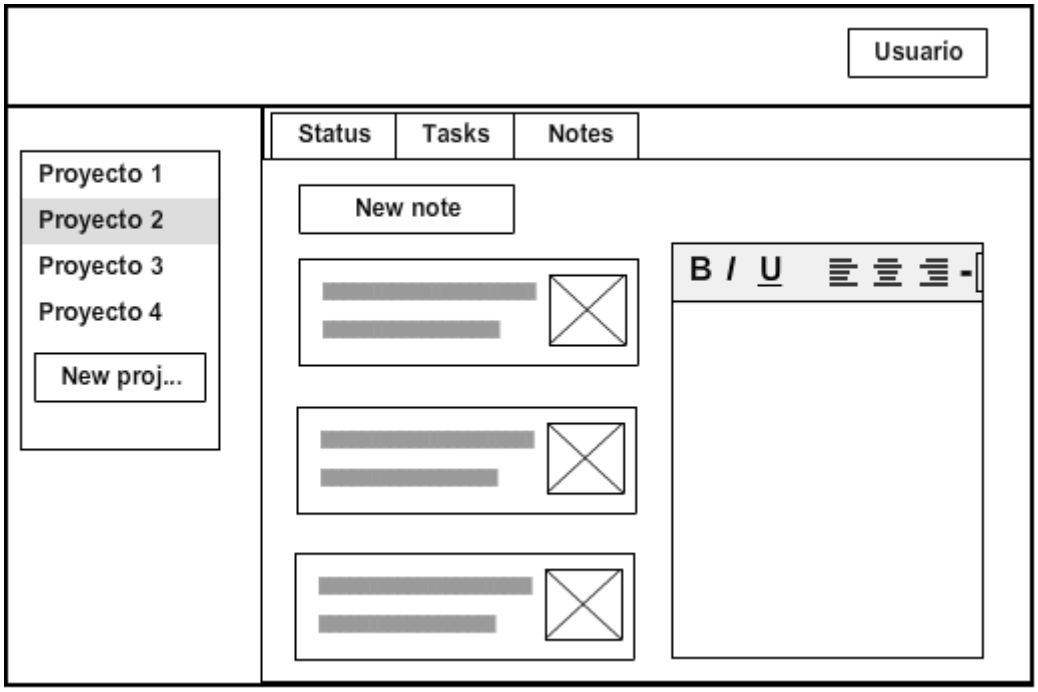


Ilustración 25: Vista Notas

Por último, en la cuarta vista, se puede ver la sección de ficheros. En esta se pueden ver que el usuario dispone de varios ficheros de distintos tipos que previamente ha subido a su cuenta.

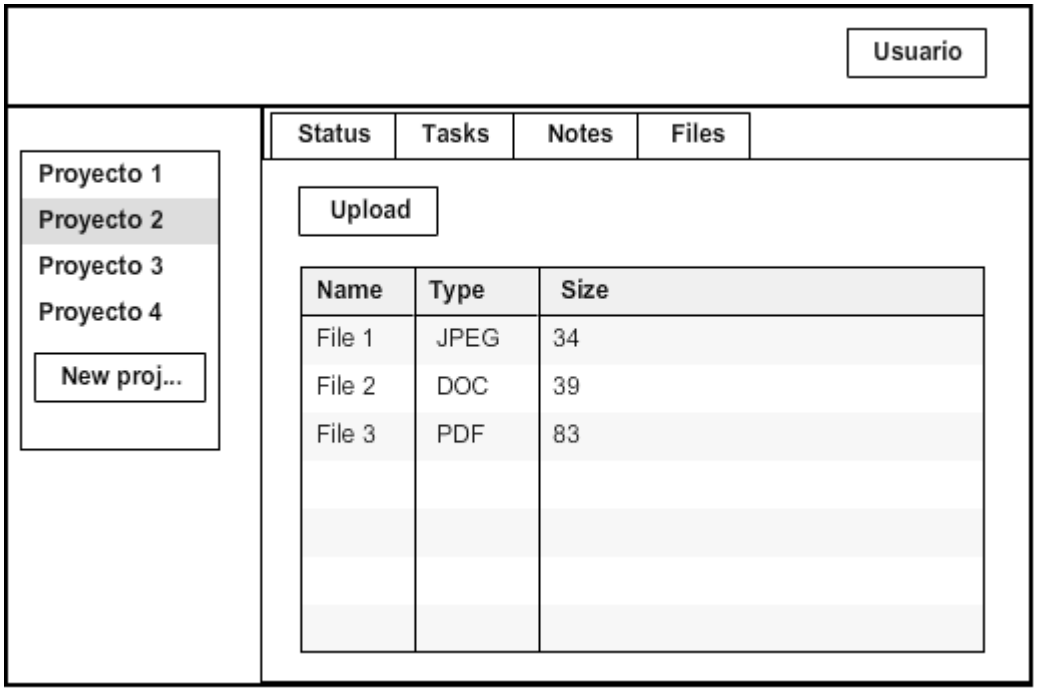


Ilustración 26: Vista Ficheros

3.3.3 Modelo de dominio

Para tener una idea general del dominio que se va a modelar se ha creado un diagrama con los conceptos más importantes y sus relaciones. Este modelo no ha de ser completo, y puede variar durante el desarrollo de la aplicación, pero sirve como base para empezar el desarrollo clarificando ideas y conceptos.

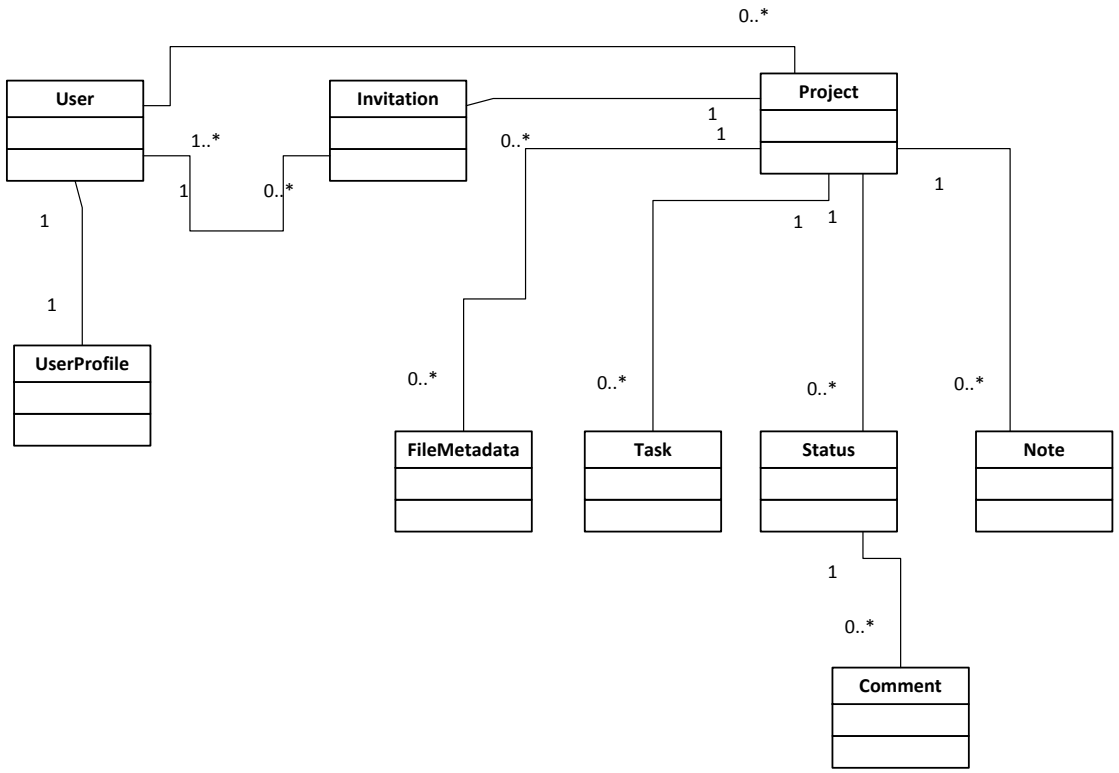


Ilustración 27: Modelo de Dominio

Como se puede apreciar, la entidad más importante es Project. Este documento contendrá el Proyecto entero, pues de él dependen el resto de componentes. Al estar utilizando una base de datos basada en documento, Project no se corresponde con una tabla sino con un documento. Este documento contendrá diversas entradas, de una forma no necesariamente estructurada. Entre esas entradas se encuentran Task, Status, Note y FileMetadata. Cada uno de ellos representa las entidades más importantes que se van a procesar en la aplicación, y a su vez contendrán más campos que no se han especificado por simplicidad. Status contendrá una entidad llamada Comment, para representar los comentarios hechos sobre un estado. Además del documento Project, existirán tres documentos más: User, UserProfile e Invitation. User representa al usuario, con atributos necesarios para el sistema como, correo electrónico, permisos, contraseña, etc. y UserProfile representa las preferencias de la cuenta de usuario. Estos dos documentos tienen una correspondencia 1:1. Por último, Invitation es un documento para recoger las invitaciones creadas de un usuario a otro para un proyecto concreto.

3.3.4 Arquitectura

La solución más adecuada a los requisitos tomados corresponde a una aplicación SPA (single page application). Este modelo corresponde a una página auto-contenida, donde realiza una carga inicial de todos los componentes necesarios para la ejecución. Una vez cargados los componentes esta se comunicara con el servidor mediante llamadas AJAX. De esta forma tenemos lo que se conoce como "thick client", es decir, un cliente que es responsable de una parte muy importante de las funcionalidades de la aplicación. Al tener un cliente complejo, se optara por utilizar un patrón de arquitectura MVC o similar, para favorecer un desarrollo correcto de la parte del cliente.

El servidor por su parte, tendrá que ofrecer un API RESTful, para poder interaccionar con el cliente. Para esto, también se optara por una arquitectura MVC. En el caso de la vista en vez de HTML, como tradicionalmente es, se optara por exponer los recursos en formato JSON. De esta forma se consigue una separación entre el servidor y el cliente, lo que permitiría en un futuro crear una aplicación móvil reutilizando el servidor, siempre y cuando la aplicación móvil respete la interfaz del servidor.

3.3.5 Entorno de desarrollo

ELECCIÓN DE LA BASE DE DATOS:

La elección de la base de datos siempre es un punto importante. En nuestro caso los datos que tendremos, van a ser unos datos desestructurados para una aplicación web. La operación más repetida va a ser la operación de lectura, ya que los usuarios aunque obviamente pueden introducir nuevos datos, esto no se hace de forma intensiva si no que se realizarán más operaciones de consulta que de escritura. Además el modelo de dominio no presenta muchas relaciones entre las entidades.

MongoDB parece una solución ideal para este caso de uso, ya que no presenta una estructura de documentos fija. Al contrario que las bases de datos tradicionales no existen tablas con campos fijos, sino que la información se almacena en documentos BSON, esto es JSON binario. De esta forma un documento usuario, por ejemplo, puede tener una cantidad distinta de campos que otro usuario. Este tipo de base de datos se conoce como bases de datos orientadas a documento. Además posee una velocidad de lectura y escritura muy superior a la de las bases de datos tradicionales. Por contra no soporta transacciones, ni JOINs. La falta de JOINs puede ser un problema si no se modela correctamente el dominio, pues cualquier relación N:N ha de controlarse manualmente por código. Ademas las funciones de agregación, como era de esperar, son más lentas.

	MySQL	MongoDB
Leer todos los objetos con id entre 100 y 110 millones	10m 5s	0,15s
Seleccionar un registro por id (entre 113 millones)	0,30s	0,12s
Seleccionar los ids de los registros entre dos fechas	4m 10s	0,16s
Seleccionar los ids de los registros entre dos fechas y contarlos	2m 48s	0,85s

Tabla 4: Test de Lectura

	MySQL	MongoDB
70.000 usuarios	12s	3s

1.300.000 dominios	4m 36s	58s
1.300.000 dominios con índices	14m 13s	8m 27s
5.000.000 de entradas en el log	2h 10m 54s	1h 03m 53s
10.000.000 de entradas en el log	3h 27m 10s	1h 59m 11s
30.000.000 de entradas en el log	10h 18m 46s	5h 55m 25s

Tabla 5: Test De Escritura

	MySQL	MongoDB
10 dominios más visitados con totales de visitas	2m 37s	13m 13s
10 dominios más visitados en la segunda quincena de Junio	17m 43s	52m 39s
10 usuarios con más accesos	3m 53s	24m 02s
Media de tráfico en Junio	2m 42s	12m 05s

Tabla 6: Test de Agregacion

Como se puede observar para test de escritura y lectura MongoDB es por regla general el doble de rápido. Sin embargo en el test de agregación MySQL supera ampliamente el rendimiento de MongoDB. Esto ha de estar presente a la hora del diseño de las entidades buscándose siempre estructuras en forma de árbol, y evitando las relaciones N:N entre estas en la medida de lo posible.

ELECCIÓN DEL ENTORNO DE DESARROLLO WEB

El lenguaje de elección para el cliente es bastante directo, JavaScript, pues es el único lenguaje aceptado por todos los navegadores web modernos. Como se ha comentado se va a construir un cliente complejo, por lo tanto será muy útil de hacer uso de alguno de los *frameworks*. De entre todos los *frameworks* que estructuran el código se ha escogido Backbone.js pues es el *framework* más maduro en estos momentos. Backbone.js proporciona una interfaz RESTful, con un patrón basado en el paradigma MVP. Es un *framework* muy ligero, con tan solo una dependencia fuerte: Underscore.js, una librería que proporciona una serie de utilidades sobre JavaScript. Además se hará uso de jQuery para la manipulación del DOM

PYTHON Y DJANGO, TASTYPIE:

Para la parte del servidor se ha optado por el lenguaje Python para el desarrollo. Python es un lenguaje maduro, que ofrece una gran flexibilidad y una gran productividad. Como *framework* se ha escogido una versión modificada de Django 1.3. Django es el *framework* MVC más popular para Python. La versión modificada que se ha escogido es una versión que permite conservar muchas de las funciones ORM de Django para las bases de datos no relacionales. Como conector interno con MongoDB Django-MongoDB utiliza PyMongo.

Tastypie va a ser el *framework* escogido para exponer los recursos del API RESTful. Tastypie es el *framework* más importante en la comunidad de Python para exponer una API REST, y el más activo.

SERVICIOS DE SINCRONIZACIÓN

Para sincronizar los ficheros se ha optado por **Dropbox**, debido a que actualmente es uno de los servicios más populares y ofrece una integración sencilla con su API, mediante un componente en **Python**.

Para la sincronización de las notas se ha elegido **Evernote** por las mismas razones que se ha elegido **Dropbox**. Es la herramienta más popular en su campo y se dispone de un componente en **Python**, lo cual facilitara la integración.

PLATAFORMA DE DESARROLLO

Como entorno de desarrollo se ha escogido **Heroku**. **Heroku** proporciona un servicio donde la capacidad está bajo demanda. En este caso nuestra demanda durante el desarrollo será baja, y solo necesitaremos una pequeña instancia para mostrar al cliente las nuevas versiones. Aparte de esto, **Heroku** ofrece como grandes ventajas una administración muy sencilla, por lo que libera al desarrollador de muchas actividades de administración del servidor por lo que se puede permitir enfocarse más en el desarrollo. Además, el despliegue de las nuevas versiones es automático y seguro. Con un solo comando se puede poner en producción una nueva versión del producto. Por otra parte existe una gran variedad de add-ons para ser añadidos. Para hacer un uso sencillo de la base de datos se hará uso del add-on MongoLab, el cual ofrece una instancia **MongoDB**.

Para el almacenamiento de archivos estáticos como imágenes, ficheros de **CSS**, **JavaScript** o cualquier otro archivo que no sea necesario ejecutar en el servidor se hará uso de **Amazon** S3.

Para monitorizar el rendimiento de la aplicación se hará uso de un complemento de **Heroku** llamado **NewRelic**.

OTROS

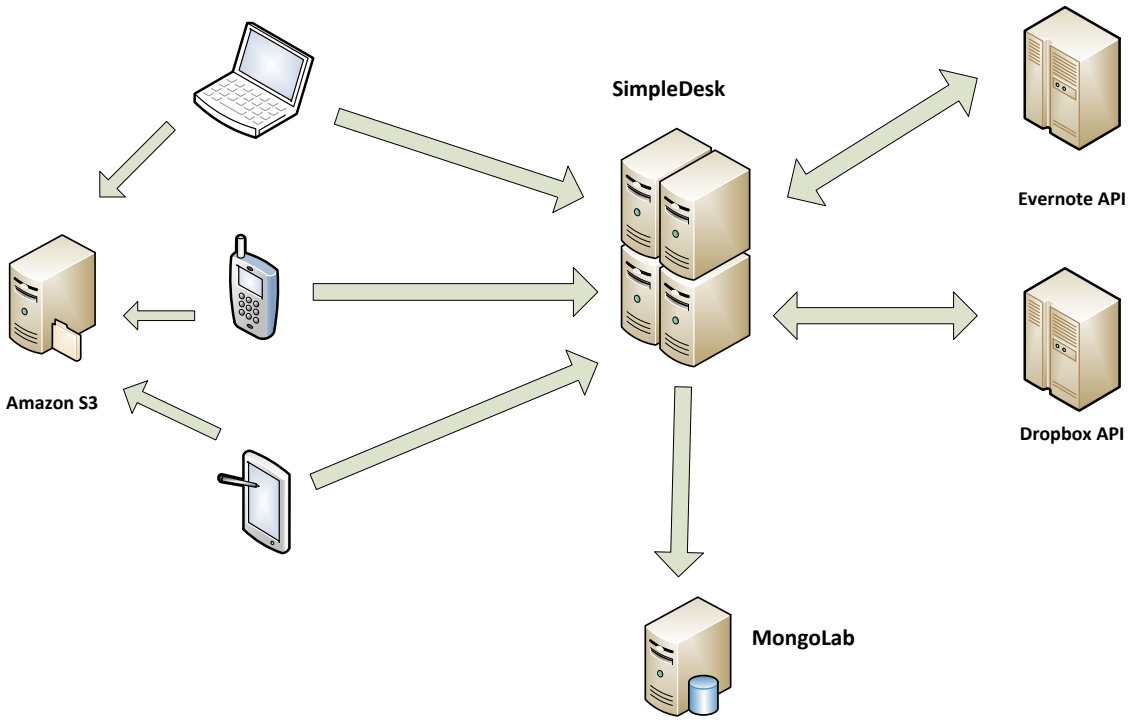
En otras categorías se utilizara como herramienta para el control de versiones **Git**. **Git** es el gestor de controles distribuido más popular en la actualidad. Además ofrece una integración muy sencilla con **Heroku**. Aunque no es estrictamente necesario, para el desarrollo, se va a utilizar una plataforma de almacenamiento del código. En este caso se ha decidido utilizar **GitHub**, el cual es un repositorio para código, utilizando la herramienta **Git**.

Para la gestión del proyecto de una forma ágil, se utilizara Pivotal Tracker.

3.3.6 Diagramas

En esta sección se van a exponer algunos diagramas que describen la arquitectura que se va a construir.

A continuación se puede ver un diagrama simple de la arquitectura general de la aplicación.



El servidor de aplicaciones recibirá peticiones de diversos dispositivos. Para acceder a los datos guardaros se ha de conectar al servidor de datos MongoLab. Para servir los datos estáticos el documento HTML servido por la aplicación contendrá la dirección de un servidor.

A continuación se expone un diagrama de despliegue UML:

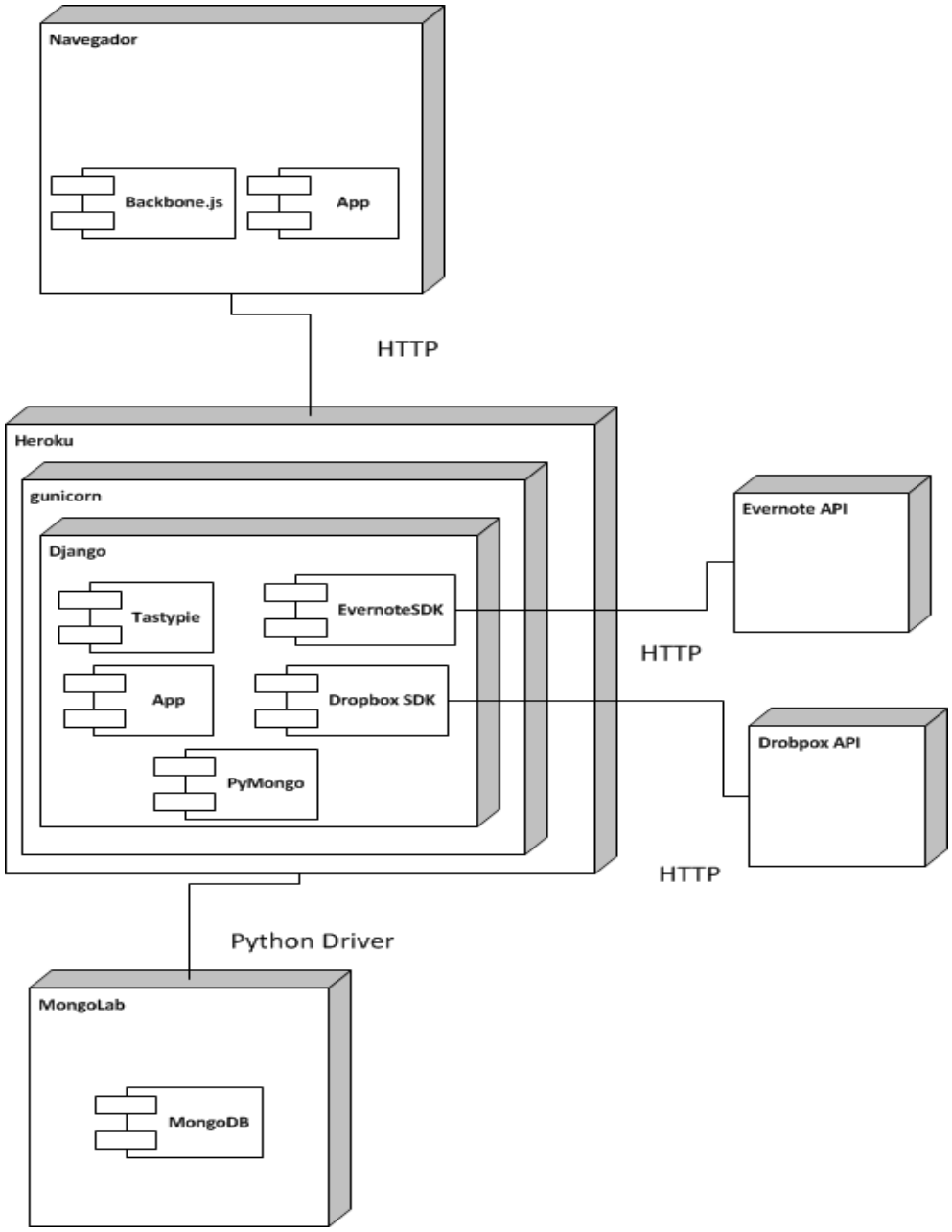


Ilustración 28: Arquitectura Aplicación

Como se puede ver, la aplicación se aloja en Heroku. A su vez, la aplicación será ejecutada por el servidor de aplicaciones Gunicorn. Dentro la propia aplicación se utilizara varias aplicaciones y *frameworks*. Los mas relevantes son Django, pues dota a toda la aplicación de una estructura MVC, Tastypie que expondrá una interfaz REST a partir de los modelos de la aplicación, y los SDK de Evernote y Dropbox. La aplicación se comunicara con la base de datos, alojada en MongoLab, mediante un driver de Python, PyMongo. La comunicación con las APIs de Dropbox y Evernote será a través de HTTP plano, pues son APIs REST. La comunicación con el usuario, será sobre HTTP, y la aplicación será encargada de entregar el código que se ejecutara en el cliente.

A su vez, la aplicación se ejecutara en los servidores de **Heroku**. **Heroku** por su parte tiene su propia arquitectura. En este diagrama se puede ver el comportamiento de este:

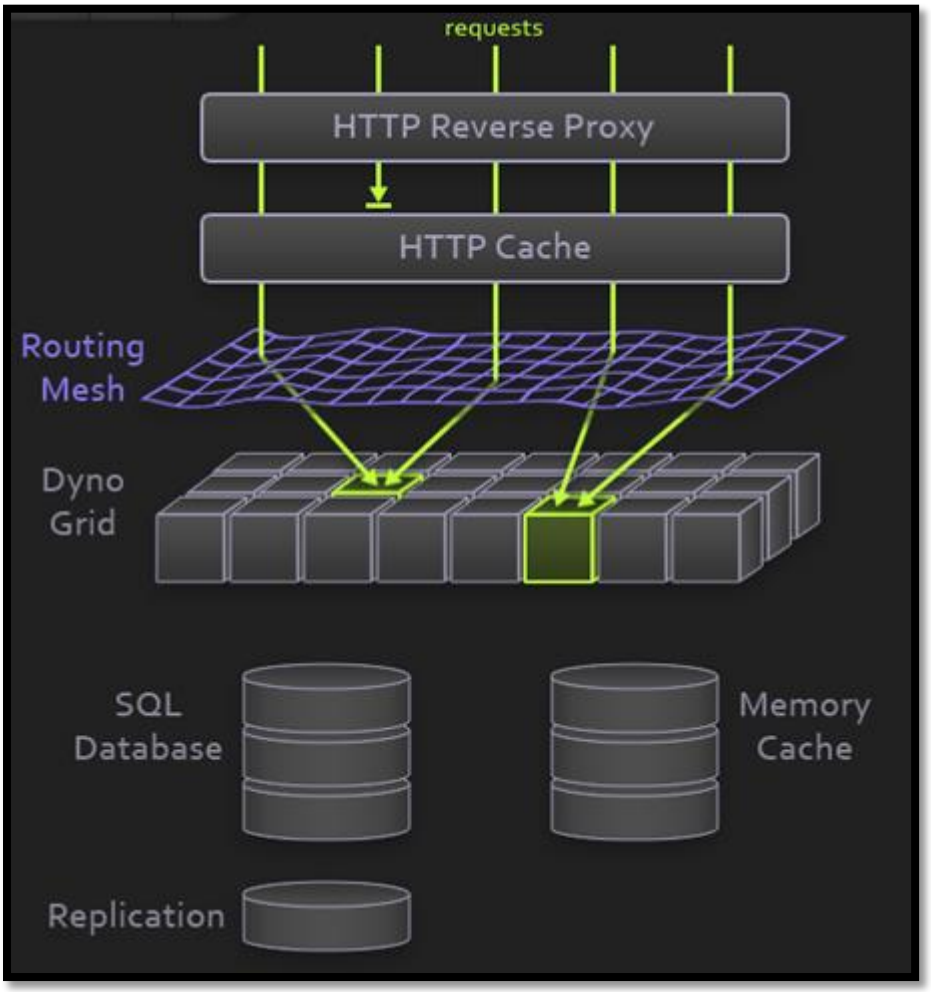


*Ilustración 29: Arquitectura **Heroku***

Las llamadas del cliente llegan a los servidores reversos (**NGINX**). A continuación la llamada es reenviada a la capa cache. Si la llamada se encuentra en la capa cache, es devuelta inmediatamente. Si no se encuentra en la capa cache, busca un proceso servidor (llamada *dyno*) que sea capaz de dar respuesta a la petición y si no hay ninguno lo crea. Este proceso servidor es el que ejecutara el código de la aplicación.

3.4 DESARROLLO

Una vez finalizado un análisis y diseño inicial se comienza el desarrollo de las iteraciones. Como se ha comentado en la introducción, en Scrum se realiza una fase completa del ciclo de vida del desarrollo por cada funcionalidad, en este caso historias de usuario. Por tanto en este apartado se describirá el análisis y diseño por cada historia de usuario, y si procede existirá una sección de desarrollo donde se mostrará el código implementado.

Para las iteraciones vamos a desarrollar las historias de usuario, haciéndolas más formales y con más detalle. Los campos que se van a rellenar son estos.

ID	
OBJETIVO	
SECUENCIA	
COMPLEJIDAD	
PRIORIDAD	
PRUEBA DE ACEPTACIÓN	

Tabla 7: Plantilla Historia de Usuario Formal

- ID - Es el identificador único de cada historia. Tiene un formato HU-XXX
- Objetivo - la funcionalidad esperada. Se escriben con el formato: Quiero <funcionalidad> para <objetivo>
- Secuencia - Los pasos que ha de dar el usuario para llevar acabo la funcionalidad
- Complejidad - La complejidad estimada de la historia en puntos relativos. Estos puntos seguirán una distribución de Fibonacci de 1 a 8.
- Prioridad - La prioridad en formato *MoSCoW*.
- Prueba de aceptación - Las condiciones para que se dé por válida la funcionalidad

Para cada historia se especificaran las tareas relacionadas con su desarrollo. En el caso de historias de usuario complejas se puede hacer uso de apoyos como diagramas UML o fichas CRC. Las fichas CRC (*Class Responsibility Collaborator*) son tarjetas donde se escriben las entidades involucradas, anotando cuáles son sus responsabilidades y con qué otras entidades colaboran. Seguirán un modelo como el que se muestra a continuación:

Nombre clase	
Responsabilidad	Colaboradores

Tabla 8: Ficha CRC

3.4.1 Iteración 1

ANÁLISIS

En la fase de análisis se había priorizado las historias de usuario, y para la primera iteración tenemos las siguientes historias para realizar.

Nombre	Prioridad	Complejidad
Registro Aplicación	*Must*	5
Acceso aplicación	*Must*	5

*Tabla 9: **Spring** Backlog 1*

ID	**HU-001**
OBJETIVO	Quiero registrarme para estar registrado en la aplicación
SECUENCIA	1. El usuario accede a la página principal 2. Rellena los campos nombre, email y contraseña 3. Presiona el botón registrarse
COMPLEJIDAD	5
PRIORIDAD	*Must*
PRUEBA DE ACEPTACIÓN	● El usuario ha de escribir su usuario, email y contraseña para registrarte ● Si no escribe todos los datos se indicara que campos son incorrectos ● Si escribe un email o un nombre que ya estuviera registrado se ha de indicar

Tabla 10: Historia De Usuario 001

ID	**HU-002**
OBJETIVO	Quiero acceder a la aplicación para entrar en mi zona personal
SECUENCIA	1. El usuario accede a la página principal 2. Rellena sus credenciales 3. Presiona el botón Login
COMPLEJIDAD	5
PRIORIDAD	*Must*
PRUEBA DE ACEPTACIÓN	● Una vez rellenados los credenciales y presionado el botón se accede a la zona personal de la aplicación ● Si se introduce una combinación incorrecta se ha de indicar. ● Si accedió con anterioridad se mantendrá la sesión.

Tabla 11: Historia De Usuario 002

En este caso la planificación es sencilla pues la historia de usuario HU-002 es dependiente de HU-001, pues un usuario no puede acceder a la aplicación sin antes haberse registrado.

Se procede entonces a detallar las historias de usuario. Cada historia se compone de varias tareas. Estas tareas tienen sentido para los desarrolladores, pues describen los pasos para la implementación de la tarea.

DISEÑO

HU-001

Se va a realizar un prototipo de la UI para clarificar como se verá la interfaz.

Prototipos

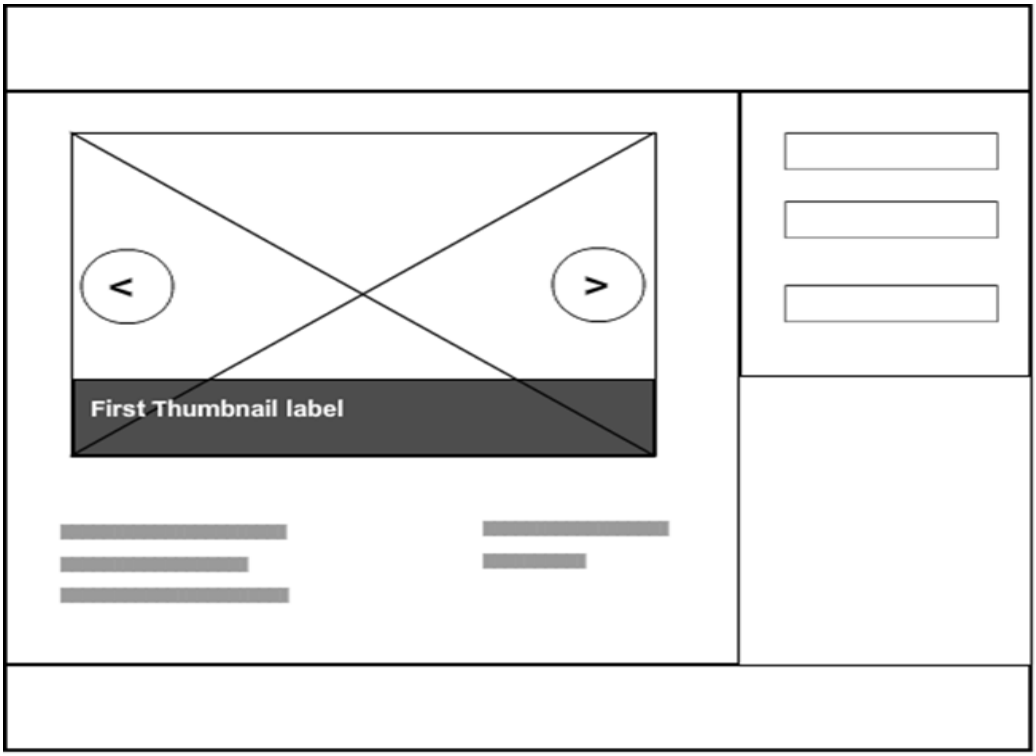


Ilustración 30: Prototipo Pagina Bienvenida

En este prototipo se puede ver en qué consistirá la página de bienvenida. El cuadro principal muestra algunas imágenes de la aplicación con algunas anotaciones explicando en que consiste esta. En la parte superior se muestra el logo y nombre de la aplicación, mientras que en la parte inferior se muestra el pie de página.

Por ultimo en la parte derecha se dispondrá del registro donde el usuario introduce sus datos, para poder registrarse.

CRC

A continuación se especifica las entidades involucradas.

Usuario	
Nombre Email Contraseña	

Tabla 12: CRC Usuario 1

Tareas

Por último se especifican las tareas a realizar:

HU-001	Tarea
HU-001-T1	Crear proyecto con código inicial
HU-001-T2	Crear controlador inicial y pagina bienvenida
HU-001-T3	Crear formulario registro y su controlador
HU-001-T4	Verificar campos son correctos

Tabla 13: Tareas HU-001

HU-002

Prototipos

El prototipo de la interfaz es el que se ofrece a continuación:

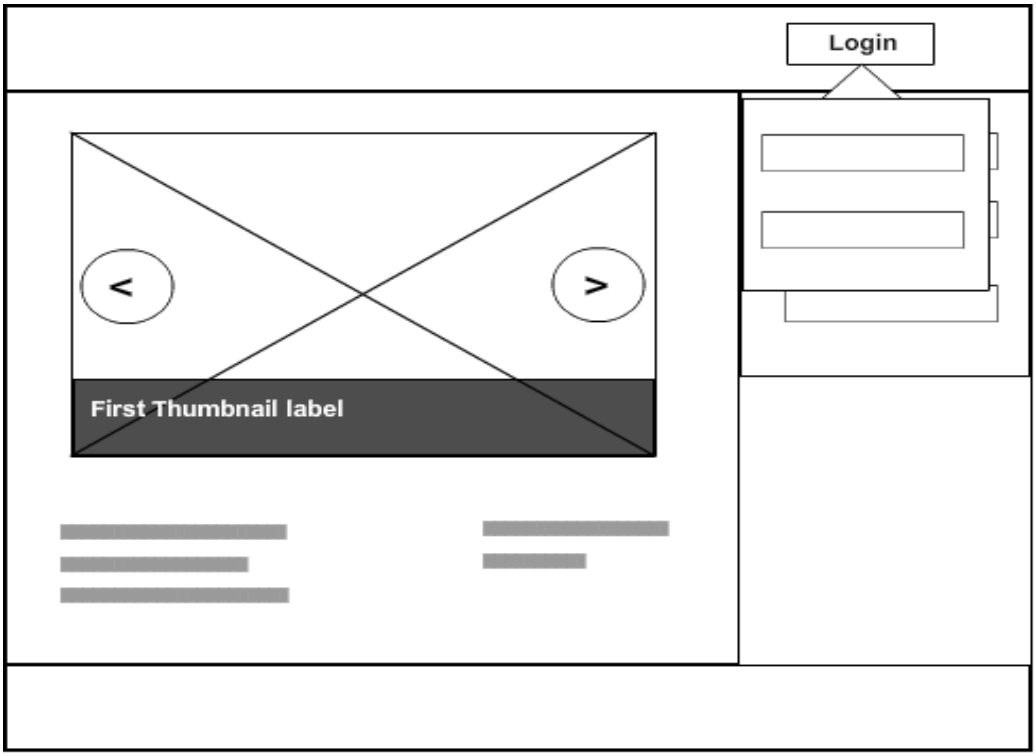


Ilustración 31: Prototipo Registro

CRC

En el caso de los CRC, no se añade ninguna nueva entidad, pero la entidad Usuario gana nuevas responsabilidades

Usuario	
Nombre Email Contraseña Acceder Salir	

Tabla 14: CRC Usuario 2

Tareas

Las tareas a realizar serán las siguientes:

HU-002	Tarea
HU-002-T1	Crear página de aplicación
HU-002-T2	Crear controlador
HU-002-T3	Crear formulario logging
HU-002-T4	Verificar campos son correctos y señalizar en caso de error

Tabla 15: Tareas HU-002

DESARROLLO

Código

A continuación se muestra una parte del código creado para la historia de usuario HU-001:

```
def signup(request):
    if request.method == 'POST':
        user_name = request.POST.get('username', None)
        user_password = request.POST.get('userpassword', None)
        user_mail = request.POST.get('usermail', None)
        if User.objects.filter(email=user_mail).exists():
            return redirect("/?error=email")
        if User.objects.filter(username=user_name).exists():
            return redirect("/?error=username")
        user = User.objects.create_user(user_name, user_mail, user_password)
        #Includes in group regular users
        group, created = Group.objects.get_or_create(name='Users')
        if created:
            utils.add_permission_to_group(Permission.objects.get(codename='add_project'),
group)
            utils.add_permission_to_group(Permission.objects.get(codename='change_project')
, group)
            utils.add_permission_to_group(Permission.objects.get(codename='delete_project')
, group)
        utils.add_user_to_group(user,group)
        user.save()
        user_profile = UserProfile()
        user_profile.user = user
        user_profile.save()
        user = authenticate(username = user_name, password= user_password)
        login(request, user)
        return redirect("/")
```

Lo que se puede ver es una función dentro del fichero views.py, lo que en Django se llama vista, y que corresponde al controlador en MVC. Como se puede ver el código captura los campos de la llamada POST y crea el grupo Users en caso de no existir. Después crea un objeto User y UserProfile asociados y los guarda en la base de datos. Si todo ha sido correcto redirige al usuario a la aplicación.

RETROINSPECCION

Como recomienda Scrum, al finalizar la iteración de debe realizar una retroinspección evaluando las estimaciones realizadas. En el caso de esta primera iteración, ha sido correcta la evaluación de un mes para las dos primeras tareas, pues la mayor parte del tiempo ha sido empleado en montar el sistema. Se ha tenido que realizar primero el desarrollo en una maquina local, y después en el sistema de Heroku.

Como la evaluación ha sido correcta en este caso, se va a dejar la velocidad en 14 puntos.

3.4.2 Iteración 2

ANÁLISIS

Para esta segunda iteración se va a realizar dos historias también. La razón para solo realizar solo dos historias de usuario, es debido a que se ha de realizar la creación inicial del sistema SPI (*single page application*), creando la funcionalidad básica para funcionar sin recargar por completo la aplicación.

Nombre	Prioridad	Complejidad
Añadir tarea	*Must*	5
Añadir estado	*Must*	5

*Tabla 16: **Spring** Backlog 2*

ID	HU-003
OBJETIVO	Quiero añadir un proyecto
SECUENCIA	1. El usuario accede a la aplicación 2. Usuario presiona en el símbolo nuevo proyecto 3. Rellena el nombre del proyecto
COMPLEJIDAD	5
PRIORIDAD	*Must*
PRUEBA DE ACEPTACIÓN	● El usuario rellena el campo nombre del proyecto y opcionalmente la descripción. ● Una vez introducido el proyecto aparece en la lista lateral. ● El cuadro para introducir el proyecto se cierra.

Tabla 17: Historia De Usuario 003

ID	HU-004
OBJETIVO	Quiero añadir una tarea al sistema
SECUENCIA	1. El usuario una vez en la aplicación accede a un proyecto. 2. El usuario introduce una tarea en el formulario.
COMPLEJIDAD	5
PRIORIDAD	*Must*
PRUEBA DE ACEPTACIÓN	● El usuario introduce un texto en el formulario. ● Una vez introducido la tarea aparece en una lista. ● El campo para introducir tareas vuelve a estar vacío.

Tabla 18: Historia De Usuario 004

Como en la primera iteración la primera historia se ha de realizar antes que la segunda, pues esta última depende de la primera.

DISEÑO

HU-003

Prototipos

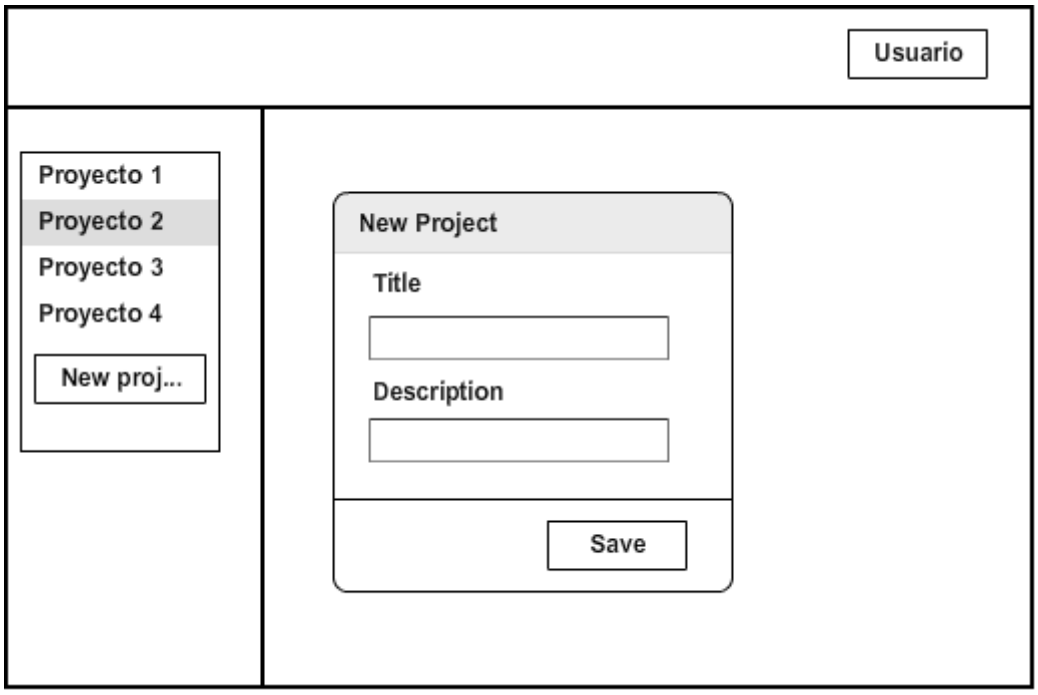


Ilustración 32: Prototipo Crear Proyecto

En el prototipo se puede ver que va a existir una barra lateral donde el usuario podrá añadir un nuevo proyecto. Una vez presionado el botón de nuevo proyecto, aparecerá un dialogo modal donde se podrá rellenar el título y la descripción del proyecto. Al crear satisfactoriamente un proyecto se añadirá un nuevo elemento a la barra lateral.

CRC

A continuación se especifica las entidades involucradas.

Proyecto	
Nombre Descripción Dueño	Usuario

Tabla 19: CRC Proyecto 1

Diagrama

Para esta funcionalidad se va a mostrar un pequeño diagrama del funcionamiento del *front-end*, puesto que es la primera funcionalidad que hace uso de **Backbone.js**.

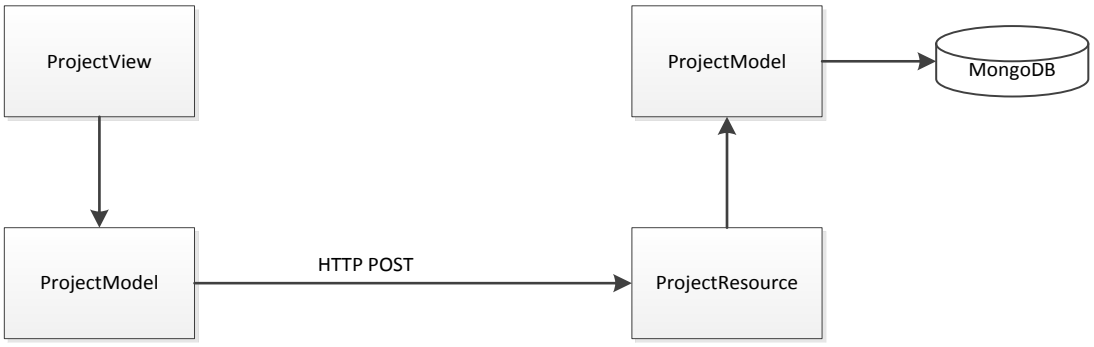


Ilustración 33: Creacion Proyecto

En este ejemplo podemos ver el funcionamiento simplificado de la creación de un proyecto. Una vez que el usuario ha rellenado el formulario de creación del proyecto la vista de **Backbone.js** ProjectView captura el evento y crea un modelo ProjectModel. Al guardarse se lanza una llamada **HTTP** POST con los datos necesarios para crear el modelo (Nombre, Descripcion, Usuario, etc) a la dirección del API correspondiente, que en este caso sería /api/v1/project. El servicio *back-end* captura la llamada y se crea un documento Project en la base de datos. En caso de que todo se realice correctamente se devolverá una llamada **HTTP** con código 201, indicando que se ha procesado correctamente.

Tareas

HU-001	Tarea
HU-003-T1	Crear backend **RESTful**
HU-003-T2	Crear página con **Backbone.js**
HU-003-T3	Crear formulario para crear proyecto
HU-003-T4	Crear lista de proyectos

Tabla 20: Tareas HU-003

HU-004

Prototipos

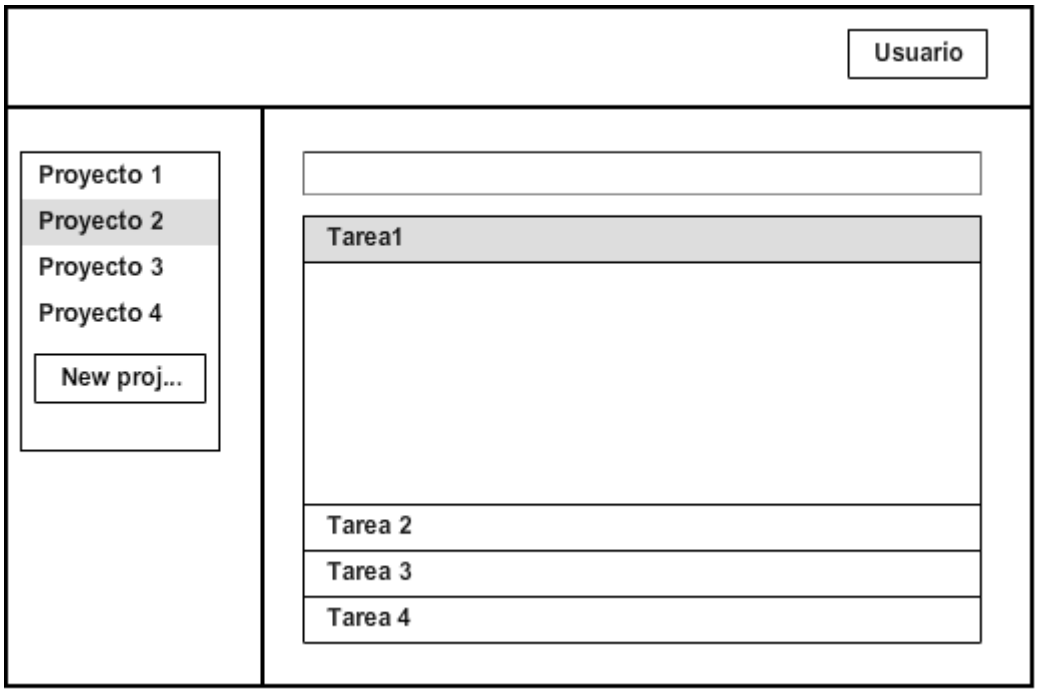


Ilustración 34: Prototipo Tareas

Como se puede ver existe un campo para introducir el título de la tarea. Una vez creado se añadirá a la lista inferior de tareas.

En este caso se va a añadir una nueva clase CRC para Tarea. Como en principio la tarea será solo un texto, los campos son limitados.

CRC

Tarea	
Nombre Proyecto Dueño	Usuario Proyecto

Tabla 21: CRC Tarea 1

Tareas

Las tareas a realizar serán las siguientes:

HU-004	Tarea

HU-004-T1	Crear recurso Tarea en servicio REST
HU-004-T2	Crear vista de tareas
HU-004-T3	Crear formulario para una tarea
HU-004-T4	Crear lista de tareas.

Tabla 22: Tareas HU-004

DESARROLLO

Código

A continuación se va a mostrar algunas clases creadas para manejar las tareas para la funcionalidad HU-004:

```
(function () {
    window.Task = Backbone.RelationalModel.extend({
    });
})();
(function() {
    window.TaskCollection = Backbone.Collection.extend({
        model: Task,
        urlRoot: APP_GLOBAL.PROJECT_API,
        URL: function(){
            return this.project.id + 'tasks/';
        }
    });
})();
```

El primer código es extremadamente simple, y representa a una clase modelo Task, con los campos vacíos. La segunda es una colección de Task, y como se puede observar se ha indicado el modelo que lo constituirá y a la URL a la que se ha de realizar las llamadas.

```
(function () {
    window.TaskListView = Backbone.View.extend({
        el: '#task-list',
        initialize: function(arguments){
            _.bindAll(this);
            this.collection.bind('reset', this.addAll, this);
            this.views = [];

            var self = this;
            this.collection.on('add', function(task){
                self.addOne(task);
            });

        },
        addAll: function(){
            this.views = [];
            this.collection.each(this.addOne);
        },
        addOne: function(task){
            var view = new TaskDetailView({
                parentView: this,
                model: task
            });
            if(task.get('status') == "DONE"){
                $(this.el).find("#task-list-recently-
one").hide().prepend(view.render().el).fadeIn('slow');
            }
            else{
                $(this.el).find("#task-to-
do").hide().prepend(view.render().el).fadeIn('slow');
            }
            this.views.push(view);
            view.bind('all', this.rethrow, this);
        },
        deleteOne: function(task){
            var v = this.views.filter(function(view) { return view.model == task })[0];
            var index = this.views.indexOf(v);
            this.views.splice(index, 1);
            v.remove();
        },
        render: function(){
            this.addAll();
        }
    });
})();
```

Como se puede ver esta clase es una vista de **Backbone.js**. Su cometido es controlar los eventos relacionados con las listas de tareas. Al inicializar la clase, se ligan los eventos “add” y “destroy” con funciones, de manera que cuando se añade un modelo a la colección de esta vista, reacciona

ejecutando la función addOne, que creara una nueva vista Detail (dejada fuera del ejemplo por su extensión) que manipulara el DOM añadiendo un nuevo elemento a partir de las plantillas HTML utilizando Mustache.js.

```
(function () {
    window.InputView = Backbone.View.extend({
        events: {
            'click .taskInput': 'createTask',
            'keypress #task-title': 'createOnEnter'
        },
        createOnEnter: function (e) {
            if ((e.keyCode || e.which) === 13) {
                this.createTask();
                e.preventDefault();
            }
        },
        createTask: function () {
            var title = this.$('#task-title').val();
            if (title) {
                var task = new Task({
                    title: title,
                    status : 'TODO',
                    creator: '/api/v1/user/' + APP_GLOBAL.USER + '/'
                });
                var that = this;
                var result = this.model.get('tasks').create(task,{
                    success : function(model) {
                        this.$('#task-title').val('');
                    },
                    error : function(model, response){
                        that.errorView = new Flash();
                        that.errorView.render("Sorry, there has been an error. :(", "error");
                    }
                });
            }
        },
        render: function () {
            $(this.el).HTML(ich.taskAddTemplate());
        }
    });
})();
```

En este caso se puede ver una plantilla para controlar la vista del formulario que añade un Proyecto. En la parte superior podemos ver una serie de eventos. Al lanzarse un evento DOM *click*, este fichero lanzara la función createTask, que llamara a la colección creando un nuevo elemento.

RETROINSPECCION

En esta iteración también se realizado en el tiempo especificado. Al igual que en la primera iteración se ha debido realizar tareas técnicas para iniciar el desarrollo sobre el que continuara en un futuro.

3.4.3 Iteración 3

ANÁLISIS

Después de evaluar el producto hasta el momento se van a añadir nuevas funcionalidades. Las tareas han de ser posible editarlas con más detalles, como la fecha de finalización, localización y descripción. Además ha de ser posible marcar como finalizada una tarea.

Estas serán las dos tareas nuevas:

Nombre	Prioridad	Complejidad
Detallar tarea	*Should*	3
Completar tarea	*Should*	2

Por tanto el *Backlog* en este momento, será este:

Nombre	Prioridad	Complejidad
Añadir estado	*Must*	5
Añadir nota	*Must*	5
Añadir fichero	*Must*	8
Borrar tarea	*Should*	1
Borrar nota	*Should*	1
Completar tarea	*Should*	2
Cambiar contraseña	*Should*	2
Editar tarea	*Should*	3
Detallar tarea	*Should*	3
Editar nota	*Should*	3
Refrescar lista de ficheros en la nube	*Should*	5
Compartir proyecto	*Should*	5
Refrescar proyecto	*Should*	5
Subir varios archivos a la vez	*Should*	8
Borrar estado	*Could*	1
Borrar fichero	*Could*	2
Sincronizar nota al editarse	*Could*	3
Añadir comentario	*Could*	5
Sincronizar nota al crearse	*Could*	8
Sincronizar nota al borrarse	*Won't*	2
Desincronizar servicios	*Won't*	3
Añadir avatar	*Won't*	5

Para esta iteración se va a revaluar la complejidad de historia "Añadir estado", en vez de 5 a 3, pues la estructura básica se realizó en la anterior iteración para la historia HU-004. Por tanto las hisotrias para esta iteración serán:

Nombre	Prioridad	Complejidad
Añadir estado	*Must*	3
Añadir nota	*Must*	5
Añadir fichero	*Must*	8

*Tabla 23: **Spring** Backlog 3*

En este caso la suma de puntos es 16. Al ser la suma de las dos primeras historias de usuario 8 y estar muy alejado de la velocidad actual se ha decidido incluir la siguiente historia aunque en principio no vaya a poder ser finalizada.

ID	HU-005
OBJETIVO	Quiero añadir un estado
SECUENCIA	1. El usuario accede a un proyecto 2. Usuario hace *click* en la vista Estado 3. Añade un texto al formulario y presiona enter
COMPLEJIDAD	5
PRIORIDAD	*Must*
PRUEBA DE ACEPTACIÓN	● El usuario rellena el formulario para estado. ● Se añade un nuevo estado, a la lista. ● El estado muestra el autor además del texto.

Tabla 24: Historia De Usuario 005

ID	HU-006
OBJETIVO	Quiero añadir una nota
SECUENCIA	1. El usuario accede a un proyecto 2. Usuario hace *click* en la vista Notas 3. Rellena el cuerpo de la nota 4. Presiona añadir nota
COMPLEJIDAD	5
PRIORIDAD	*Must*
PRUEBA DE ACEPTACIÓN	● El usuario rellena el cuerpo de la nota y el titulo ● Se añade una nueva nota a la izquierda

Tabla 25: Historia De Usuario 006

ID	HU-007
OBJETIVO	Quiero añadir un fichero
SECUENCIA	4. El usuario accede a un proyecto 5. Usuario hace *click* en la vista Fichero 6. Presiona botón subir fichero 7. Selecciona un fichero del sistema
COMPLEJIDAD	8
PRIORIDAD	*Must*
PRUEBA DE ACEPTACIÓN	● El usuario ha de ver un nuevo fichero al finalizar la carga. ● La lista se ha de actualizar ● Al hacer *click* en una fila se ha de descargar el mismo fichero que se subió.

Tabla 26: Historia de Usuario 007

DISEÑO

HU-005

Prototipos

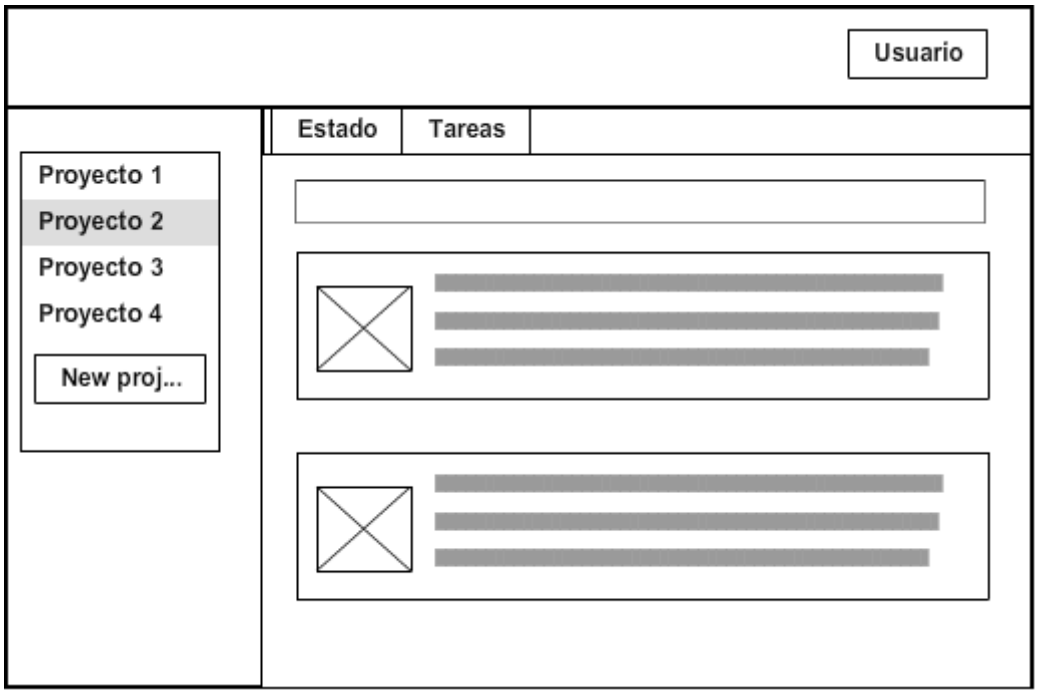


Ilustración 35: Prototipo Estados

Como se puede ver, el funcionamiento es similar al de las tareas; existe un formulario de entrada, donde se van a introducir los estados. Una vez introducidos, aparecerán en la lista de estados con la imagen del usuario y el texto introducido.

CRC

Estado	
Texto Proyecto Dueño	Usuario Proyecto

Tareas

HU-005	Tarea
HU-005-T1	Crear recurso Estado en servicio REST
HU-005-T2	Crear vista de estado
HU-005-T3	Crear formulario para un estado
HU-005-T4	Crear lista de estados.

Tabla 27: Tareas HU-005

HU-006

Prototipos

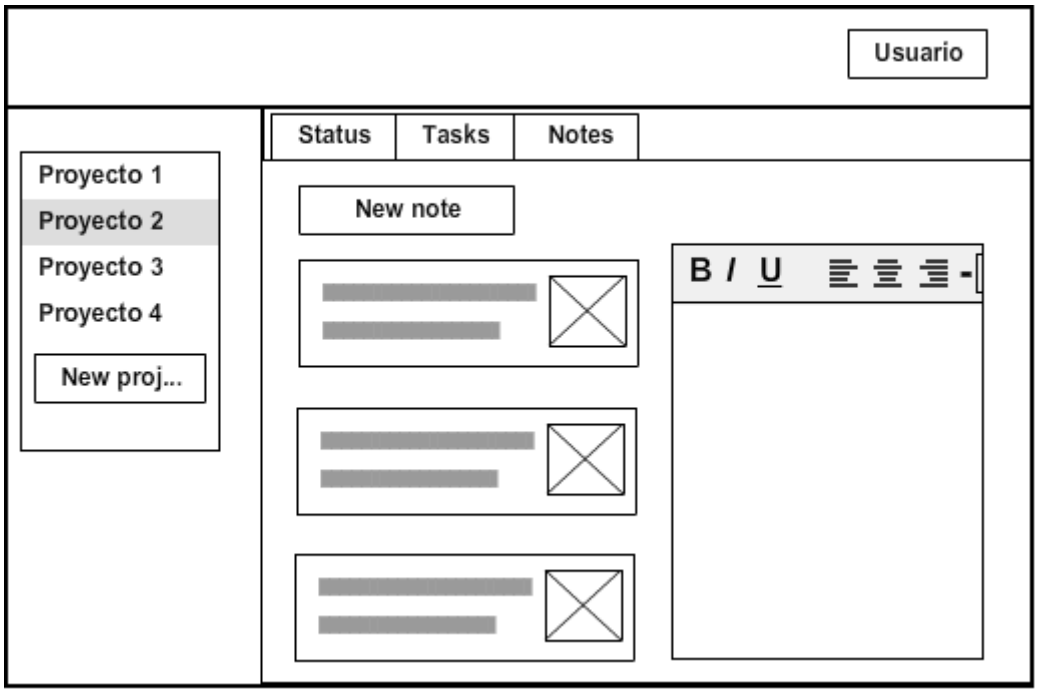


Ilustración 36: Prototipo Notas

En este caso, el usuario podrá añadir una nota escribiendo en el editor de texto incorporado, y a continuación pinchando en el botón crear nota. Una vez creada correctamente se añadirá a la lista lateral.

CRC

Nota	
Texto Titulo Proyecto Dueño	Usuario Proyecto

Tabla 28: CRC Nota 1

Tareas

HU-006	Tarea
HU-006-T1	Crear recurso Nota en servicio REST
HU-006-T2	Crear vista general.
HU-006-T3	Crear editor de texto para la nota.
HU-006-T4	Crear lista de estados.

Tabla 29: Tareas HU-006

HU-007

Prototipos

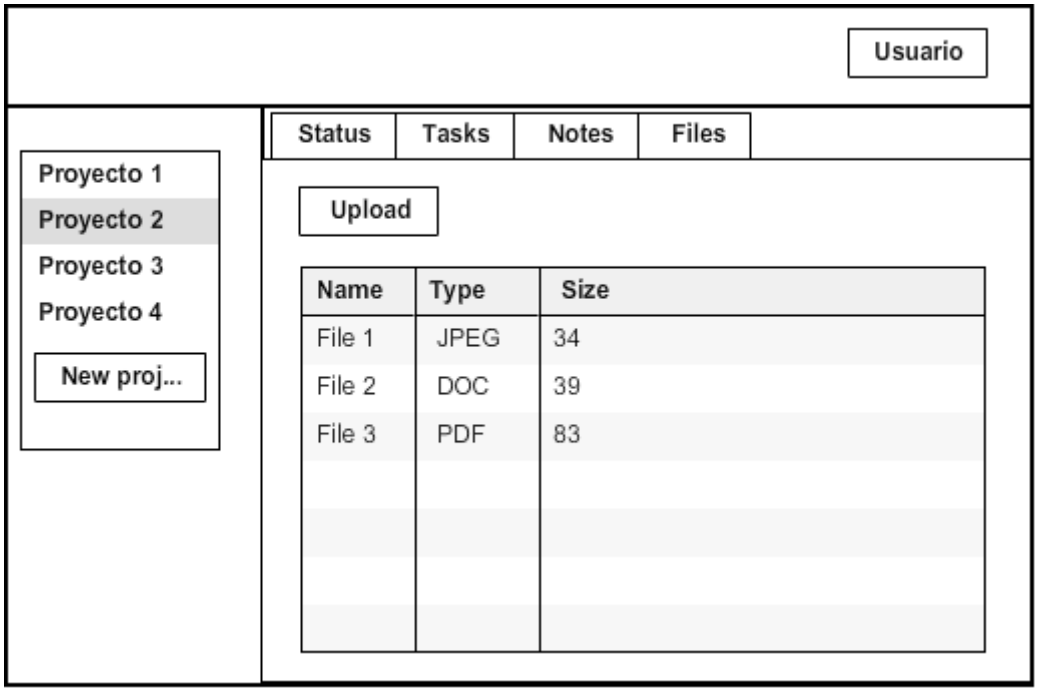


Ilustración 37: Prototipo Archivos

Para el caso de los ficheros en un principio se tendrá una vista similar al resto. Dentro de esta vista se tendrá un botón, el cual al ser pinchado abrirá un dialogo para subir un archivo. En un principio es una subida de un solo fichero. Una vez completada la subida del fichero, se actualizara la lista de ficheros. Además en el caso de no estar conectada con a la cuenta de Dropbox se ha de mostrar una vista vacía con la posibilidad de la conexión.

CRC

FicheroMetadatos	
Ruta Fecha modificación Proyecto Tipo	Usuario Proyecto

Revisión Dueño Sincronizar fichero	

Tabla 30: CRC FicheroMetadatos 1

PerfilDropbox	
Token Sincronizado	Usuario

Tabla 31: CRC Perfil Dropbox 1

En este caso se ha de añadir dos tarjetas de CRC. El primero va a ser el recurso FicheroMetadatos, el cual representa toda la información del fichero. Solo se almacenara los metadatos pues el fichero binario, se enviara a Dropbox donde se realiza el alojamiento. También se necesitara realizar una clase para manejar el acceso al API de Dropbox.

Diagramas

Para la realización de esta funcionalidad va a ser necesario implementar una autentificación basada en el protocolo OAuth. El protocolo OAuth permite la autorización segura de una API de una forma fácil y estándar para un cliente. En el siguiente diagrama se expone el funcionamiento que se va a implementar para autorizar la aplicación por parte del API de Dropbox.

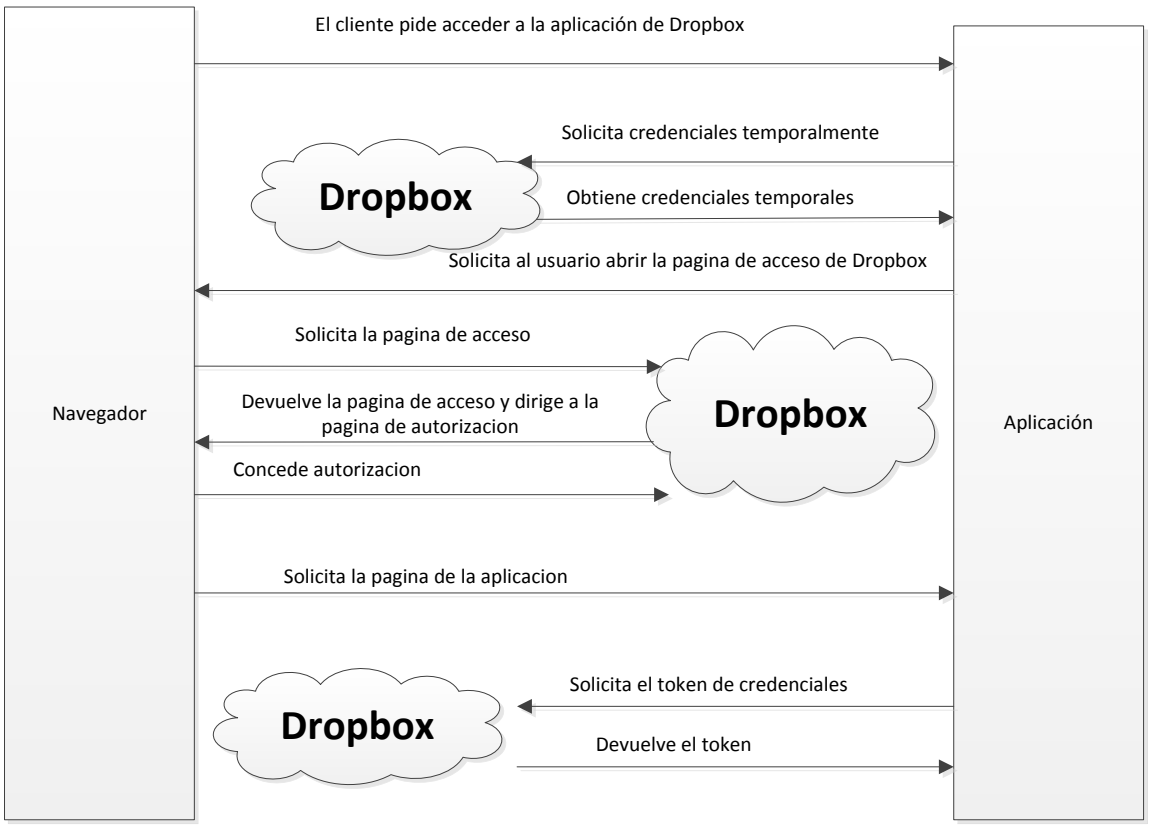


Ilustración 38: Diagrama OAuth

Tareas

HU-007	Tarea
HU-007-T1	Crear vista en caso de no estar autorizado a Dropbox.
HU-007-T2	Crear autorización OAuth
HU-007-T3	Crear recurso FicheroMetadatos
HU-007-T4	Subir archivo
HU-007-T5	Lista de archivos

RETROINSPECCION

A pesar de haber incluido más puntos de la velocidad indicada, se ha conseguido entregar las tres historias de usuario. Por tanto, se va a realizar un aumento de la velocidad hasta 15.

3.4.4 Iteración 4

ANÁLISIS

Una vez implementados los estados, se cree conveniente ser capaz de ofrecer una vista previa en caso de introducir un link. Esto aumentara la interactividad del sistema y mayor capacidad de compartir contenidos. Tendrá una complejidad media, pues se ha de conseguir un proveedor que enviando un link se consiga la pre visualización.

Nombre	Prioridad	Complejidad
Pre visualizar link	*Should*	5

El *Backlog* en estos momentos será:

Nombre	Prioridad	Complejidad
Borrar tarea	*Should*	1
Borrar nota	*Should*	1
Completar tarea	*Should*	2
Cambiar contraseña	*Should*	2
Editar tarea	*Should*	3
Detallar tarea	*Should*	3
Editar nota	*Should*	3
Pre visualizar link	*Should*	5
Refrescar lista de ficheros en la nube	*Should*	5
Compartir proyecto	*Should*	5
Refrescar proyecto	*Should*	5
Subir varios archivos a la vez	*Should*	8
Borrar estado	*Could*	1
Borrar fichero	*Could*	2
Sincronizar nota al editarse	*Could*	3
Añadir comentario	*Could*	5
Sincronizar nota al crearse	*Could*	8
Sincronizar nota al borrarse	*Won't*	2
Desincronizar servicios	*Won't*	3
Añadir avatar	*Won't*	5

Y las historias que se eligen para ser realizadas en la cuarta iteración serán:

Nombre	Prioridad	Complejidad
Cambiar contraseña	*Must*	2
Borrar tarea	*Should*	1
Borrar nota	*Should*	1
Completar tarea	*Should*	2
Editar tarea	*Should*	3
Detallar tarea	*Should*	3
Editar nota	*Should*	3

*Tabla 32: **Spring** Backlog 4*

Se ha revaluado la importancia de una historia de usuario "Cambio de contraseña", que va a pasar a tener un valor *Must*, por lo que será la primera tarea en ser realizada.

ID	HU-008
OBJETIVO	Quiero cambiar la contraseña
SECUENCIA	1. El usuario accede a las opciones de perfil 2. El usuario introduce su contraseña actual 3. El usuario introduce dos veces la nueva contraseña
COMPLEJIDAD	2
PRIORIDAD	*Must*
PRUEBA DE ACEPTACIÓN	● La contraseña ha de ser cambiada. ● Se ha de ofrecer un mensaje tanto si se ha cambiado correctamente como si no.

Tabla 33: Historia de Usuario 008

ID	HU-009
OBJETIVO	Quiero borrar una tarea
SECUENCIA	1. El usuario accede a la vista de tareas. 2. El usuario presiona el botón de la papelera.
COMPLEJIDAD	1
PRIORIDAD	*Should*
PRUEBA DE ACEPTACIÓN	● Al presionar el botón borrar sale una confirmación de borrado. ● Una vez borrado desaparece de la lista.

Tabla 34: Historia de Usuario 009

ID	HU-010
OBJETIVO	Quiero borrar una nota
SECUENCIA	1. El usuario accede a la vista de notas. 2. El usuario presiona el botón de la papelera.
COMPLEJIDAD	1
PRIORIDAD	*Should*
PRUEBA DE ACEPTACIÓN	● Al presionar el botón borrar sale una confirmación de borrado. ● Una vez borrado desaparece de la lista.

Tabla 35: Historia de Usuario 010

ID	HU-011
OBJETIVO	Quiero marcar como completada una tarea
SECUENCIA	1. El usuario accede a la vista de tareas 2. El usuario presiona el marcador de checkbox.
COMPLEJIDAD	2
PRIORIDAD	*Should*
PRUEBA DE ACEPTACIÓN	● Al presionar el botón de completado la tarea pasa a la lista de completados, marcándose con colores translucidos ● Al marcar una tarea completada vuelve a la lista de no completadas.

Tabla 36: Historia de Usuario 011

ID	HU-012
OBJETIVO	Quiero editar una tarea.
SECUENCIA	1. El usuario accede a la vista de tarea. 2. El usuario pincha sobre la tarea, cambia los campos

	necesarios. 3. Guarda los cambios
COMPLEJIDAD	3
PRIORIDAD	*Should*
PRUEBA DE ACEPTACIÓN	● Una vez guardado los campos son actualizados

Tabla 37: Historia de Usuario 012

ID	HU-013
OBJETIVO	Quiero detallar una tarea
SECUENCIA	1. El usuario accede a la vista de tarea. 2. El usuario pincha sobre la tarea, puede añadir la localización, la fecha de finalización y descripcion 3. Guarda los cambios
COMPLEJIDAD	3
PRIORIDAD	*Should*
PRUEBA DE ACEPTACIÓN	● Una vez guardado los campos son actualizados ● Al introducir una localización y pinchar sobre ella se abre un mapa de Google maps

Tabla 38: Historia de Usuario 013

ID	HU-014
OBJETIVO	Quiero editar una nota
SECUENCIA	1. El usuario accede a la vista de notas. 2. El usuario pincha sobre una nota 3. El usuario pincha sobre el botón editar 4. Realiza cambios y pincha sobre guardar.
COMPLEJIDAD	3
PRIORIDAD	*Should*
PRUEBA DE ACEPTACIÓN	● Una vez guardada la nota se ha de reflejar los cambios

Tabla 39: Historia de Usuario 014

DISEÑO

HU-008

El desarrollo de esta tarea es relativamente sencillo, por lo que se han omitido algunos apartados.

CRC

Usuario	
Nombre Email Contraseña Cambiar contraseña	

Tabla 40: CRC Usuario 3

Se va a ampliar la clase Usuario, posibilitando la opción de cambiar la contraseña

Tareas

HU-008	Tarea
HU-008-T1	Crear vista opciones de usuario, con formulario para introducir contraseña
HU-008-T2	Crear servicio backend para cambiar contraseña

Tabla 41: Tareas HU-008

HU-009

El desarrollo de esta tarea es relativamente sencillo, por lo que se han omitido algunos apartados.

CRC

Tarea	
Nombre Proyecto Dueño Borrar	Usuario Proyecto

Tabla 42: CRC Tarea 2

Se va a ampliar la clase Tarea, añadiendo la función borrar.

Tareas

HU-009	Tarea
HU-009-T1	Crear botón borrar tarea
HU-009-T2	Crear backend para borrar tarea.

Tabla 43: Tareas HU-009

HU-010

El desarrollo de esta tarea es relativamente sencillo, por lo que se han omitido algunos apartados.

CRC

Nota	
Texto Titulo Proyecto Dueño Borrar	Usuario Proyecto

Tabla 44: CRC Nota 2

Se va a ampliar la clase Nota, posibilitando la opción de cambiar la contraseña.

Tareas

HU-010	Tarea
HU-010-T1	Crear botón borrar nota
HU-010-T2	Crear backend para borrar nota.

Tabla 45: Tareas HU-010

HU-011

Prototipos

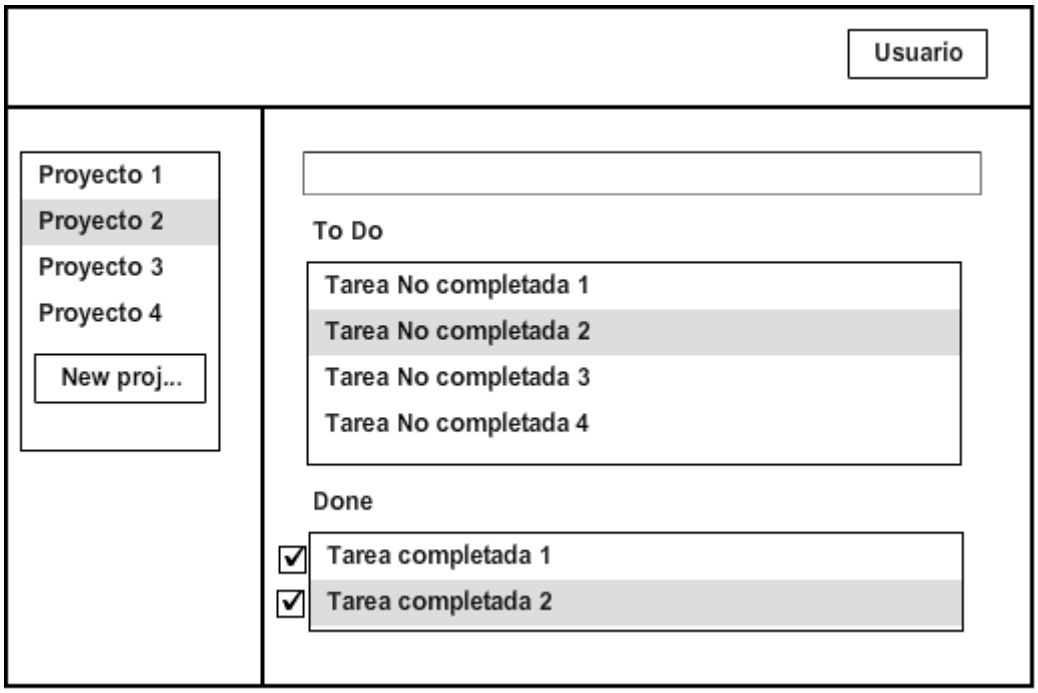


Ilustración 39: Prototipo Completar Tarea

Como se puede ver va a haber dos listas de tareas, las completadas y las no completadas. El funcionamiento consiste en al ser marcada una tarea como completada, la tarea ha de moverse a la lista de completadas y viceversa.

CRC

Tarea	
Nombre Proyecto Dueño Borrar Completada	Usuario Proyecto

Tabla 46: CRC Tarea 3

Se va a ampliar la clase Tarea con el atributo Completado

Tareas

HU-011	Tarea
HU-011-T1	Modificar recurso Tarea
HU-011-T2	Crear vista, con eventos al marcar una tarea.

Tabla 47: Tareas HU-011

HU-012

Prototipos

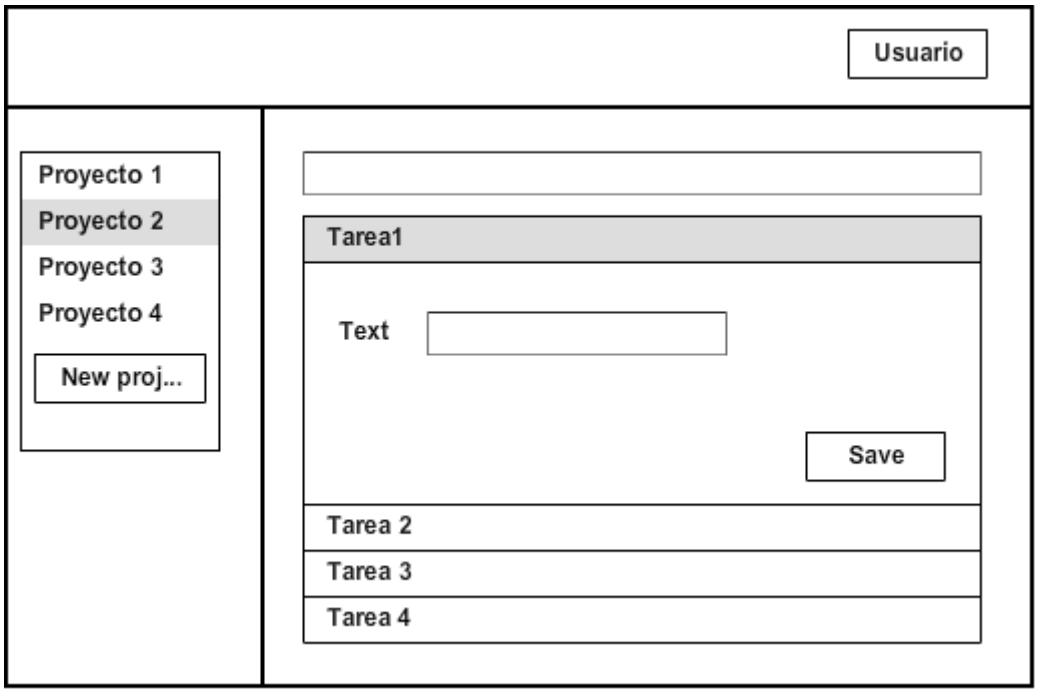


Ilustración 40: Prototipo Editar Nota

Se va a ampliar la interfaz de la tarea dando la posibilidad de cambiar el texto de la tarea.

CRC

Tarea	
Nombre Proyecto Dueño Borrar Completada Editar	Usuario Proyecto

Tabla 48: CRC Tarea 4

Se va a ampliar la clase Tarea con el atributo Completado

Tareas

HU-012	Tarea
HU-012-T1	Crear controlador para editar tarea
HU-012-T2	Modificar vista para poder guardar tarea modifciada

Tabla 49: Tareas HU-012

HU-013

Prototipos

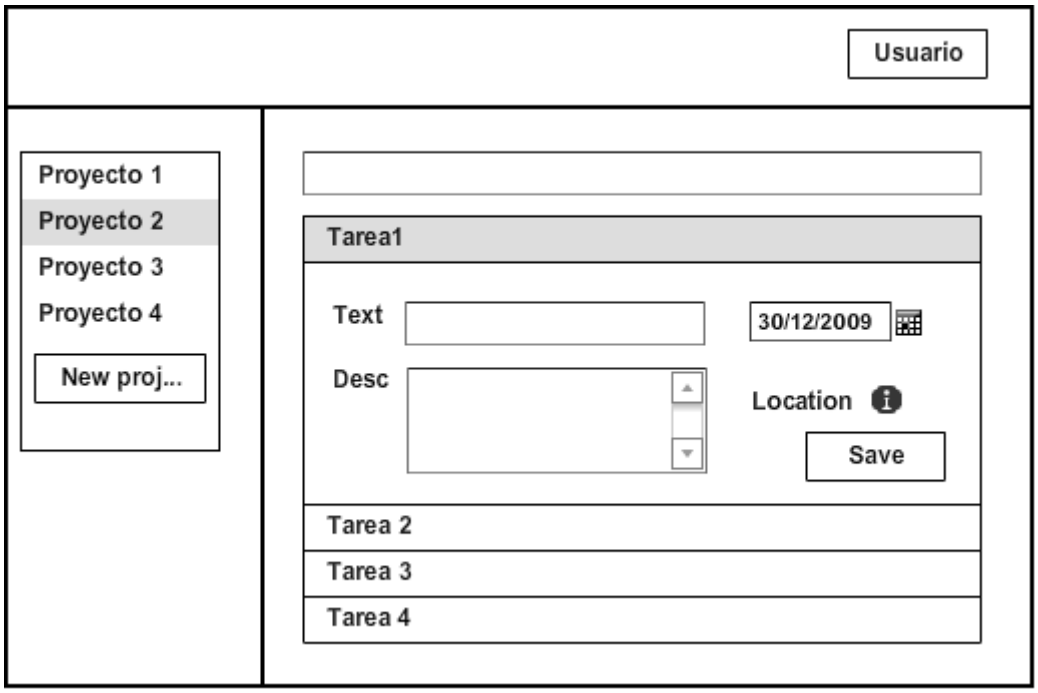


Ilustración 41: Prototipo Detalle Nota

Se va a ampliar la cantidad de opciones para rellenar una tarea, dando la posibilidad de especificar una descripción, una fecha límite y una localización. Al introducir una localización y acceder más tarde, se va a mostrar un mapa.

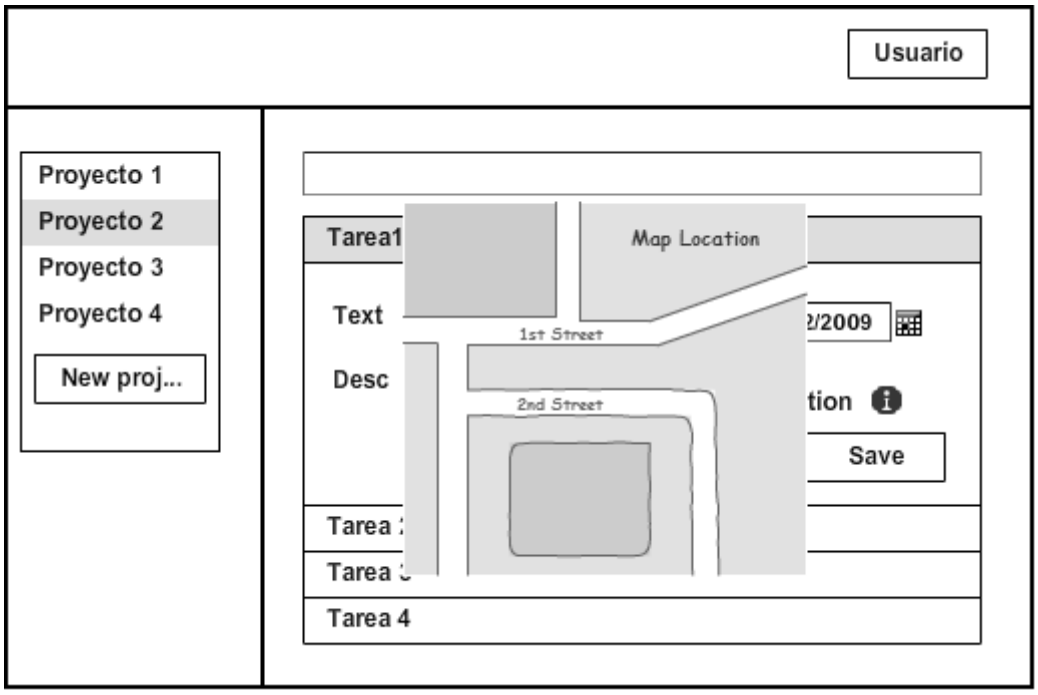


Ilustración 42: Prototipo Localización Tarea

CRC

Tarea	
Nombre Proyecto Dueño Borrar Completada Editar Localización Descripción Fecha limite	Usuario Proyecto

Tabla 50: CRC Tarea 5

Se van a incluir los nuevos atributos para Tarea.

Tareas

HU-013	Tarea
HU-013-T1	Modificar recurso Tarea
HU-013-T2	Modificar vista para añadir nuevos atributos
HU-013-T3	Utilizar *plugin* para mostrar localización.

Tabla 51: Tareas HU-013

HU-014

CRC

Nota	
Texto Titulo Proyecto Dueño Borrar Editar	Usuario Proyecto

Tabla 52: CRC Nota 3

Tareas

HU-014	Tarea
HU-014-T1	Crear controlador para editar nota
HU-014-T2	Modificar vista para poder guardar nota modificada

Tabla 53: Tareas HU-014

RETROINSPECCION

Se han podido realizar todas las tareas, ya que a pesar de ser muchas, la dificultad de ellas era baja.

3.4.5 Iteración 5

ANÁLISIS

Para la quinta iteración se pretende realizar las siguientes historias de usuario.

Nombre	Prioridad	Complejidad
Pre visualizar link	*Should*	5
Refrescar lista de ficheros en la nube	*Should*	5
Compartir proyecto	*Should*	5

*Tabla 54: **Spring** Backlog 5*

A continuación se detallan las historias de usuario

ID	HU-015
OBJETIVO	Quiero que se muestre una pre visualización de un link
SECUENCIA	1. El usuario se dirige a la vista de estados 2. El usuario introduce un link en el formulario de estado 3. El usuario presiona enter
COMPLEJIDAD	5
PRIORIDAD	*Should*
PRUEBA DE ACEPTACIÓN	● El link se ha de pre visualizar en caso de ser una página web ● Si es un link a un servicio de alojamiento de videos como YouTube, se ha de introducir el re*Product*or de video ● Si se introduce video mezclado con texto, se ha de mostrar el texto junto con la pre visualización.

Tabla 55: Historia de Usuario 015

ID	HU-016
OBJETIVO	Quiero que se refresquen los ficheros en la nube.
SECUENCIA	1. El usuario accede a la vista de ficheros 2. El usuario presiona el botón de refresco
COMPLEJIDAD	5
PRIORIDAD	*Should*
PRUEBA DE ACEPTACIÓN	● Los ficheros de la cuenta de **Dropbox** se han de mostrar en la lista ● Los ficheros que han sido modificados han de mostrar los nuevos cambios. ● Los ficheros borrados de la cuenta de **Dropbox** se han de eliminar.

Tabla 56: Historia de Usuario 016

ID	HU-017
OBJETIVO	Quiero compartir un proyecto
SECUENCIA	1. El usuario accede a la vista de compartir proyecto 2. El usuario introduce la dirección de email de su amigo 3. Manda la invitación
COMPLEJIDAD	5
PRIORIDAD	*Should*
PRUEBA DE ACEPTACIÓN	● El usuario invitado tendrá un nuevo proyecto en su lista. ● El proyecto tendrá todo el contenido que tenía en la cuenta original

Tabla 57: Historia de Usuario 017

DISEÑO

HU-015

Prototipos

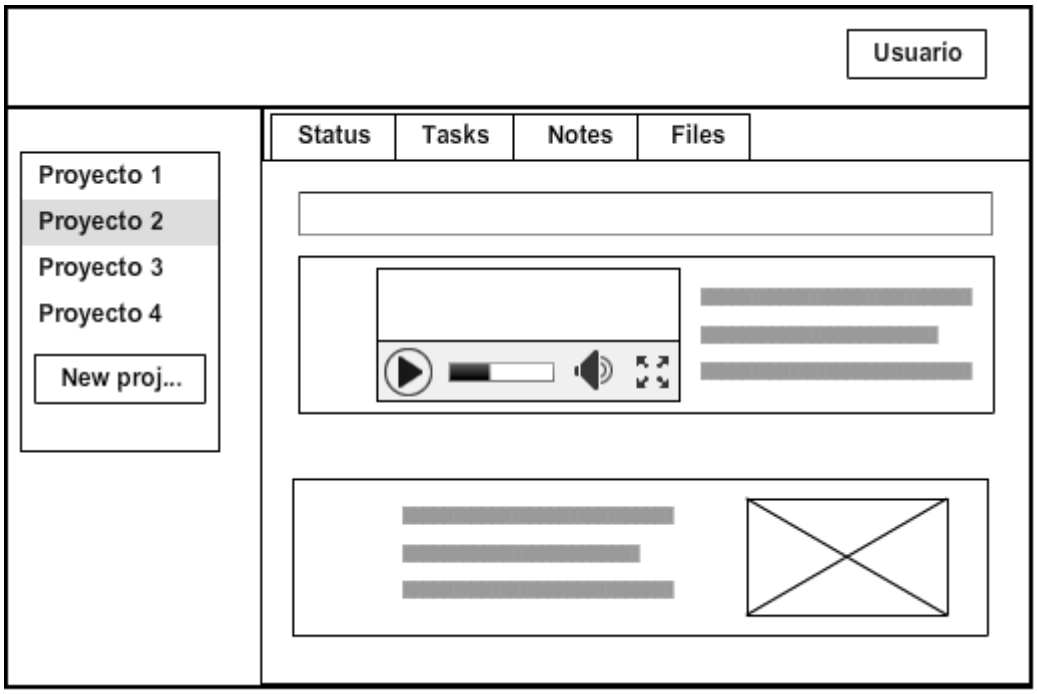


Ilustración 43: Prototipo Previsualizar Link

CRC

Estado	
Texto Proyecto Dueño Pre visualizar *Thumbnail* Código HTML	Usuario Proyecto

Tabla 58: CRC Estado 2

Se va a ampliar la clase Estado, añadiendo la función pre visualizar y dos atributos para mostrar una pre visualización.

Tareas

HU-015	Tarea
HU-015-T1	Conseguir *plugin* visualización
HU-015-T2	Procesar estado para detectar link

HU-015-T2	Crear llamada AJAX a servicio pre visualización

Tabla 59: Tareas HU-015

HU-016

CRC

FicheroMetadatos	
Ruta Fecha modificación Proyecto Tipo Revisión Dueño Sincronizar fichero Refrescar	Usuario Proyecto

Tabla 60: CRC Fichero Metadatos 2

La clase FicheroMetadatos tan solo es ampliada para incluir la función Refrescar

Tareas

HU-016	Tarea
HU-016-T1	Añadir icono refresco
HU-016-T2	Añadir nuevos ficheros desde Dropbox
HU-016-T3	Comprobar si han sido editado algún fichero

Tabla 61: Tareas HU-016

HU-017

CRC

Invitación	
Dirección receptor Dirección emisor	Usuario Proyecto

Tabla 62: CRC Invitación 1

Proyecto	
Nombre	Usuario

Descripción Dueño Lista de usuarios	

Tabla 63: CRC Proyecto 2

Para esta historia de usuario será necesario añadir una nueva clase, Invitación, para registrar quien ha enviado una invitación y a quien. Además también será necesario modificar la clase Proyecto, pues pasara a tener una lista de usuarios que pueden ver y modificar el proyecto.

Tareas

HU-017	Tarea
HU-017-T1	Crear recurso Invitación
HU-017-T2	Crear vista Invitación
HU-017-T3	Modificar backend y permisos para tener varios usuarios un proyecto

Tabla 64: Tareas HU-017

RETROINSPECCION

En este caso, no se ha podido completar la historia HU-017 en la iteración, principalmente por los problemas de las herramientas para Django-nonrel. Al no permitir las operaciones JOIN MongoDB, las relaciones Many-to-Many no están soportadas nativamente, y ha tenido que ser modificado manualmente el *framework*.

3.4.6 Iteración 6

ANÁLISIS

Para la sexta iteración se pretende realizar las siguientes historias de usuario.

Nombre	Prioridad	Complejidad
Refrescar proyecto	*Should*	5
Subir varios archivos a la vez	*Should*	8

Tabla 65: Iteración 6

Además es necesario finalizar la historia HU-017

A continuación se detallan las historias de usuario

ID	HU-018
OBJETIVO	Quiero que se refresque con nueva información los proyectos
SECUENCIA	1. El usuario accede a un proyecto compartido 2. El usuario añade un nuevo estado
COMPLEJIDAD	5
PRIORIDAD	*Should*
PRUEBA DE ACEPTACIÓN	● Si un usuario añade nuevos estados o tareas ha de añadirse a la interfaz de los usuarios que tengan acceso a ese proyecto y tengan abierta la aplicación ● Se ha de comprobar periódicamente si hay nueva información que mostrar.

Tabla 66: Historia de Usuario 018

ID	HU-019
OBJETIVO	Quiero que se puedan subir más de un fichero a la vez
SECUENCIA	1. El usuario va a la vista de ficheros 2. El selecciona varios ficheros 3. El usuario presiona a subir ficheros
COMPLEJIDAD	8
PRIORIDAD	*Should*
PRUEBA DE ACEPTACIÓN	● Los ficheros han de subirse a la vez ● Las cargas han de poder ser detenidas o canceladas ● Se ha de mostrar el progreso de las cargas.

Tabla 67: Historia de Usuario 019

DISEÑO

HU-018

CRC

Proyecto	
Nombre Descripción Dueño Lista de usuarios Refrescar	Usuario

Tabla 68: CRC Proyecto 3

Tan solo se va a añadir la posibilidad de refrescar el proyecto.

Tareas

HU-018	Tarea
HU-018-T1	Crear cliente Long Polling JavaScript
HU-018-T2	Adaptar interfaz

Tabla 69: Tareas HU-018

HU-019

Prototipos

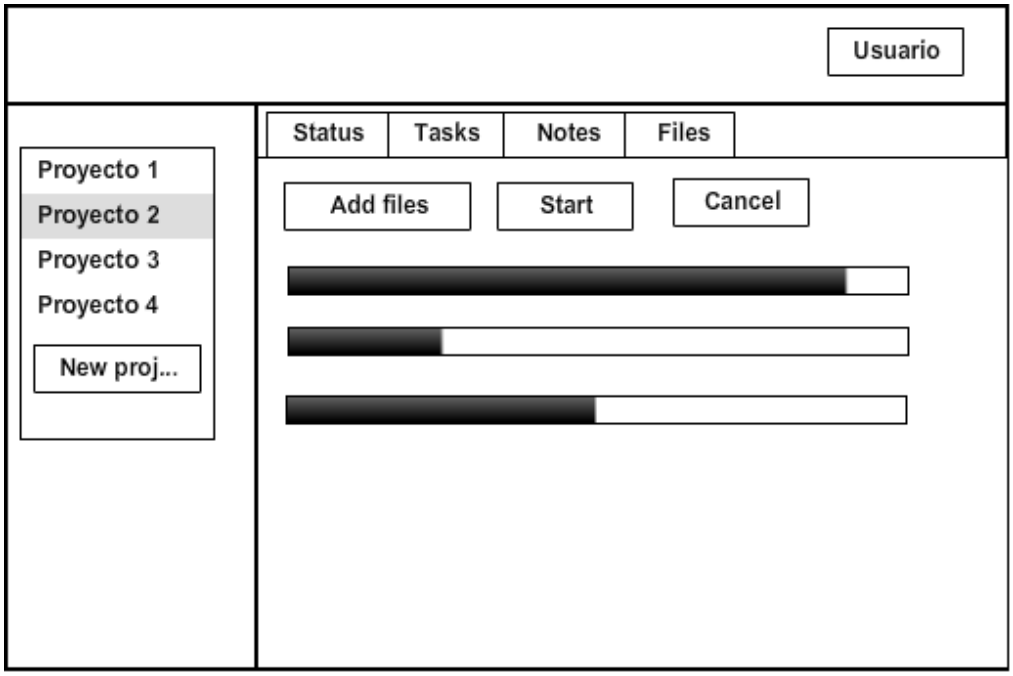


Ilustración 44: Prototipo Subir Archivos

Como se puede apreciar, se pueden seleccionar varios ficheros que cargaran simultáneamente.

CRC

MultiCarga	
Lista de ficheros Subir Pausar Cancelar	Proyecto

Tabla 70: CRC MultiCarga 1

Se añade una nueva clase, Multicarga.

Tareas

HU-019	Tarea
HU-019-T1	Buscar *plugin* multicarga HTML5
HU-019-T2	Crear servicio backend para tratar varias cargas
HU-019-T3	Crear vista y JavaScript necesario

Tabla 71: Tareas HU-019

RETROINSPECCION

En esta iteración no se ha conseguido acabar todas las historias de usuario. Además de existir una historia de usuario pendiente de la anterior iteración, la subida simultánea ha sido más costosa de lo previsto, creando grandes problemas en su implementación

3.4.7 Iteración 7

ANÁLISIS

Para la séptima iteración se pretende realizar las siguientes historias de usuario.

Nombre	Prioridad	Complejidad
Borrar estado	*Could*	1
Borrar fichero	*Could*	2
Añadir comentario	*Could*	5

Tabla 72: Sprint Backlog 7

Además es necesario finalizar la historia HU-019

A continuación se detallan las historias de usuario

ID	HU-020
OBJETIVO	Quiero que borrar un estado
SECUENCIA	1. El usuario accede a la vista de estados 2. El usuario presiona el botón borrar
COMPLEJIDAD	1
PRIORIDAD	*Could*
PRUEBA DE ACEPTACIÓN	● El estado desaparece de la lista de estados ● Se ha de comprobar periódicamente si hay nueva información que mostrar.

Tabla 73: Historia de Usuario 020

ID	HU-021
OBJETIVO	Quiero que borrar un fichero
SECUENCIA	1. El usuario va a la vista de ficheros 2. El usuario presiona el botón borrar
COMPLEJIDAD	2
PRIORIDAD	*Could*
PRUEBA DE ACEPTACIÓN	● El fichero desaparece de la lista de ficheros ● Se ha de comprobar periódicamente si hay nueva información que mostrar.

Tabla 74: Historia de Usuario 019

ID	HU-022
OBJETIVO	Quiero añadir un comentario a un estado
SECUENCIA	1. El usuario va a la vista de estados 2. El usuario añade un comentario en un estado
COMPLEJIDAD	5
PRIORIDAD	*Could*
PRUEBA DE ACEPTACIÓN	● Un comentario aparece en la parte inferior del comentario

DISEÑO

HU-018

CRC

Estado	
Texto Proyecto Dueño Previsualizar Thumbnail Código HTML Borrar	Usuario Proyecto

Tabla 75: CRC Estado 4

Tan solo se va a añadir la posibilidad de borrar el estado.

Tareas

HU-020	Tarea
HU-020-T1	Crear interfaz con boton para borrar
HU-020-T2	Crear función backend

Tabla 76: Tareas HU-020

HU-021

CRC

FicheroMetadatos	
Ruta Fecha modificación Proyecto Tipo Revisión Dueño Sincronizar fichero Refrescar Borrar	Usuario Proyecto

Tabla 77: CRC Fichero Metadatos 3

Se añade una nueva función, borrar.

Tareas

HU-021	Tarea
HU-021-T1	Añadir botón borrar en la interfaz
HU-021-T2	Crear servicio backend para borrar
HU-021-T3	Sincronizar borrado con Dropbox

Tabla 78: Tareas HU-021

HU-022

CRC

Comentario	
Texto Dueño Estado	Usuario Estado

Tabla 79: CRC Comentario 1

Se añade una nueva clase, comentario.

Tareas

HU-022	Tarea
HU-022-T1	Añadir botón añadir comentario en la interfaz
HU-022-T2	Crear servicio *back-end* para añadir una lista de comentarios
HU-022-T3	Actualizar interfaz con nuevos comentarios

Tabla 80: Tareas HU-022

RETROINSPECCION

En esta iteración se ha conseguido acabar todas las historias de usuario, pero con un gran esfuerzo dedicado a la creación de comentarios. La razón de esto es que el *framework* Tastypie no permite las relaciones doblemente anidadas de entidades y MongoDB no permite relaciones *Many-to-Many* lo que no permite tener una dirección inequívoca del comentario.

3.4.8 Iteración 8

ANÁLISIS

Para la octava iteración se pretende realizar las siguientes historias de usuario. Se ha re-priorizado algunas historias que quedaran fuera de esta iteración.

Nombre	Prioridad	Complejidad
Sincronizar nota al crearse	*Could*	8
Sincronizar nota al editarse	*Could*	3
Sincronizar nota al borrarse	*Could*	2

Tabla 81: Sprint Backlog 8

A continuación se detallan las historias de usuario

ID	HU-022
OBJETIVO	Quiero sincronizar una nota al crearse
SECUENCIA	1. El usuario accede a la vista de notas 2. El usuario crea una nota 3. El usuario guarda la nota
COMPLEJIDAD	8
PRIORIDAD	*Could*
PRUEBA DE ACEPTACIÓN	● Una vez creada la nota esta aparece en la cuenta de **Evernote**

Tabla 82: Historia de Usuario 021

ID	HU-023
OBJETIVO	Quiero sincronizar una nota al editarse
SECUENCIA	1. El usuario accede a la vista de notas 2. El usuario edita una nota 3. El usuario guarda la nota
COMPLEJIDAD	3
PRIORIDAD	*Could*
PRUEBA DE ACEPTACIÓN	● Una vez editada la nota los cambios se ven reflejados en la cuenta de **Evernote**

Tabla 83: Historia de Usuario 022

ID	HU-024
OBJETIVO	Quiero sincronizar una nota al crearse
SECUENCIA	1. El usuario accede a la vista de notas 2. El usuario borra una nota
COMPLEJIDAD	2
PRIORIDAD	*Could*
PRUEBA DE ACEPTACIÓN	● Una vez borrada la nota esta desaparece de la cuenta de **Evernote**

Tabla 84: Historia de Usuario 023

DISEÑO

HU-022

CRC

Nota	

Texto Titulo Proyecto Dueño Borrar Evernote Id Evernote USN Sincronizar	Usuario Proyecto

Tabla 85: CRC Nota 4

PerfilEvernote	
Token Sincronizado	Usuario

Tabla 86: CRC Perfil Evernote 1

Se van añadir los campos necesarios para poder sincronizar las notas con Evernote. Esto será un perfil de Dropbox y también diversos atributos para el recurso Nota.

Tareas

HU-022	Tarea
HU-022-T1	Adaptar SDK Evernote
HU-022-T2	Crear *back-end* autorización OAuth Evernote
HU-022-T3	Crear vista autorización
HU-022-T4	Parseador XML para nota
HU-022-T5	Sincronizar nota

Tabla 87: Tareas HU-022

HU-023

CRC

Nota	
Texto Titulo Proyecto Dueño	Usuario Proyecto

Borrar **Evernote** Id **Evernote** USN Sincronizar Comprobar cambios	

Tabla 88: CRC Nota 5

Tan solo se añadirá una nueva funcionalidad, consistente en comprobar los cambios de la nota.

Tareas

HU-023	Tarea
HU-023-T1	Comprobar cambios en backend
HU-023-T2	Sincronizar nota

Tabla 89: Tareas HU-023

DESARROLLO

HU-022

Código

A continuación se expone un fragmento de código utilizado para la realización de la historia de usuario HU-022

En primer lugar se muestra el recurso Nota. Esta representa lo que se expone a través del API. Además de esto existe una función obj_crete que hereda de la clase superior. Al recibirse una llamada POST para crear un nuevo recurso, además de las operaciones que realiza la función base, comprueba si el usuario esta sincronizado con **Evernote**. En caso afirmativo llama a la función del modelo.

```
class NoteCollectionResource(MongoListResource):
    title = fields.CharField(attribute= 'title', default='', blank = True)
    Evernote_usn = fields.IntegerField(default=0, blank=True, null=True)
    Evernote_guid = fields.CharField(null=True, blank=True)
    modified = fields.DateTimeField(attribute='modified', null=True, blank=True)
    created = fields.DateTimeField(attribute='created', null=True, blank=True)
    […]
    def obj_create(self, bundle, request=None, **kwargs):
        bundle.data['created'] = datetime.now().isoformat()
        bundle = super(NoteCollectionResource, self).obj_create(bundle,request, **kwargs)
        #Check if Evernote accout is set up
        user_profile = UserProfile.objects.get(user = request.user)
        if user_profile.is_Evernote_synced:
            project = Project.objects.get(pk = self.instance.pk)
            self.instance.notes[int(bundle.obj.id)].create_note_Evernote(user_profile, project)
        return bundle
    […]
```

El modelo Nota contiene la función create_note_Evernote que simplemente crea una clase ayuda llamada EvernoteHelper con los credenciales del usuario. Una vez que tiene la nota selecciona la libreta donde se va a mandar y se envía con los credenciales y la nota.

```
class Note(models.Model):
    title = models.CharField(max_length=200, default='', blank= True)
    content = models.TextField(max_length=5000)
    created = models.DateTimeField()
    modified = models.DateTimeField()
    Evernote_usn = models.IntegerField(default=0, blank=True)
    Evernote_guid = models.CharField(max_length=200, null=True, blank=True)
    comments = EmbeddedModelListField(EmbeddedModelField('Comment'), null=True, blank=True)
    resources = DictField(blank=True, null=True)
    […]
    def create_note_Evernote(self, user_profile, project):
        auth_token = user_profile.Evernote_profile.auth_token
        Evernote_helper = EvernoteHelper(user_profile.Evernote_profile)
        noteStore = Evernote_helper.note_store
        note = Evernote_helper.create_note(self.title, self.content, project)
        try:
            created_note = noteStore.createNote(auth_token, note)
        except Errors.EDAMUserException as e:
            return None
        except Errors.EDAMNotFoundException as e:
            return None
        except Exception as e:
            raise e
        #If note is synced fine the usn is updated
        user_profile.Evernote_profile.latest_update_count = created_note.updateSequenceNum
        user_profile.save()
        self.Evernote_usn = created_note.updateSequenceNum
        self.Evernote_guid = created_note.guid
        project.notes[int(self.id)] = self
        project.save()
        return created_note
    […]
```

Como se puede comprobar la function create_note crea un document XML, añade el contenido en formato UTF-8 y devuelve la nota.

```
def create_note(self, title, content, project):
    # To create a new note, simply create a new Note object and fill in
        # attributes such as the note's title.
        note = Types.Note()
        note.title = title if (title is not None and len(title)  > 0) else 'No title'
        note.title = note.title.encode('utf-8')
        # The content of an Evernote note is represented using Evernote Markup Language
        # (ENML). The full ENML specification can be found in the Evernote API Overview
        # at http://dev.Evernote.com/documentation/cloud/chapters/ENML.PHP
        note.content = '<?XML version="1.0" encoding="UTF-8"?>'
        note.content += '<!DOCTYPE en-note SYSTEM "http://XML.Evernote.com/pub/enml2.dtd">'
        note.content += content.encode('utf-8')
        #note.content += '<br/>'

        note.notebookGuid = self.get_notebook(self.Evernote_profile, project).guid
        return note
```

HU-023

Código

A continuación se muestra el código necesario para sincronizar las notas.

```
@classmethod
    def get_synced_notes(cls, user_profile, parent_project):
        Evernote_profile = user_profile.Evernote_profile
        auth_token = Evernote_profile.auth_token
        Evernote_helper = EvernoteHelper(Evernote_profile)
        note_store = Evernote_helper.note_store
        current_state = Evernote_helper.current_state_count
        if Evernote_profile.latest_update_count < current_state:
            new_notes = Evernote_helper.get_metadata_notes(Evernote_profile, parent_project)
            for metadata_note in new_notes.notes:
                #look for note in db
                notes = parent_project.notes
                local_note = filter(lambda n : n.Evernote_guid == metadata_note.guid, notes)
                #If found the note we update the note
                if len(local_note) is 1 and local_note[0].Evernote_usn <
metadata_note.updateSequenceNum:
                    full_note = note_store.getNote(auth_token, metadata_note.guid, True, True, False,
False)
                    local_note = local_note[0]
                    local_note = Evernote_helper.copy_Evernote_note(full_note, local_note)
                #If not found, a new note is created
                elif len(local_note) is not 1:
                    full_note = note_store.getNote(auth_token, metadata_note.guid, True, True, False,
False)
                    local_note = Note()
                    local_note = Evernote_helper.copy_Evernote_note(full_note, local_note)
                    parent_project.notes.append(local_note)
            Evernote_profile.latest_update_count = current_state
            user_profile.save()
            parent_project.save()
```

Esta función pertenece al modelo Note, mostrado en la funcionalidad HU-22. El funcionamiento consiste primero en comprobar si el número de secuencia en el servidor de Evernote es mayor que el almacenado en la aplicación. En caso negativo, se ignora pues la aplicación está actualizada. En caso positivo, se obtiene tan solo los metadatos de las notas, para reducir el tráfico, y se comprueba una por una si esa nota esta creada o actualizada, realizando lo apropiado en cualquiera de los dos casos.

RETROINSPECCION

En esta iteración se ha conseguido acabar todas las historias de usuario.

3.4.9 Iteración 9

ANÁLISIS

Esta será la última iteración y se añadirán tan solo dos historias de usuario. Estas historias son nuevas por otra parte, resultado de una cambio de visión y nuevas ideas surgidas durante el desarrollo.

Nombre	Prioridad	Complejidad
Menú contextual	*Could*	8
Visor general	*Could*	3

Tabla 90: Sprint Backlog 9

A continuación se detallan las historias de usuario

ID	**HU-024**
OBJETIVO	Quiero un menú contextual al lado de los estados para añadir tareas y notas.
SECUENCIA	1. El usuario accede a la vista de estados 2. El usuario selecciona un estado 3. El usuario añade una tarea
COMPLEJIDAD	8
PRIORIDAD	*Could*
PRUEBA DE ACEPTACIÓN	● Al añadir una tarea en el menú aparece una tarea en el menú lateral y en la lista de tareas ● Al añadir una nota en el menú aparece una nota en el menú lateral y en la lista de notas

Tabla 91: Historia de Usuario 024

ID	**HU-025**
OBJETIVO	Quiero una vista general de todos los elementos
SECUENCIA	1. El usuario accede a la aplicación 2. El usuario accede a un proyecto previamente creado o compartido 3. El usuario pincha en la venta de vista general
COMPLEJIDAD	3
PRIORIDAD	*Could*
PRUEBA DE ACEPTACIÓN	● El visor mostrara todos los elementos del proyecto

Tabla 92: Historia de Usuario 025

DISEÑO

HU-024

Prototipos

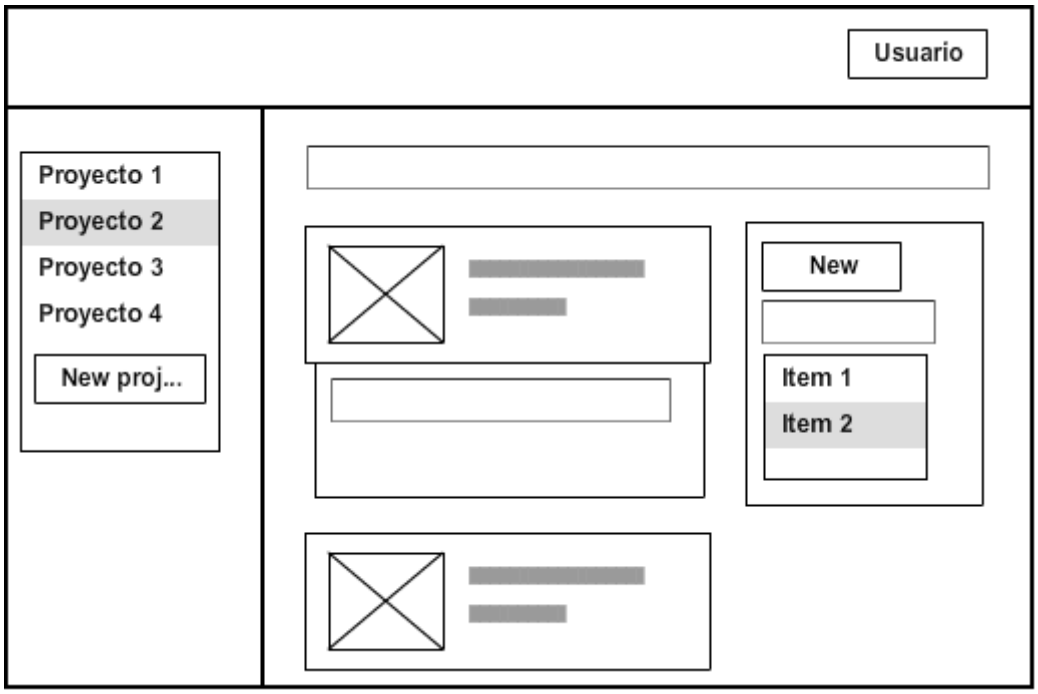


Ilustración 45: Prototipo Menu Contextual

Como se puede ver aparece un menú en la parte lateral. Este menú servirá para añadir tareas y notas asociadas a un estado en concreto.

CRC

MenuContextual	
Lista de tareas Lista de notas Añadir nota Añadir tarea	Usuario Proyecto

Tabla 93: CRC Menu Contextual 1

Se va a añadir una clase

Tareas

HU-024	Tarea
HU-024-T1	Crear vista menu contextual
HU-024-T2	Crear añadir nota
HU-024-T3	Crear añadir tarea

Tabla 94: Tareas HU-024

HU-025

Prototipo

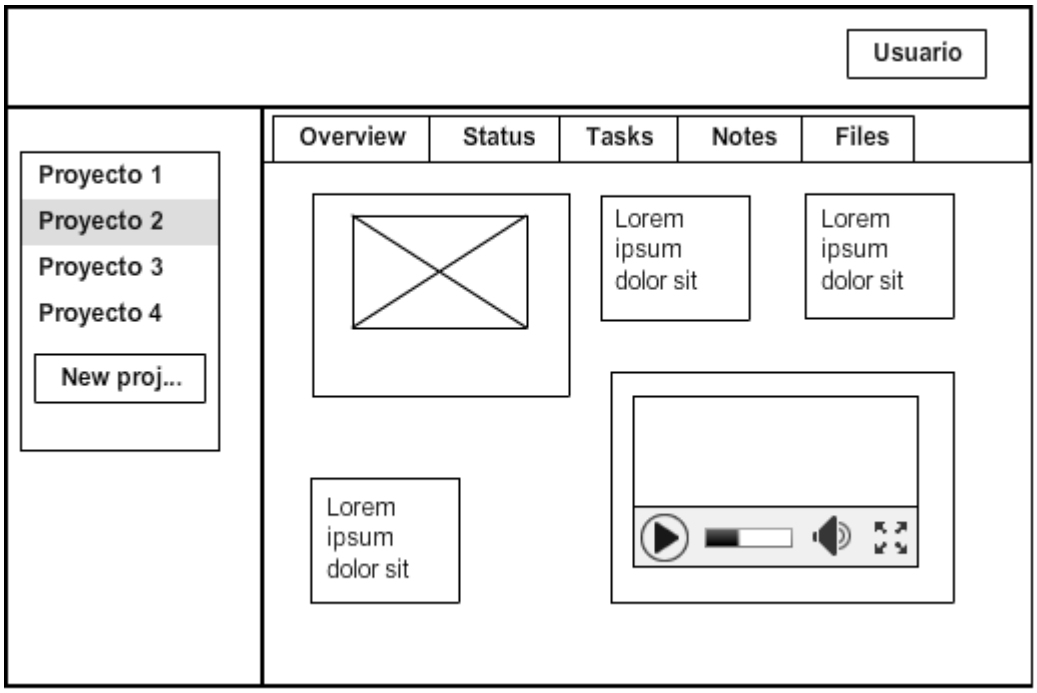


Ilustración 46: Prototipo Visor General

CRC

En este caso no se va a utilizar ninguna nueva clase, por lo que no se añade ningún CRC.

Tareas

HU-025	Tarea
HU-025-T1	Incluir *plugin* Masonry
HU-025-T2	Crear vista overview
HU-025-T3	Incluir tarea, nota, estado y fichero

Tabla 95: Tareas HU-025

RETROINSPECCION

En esta iteración se ha conseguido acabar todas las historias de usuario. Al ser la última iteración se puede hacer una valoración final. A pesar de que el objetivo de las metodologías agiles no es hacer una planificación inicial del tiempo que se va a dedicar, la estimación dada al principio del proyecto ha sido bastante acertada. Además, a medida que se desarrollaba el producto se tenía una certeza mayor de la velocidad de desarrollo del equipo y se podía hacer una mejor estimación.

4 CONCLUSIONES

A continuación se expondrá las principales conclusiones que se pueden extraer de la realización de este proyecto. Estas conclusiones se pueden dividir en la consecución de los objetivos marcados al inicio del proyecto, las conclusiones técnicas en el uso de las herramientas utilizadas y en las conclusiones personales. Además se añadirán una serie de líneas futuras en las que se podría trabajar sobre el proyecto.

4.1 CONCLUSIONES SOBRE LOS OBJETIVOS

Como se indicó en la introducción el objetivo principal del proyecto era realizar una aplicación web que permitiese compartir información y organizar pequeños proyectos. Efectivamente se ha construido una aplicación web que permite crear proyectos y compartirlos entre diferentes usuarios. Estos proyectos pueden tener una serie de tareas, notas, conversaciones y/o ficheros alojados en la nube. Por tanto se considera cumplido íntegramente el objetivo principal.

Además se marcaron una serie de sub objetivos que debería cumplir el proyecto. Estos sub-objetivos eran la sencillez de uso, la fluidez de la aplicación y la integración con servicios de terceros. El primer de estos sub-objetivos se ha conseguido gracias al diseño de una interfaz sencilla y con diversas ayudas contextuales. Para la creación de estas interfaces han sido de gran utilidad la creación de prototipos rápidos, primero como simples bocetos a bolígrafo y a continuación mediante el uso de herramientas de prototipado.

El segundo de los sub-objetivos, la fluidez, se ha conseguido de una forma satisfactoria. Para ello se ha hecho uso de técnicas novedosas, como la aplicación del patrón *Single Page Application*. De esta forma en el primer acceso de a la aplicación se descarga todo el contenido estático, como hojas de estilos, plantillas y código script. A partir de este momento la navegación será casi instantánea, tan solo teniendo que esperar a comunicación de datos con el servidor. Así mismo, para evitar una carga inicial muy prolongada se ha minimizado y comprimido los datos estáticos. De esta forma se consigue una carga inicial de entorno 1-2 segundos y una navegación fluida.

El último sub-objetivo implicaba la interacción con servicios de terceros. La principal razón de marcarlo como un objetivo es la reutilización de soluciones ya probadas. En un mundo tecnológico donde cada vez hay más interacción es preferible hacer uso de los servicios especializados, gracias a APIs públicas. Este objetivo se ha cumplido satisfactoriamente haciendo uso de los servicios Dropbox y Evernote.

4.2 CONCLUSIONES TÉCNICAS

Han sido numerosas las nuevas tecnologías y herramientas utilizadas para la creación y desarrollo de la aplicación web. En general la impresión ha sido satisfactoria, pero han existido problemas relacionados con el uso de tecnologías tan novedosas y por tanto no lo suficiente maduras. Estos puntos se describen a continuación empezando desde el *back-end* al *front-end*.

El uso de Heroku se puede considerar un acierto, debido a que ha permitido ahorrar tiempo y dinero en manejar y configurar un servidor de aplicaciones. El uso es sencillo aunque han existido ciertos problemas, no graves, a la hora de configurar la aplicación de modo que se ejecute de forma satisfactoria. A la hora de añadir nuevos paquetes de software de los que depende la aplicación no ha habido ningún problema, pues pueden especificarse en un fichero de texto y estos serán buscados en

el directorio de Python mediante la herramienta Pip, o incluso especificar un repositorio Git o una dirección HTTP como ha sido el caso del SDK de Dropbox. Por otra parte los *plugin*s ofrecidos por Heroku han funcionado correctamente. Por ultimo se ha de indicar que Heroku obliga a seguir unas especificaciones a la hora de desarrollar, pero que se considera buenas prácticas pues se obtiene un código más modular.

Sin duda el punto más conflictivo ha sido la elección de MongoDB como sistema de persistencia de los datos. MongoDB en general ha funcionado correctamente, pero un cambio de paradigma de una base de datos relacional a una orientada a documentos influye en todo el diseño posterior y como se explicará más adelante, no todos las herramientas se adaptan perfectamente. En cuanto al uso exclusivamente de MongoDB, ha sido positivo como balance general. El diseño en forma de documentos tiene mucho sentido en el modelo de dominio de esta aplicación, al recuperarse un documento entero el cual será el proyecto con el que trabajar. Por otra parte, acelera el desarrollo al no ser necesario scripts para modificar las tablas; los documentos pueden tener una estructura irregular y añadir nuevos campos en caliente. En cuanto al rendimiento, al ser una aplicación sin mucha carga de momento, no se puede apreciar un incremento de velocidad apreciable pero según las pruebas de carga todo indica que supone una velocidad de aproximadamente el doble a la hora de recuperar datos y escribirlos.

El uso del *framework* Django ha sido útil limitadamente. Al utilizarse Backbone.js en la parte del cliente ha sido innecesario el uso del sistema de plantillas que ofrece Django. Uno de los puntos fuertes de Django es su sistema ORM, el cual ha sido reemplazado por la versión de Django adaptada para MongoDB. Este sin duda es el sistema más conflictivo, pues Django está preparado para un sistema relacional y relaciones Many-to-Many no son posibles debido a la falta de JOINs por parte de MongoDB. Este ha provocado la creación de código para controlar estas relaciones. Para exponer los modelos al API REST, se hizo uso del *framework* Tastypie, que al igual que Django esta preparado para el uso en bases de datos relacionales. Para adaptarlo a MongoDB se ha usado Tastypie-nonrel. Este *plugin* a pesar de ser útil, no cubre todas las necesidades e incluso se han detectado varios bugs que han sido necesario arreglar.

En cuanto al uso de los SDK de Dropbox y Evernote se ha de decir que han sido muy utilies para comunicar la aplicación con sus respectivas APIs. Sin embargo, mientras que el uso del SDK de Dropbox ha sido muy sencillo y bien diseñado, el SDK de Evernote ha sido problemático debido a un diseño defectuoso y poco intuitivo. Por otra parte Evernote obliga a mandar los documentos a crear en su propio sistema derivado de XML, el cual ha sido muy problemático, pues el editor introduce los documentos en HTML, los cuales no siempre cumplen el estándar XHTML, además de tener que introducir las imágenes mediante conversiones a binario y el uso de tablas hash para referirse a ellas.

En la parte de *front-end* como se explicó se ha utilizado Backbone.js. La experiencia general ha sido muy positiva pues permite una navegación muy fluida, transmitiéndose tan solo los datos necesarios, sin necesidad de enviar HTML constantemente. Sin embargo Backbone.js no está exento de problemas, pues Backbone.js es un *framework* muy ligero lo que implica que toda la organización del código ha de ser realizada por el desarrollador y tiene a una duplicidad de código. Además al basarse los modelos en un ID proporcionado por el servidor existieron ciertos problemas relacionados con modelos sin ID.

En cuanto a la metodología de gestión, Scrum en este caso, se puede considerar un éxito pues ha permitió modificaciones sobre la marcha de una forma poco costosa. Además esta metodología ha permitido reducir la documentación excesivamente detallada concentrándose en su lugar en el desarrollo.

Como resumen, la elección ha sido satisfactoria pero se debería haber confiado ciertas partes en tecnologías más maduras.

4.3 CONCLUSIONES PERSONALES

El objetivo marcado a la hora de desarrollar esta aplicación era poner en práctica el conocimiento adquirido durante la carrera. Por otra parte, otro de los objetivos personales era crear una aplicación real de principio a fin utilizando tecnologías de vanguardia.

La razón de crear un proyecto de principio a fin es tener una visión general de todo el ciclo de desarrollo de un producto, pues durante la realización de trabajos durante la carrera, en la realización de prácticas profesionales o durante el desempeño de un trabajo difícilmente se controla todo el proceso y se concentra a tan solo una pequeña parte de lo que es el producto.

El uso de tecnologías de vanguardia se debe a diversas razones, pues a pesar de haber sido más fácil crear una aplicación tradicional con un lenguaje ya aprendido durante la carrera, como Java, la carrera ha de dar las herramientas necesarias para aprender nuevas herramientas y lenguajes. Sin duda ha sido un desafío aprender todas las tecnologías y se han cometido multitud de errores tanto de organización y de desarrollo, pero a cambio se dispone de una experiencia muy valiosa y una capacidad de autoaprendizaje mejorada.

4.4 LÍNEAS FUTURAS

Como cualquier desarrollo este no está exento de mejoras. Algunas de ellas son las siguientes:

4.4.1 Integración con redes sociales

La integración con redes sociales comprende dos mejoras. La primera de ellas corresponde a habilitar un registro mediante Google, Facebook o Twitter. El registro en aplicaciones a pesar de ser algo sencillo y que no consume mucho tiempo, aleja a muchos usuarios. Con un registro integrado resulta mucho más directo y por tanto una cantidad mayor de usuarios utilizarán la aplicación.

Por otra parte, una integración con una red social permitiría encontrar a otros usuarios amigos de una forma más fácil y vistosa comparada con la actual. El usuario dispondría de una lista de los contactos existentes, dependiendo de la red social que ha conectado, y podría seleccionar a que usuarios manda una invitación para compartir el proyecto.

4.4.2 Sistema de votaciones:

Un sistema de votaciones para cada estado puede ser muy útil para proponer nuevas acciones o eventos. Una vez propuesto los usuarios pueden votar sobre dicha pregunta a favor o en contra. A pesar de la aparente simplicidad de la funcionalidad, es necesario asegurar que cada usuario puede votar un y solo una vez. Además es importante asegurar que se incremente de forma atómica los votos.

4.4.3 Mejorar la sincronización

A pesar de que la sincronización funciona de un modo bastante aceptable, se ha de considerar que actualmente solo existe un proceso por usuario. Esto supone un problema para la sincronización pues esta se realiza mientras que el usuario está navegando y supone hacer esperar al usuario bloqueando las acciones. La solución sería crear una serie de procesos que trabajen de forma paralela y actualicen cada cierto tiempo los proyectos del usuario.

4.4.4 Añadir elementos arrastrables

Sería conveniente crear elementos que se puedan arrastrar mediante el uso del ratón. Un ejemplo sería la lista de tareas. Un usuario podría arrastrar un elemento a la primera posición de forma que esta tarea pasaría a ser la primera de la lista.

4.4.5 Adaptar a dispositivos móviles

Mediante el uso de un diseño *responsive* es posible adaptar la disposición de los elementos HTML al tipo de pantalla. De esta forma un usuario móvil vería la aplicación con unos elementos acordes, mientras que un usuario de escritorio mantendría la interfaz actual, mientras que el funcionamiento permanecería inalterado

4.4.6 Incluir un chat

Para mejorar la comunicación entre los miembros de un proyecto sería posible añadir un chat interactivo. De esta forma se evitaría conversaciones intrascendentes en el muro de actividades mientras que se acelera la comunicación.

5 REFERENCIAS

[1] National Institute of Standards and Technology, «The NIST Definition of Cloud Computing,» 2011. [En línea]. Available: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.

[2] Amazon Inc., «¿Qué es la informática en la nube?,» [En línea]. Available: http://aws.amazon.com/es/what-is-cloud-computing/.

[3] Amazon Inc., «Amazon Elastic Compute Cloud (Amazon EC2),» [En línea]. Available: http://aws.amazon.com/es/ec2/.

[4] Amazon Inc., «Amazon Web Services Blog,» [En línea]. Available: http://aws.typepad.com/aws/2008/10/big-day-for-ec2.html.

[5] Amazon Inc., «Amazon Web Services Blog,» [En línea]. Available: http://aws.typepad.com/aws/2012/06/amazon-s3-the-first-trillion-objects.html.

[6] Dropbox Inc., «Where does Dropbox store everyone's data?,» [En línea]. Available: https://www.dropbox.com/help/7/en.

[7] Canonical, «Technical Details,» [En línea]. Available: https://wiki.ubuntu.com/UbuntuOne/TechnicalDetails.

[8] Heroku, Inc., «Heroku,» [En línea]. Available: https://www.heroku.com.

[9] W3C, «Web Services Glossary,» [En línea]. Available: http://www.w3.org/TR/ws-arch/#whatis.

[10] W3C, «Relationship to the World Wide Web and REST Architectures,» [En línea]. Available: http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest.

[11] R. T. Fielding, Architectural Styles and the Design of Network-based Software Architectures, 2000.

[12] Facebook Inc., «Facebook Reports First Quarter 2013 Results,» [En línea]. Available: http://investor.fb.com/releasedetail.cfm?ReleaseID=761090.

[13] Techcrunch, «Analyst: Twitter Passed 500M Users,» [En línea]. Available: http://techcrunch.com/2012/07/30/analyst-twitter-passed-500m-users-in-june-2012-140m-of-them-in-us-jakarta-biggest-tweeting-city/.

[14] The Next Web, «Code-sharing site Github turns five and hits 3.5 million users, 6 million repositories,» [En línea]. Available: http://thenextweb.com/insider/2013/04/11/code-sharing-site-github-turns-five-and-hits-3-5-million-users-6-million-repositories/.

[15] Agile Alliance, «Manifiesto for Agile Software Development,» [En línea]. Available: http://agilemanifesto.org/.

[16] International Conference on Information and Network Technology, «Software Development Life Cycle AGILE vs Traditional Approaches,» 2012. [En línea]. Available: http://www.ipcsit.com/vol37/030-ICINT2012-I2069.pdf.

[17] D. Wells, «Extreme Programming,» [En línea]. Available: http://www.extremeprogramming.org/.

[18] M. F. Kent Beck, Planning Extreme Programming, 2000.

[19] VersionOne, «Annual State of Agile Development Survey Results,» [En línea]. Available: http://www.versionone.com/state-of-agile-survey-results/.

[20] Dispatch, [En línea]. Available: https://dispatch.cc/.

6 GLOSARIO

A continuación se exponen algunas de las definiciones y acrónimos usados en este documento.

6.1 ACRÓNIMOS

- HTML : HyperText Markup Language
- CSS: Cascading Style Sheets
- XML: eXtensible Markup Language
- JSON: JavaScript Object Notation
- DOM: Document Object Model
- URL: Uniform resource locator
- URI: Uniform Resource Identifier
- SOAP: Simple Object Access Protocol
- HTTP: Hypertext Transfer Protocol
- REST: Representational State Transfer
- AJAX: Asynchronous JavaScript And XML
- WSDL: Web Services Description Language
- RSS: Really Simple Syndication
- SDK: Software development kit
- TDD: Test Driven Development
- BDD: Behaviour Driven Development
- SPI: Single Page Application

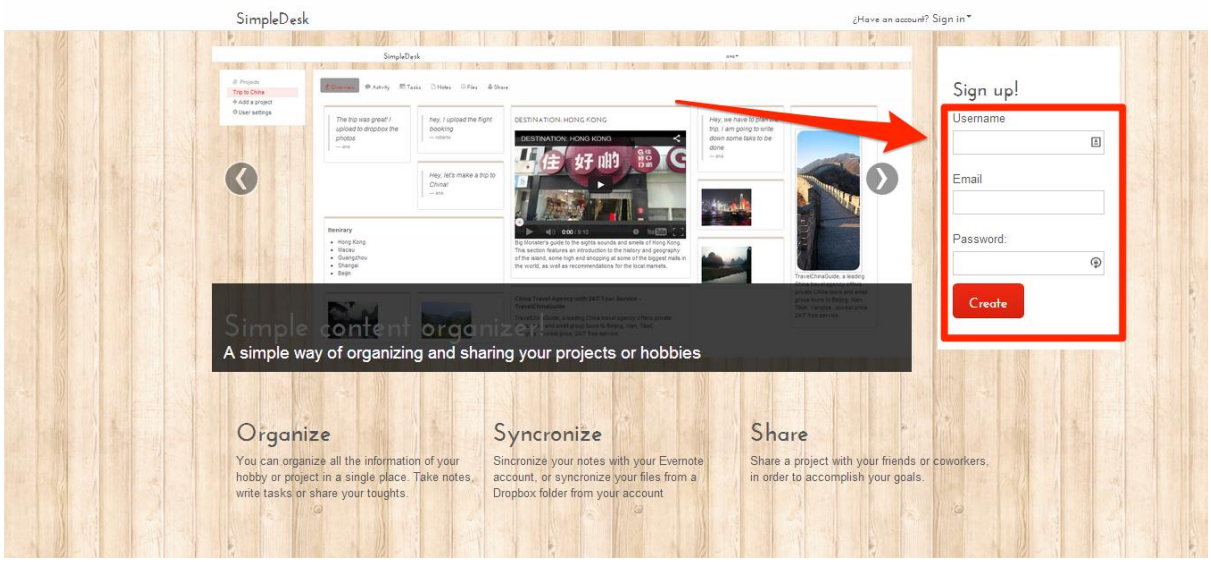
6.2 DEFINICIONES

- *Framework*: Conjunto de librerías y funciones estructuradas para ayudar en el desarrollo software.
- *Front-end*: La parte del software con la que interactúa el usuario.
- *Back-end*: La parte del software que procesa las entradas del *front-end*
- *Script*: Conjunto de instrucciones que permiten realizar una tarea de forma automática.
- *JavaScript*: Lenguaje script interpretado por los navegadores.
- *Plugin*: Software que se añade a otro y le complementa aportando nuevas funcionalidades.

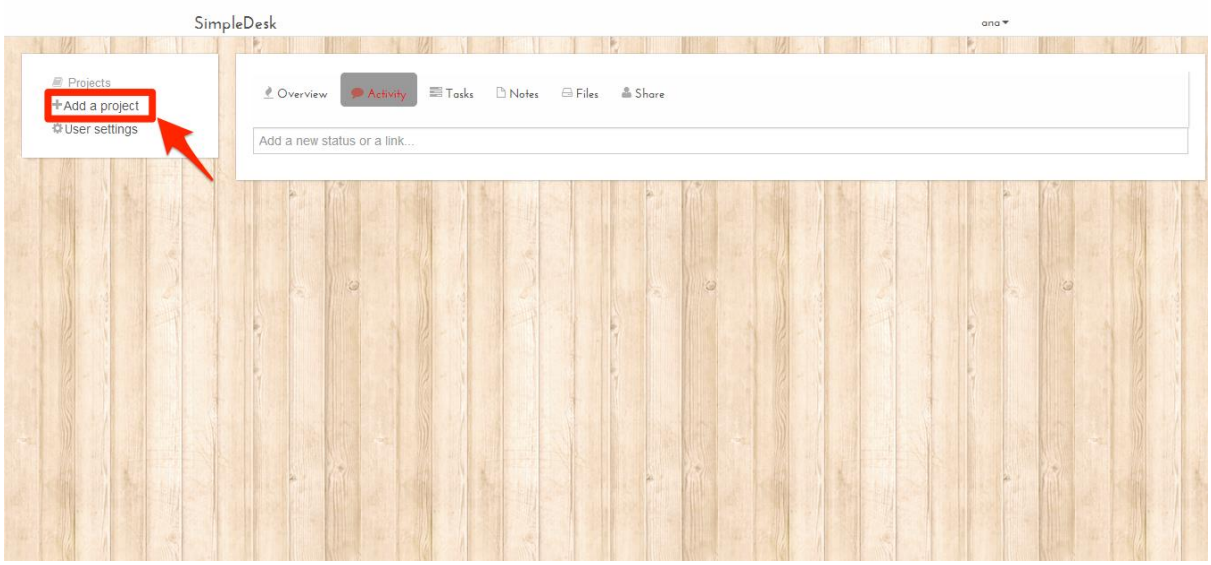
7 ANEXO I – MANUAL DE USUARIO

En esta sección se recoge una guía del uso de la aplicación web desarrollada.

Como registrarse

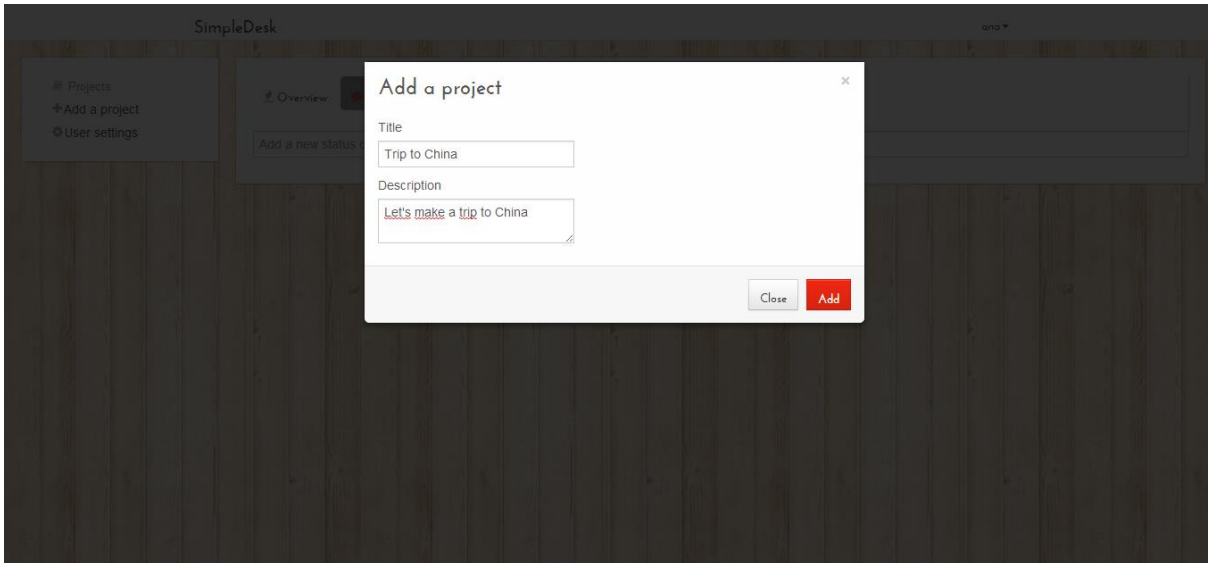


Para hacer uso de la aplicación el usuario se ha de registrar. Para ello tan solo se ha de rellenar el nombre de usuario, el email que va a usar y la contraseña. Una vez rellenados y si el nombre y el email están disponibles se creara la cuenta y se redirigirá a la aplicación.



Como crear nuevos proyectos

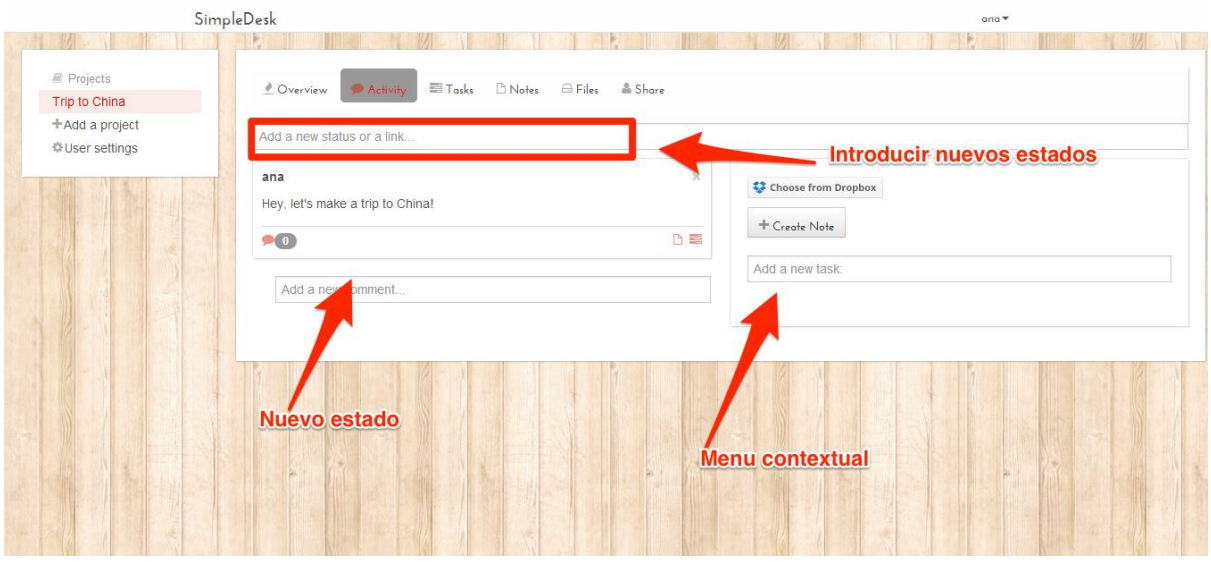
Una vez que se acceda a la página de inicio se ha de crear un proyecto. Para ello se debe presionar el link “Add a Project”.



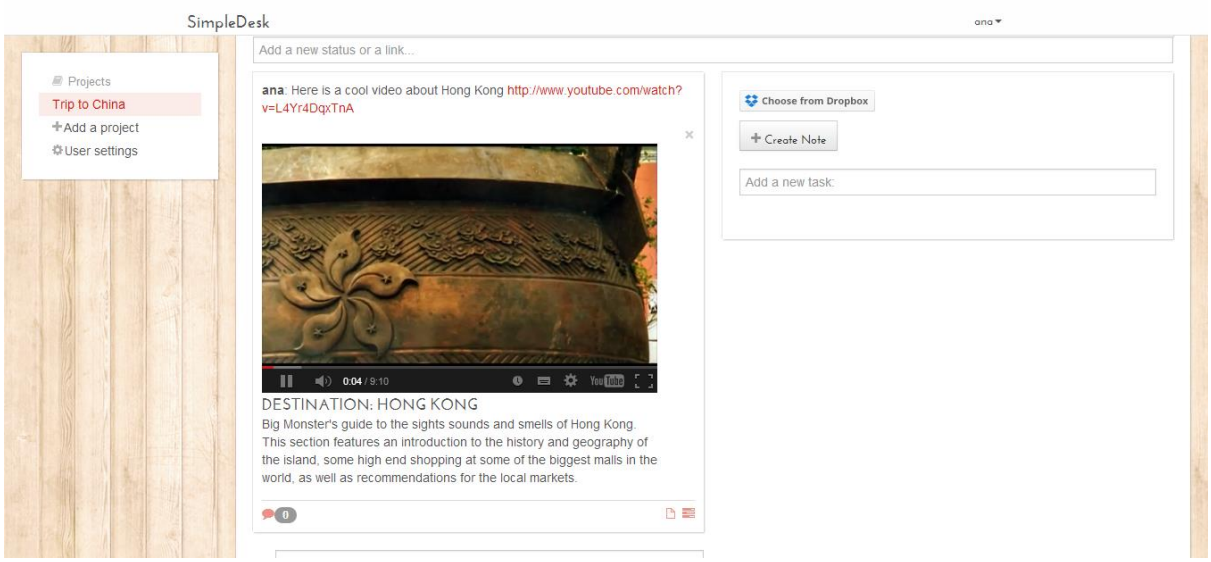
Una vez presionado, se rellena el título y la descripción opcionalmente, y se presiona el botón Add.

Como añadir estados

Después de crear un proyecto este aparecerá en la barra lateral izquierda. Al presionar sobre el proyecto, la aplicación mostrará la pestaña de actividad.



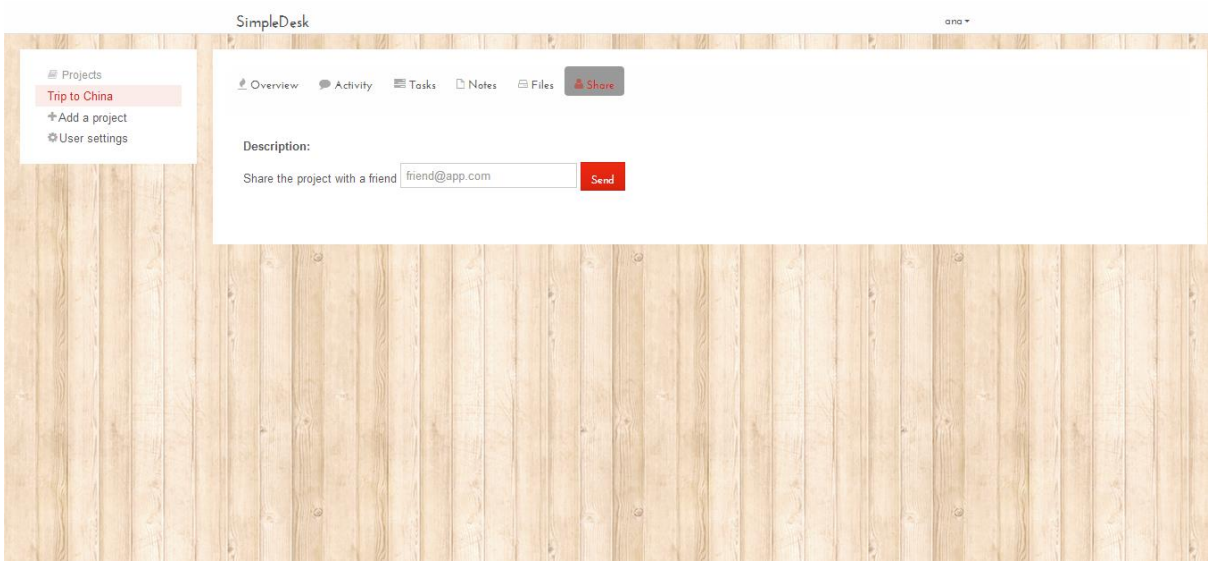
En esta ventana se puede añadir texto o un link. En la imagen superior se puede ver como se ha añadido un texto. Sin embargo también se pueden añadir links a páginas web, videos, lugares de **Google** maps o imágenes. Adicionalmente a la derecha se muestra un menú contextual en el que se pueden añadir notas, tareas o archivos de **Dropbox**.



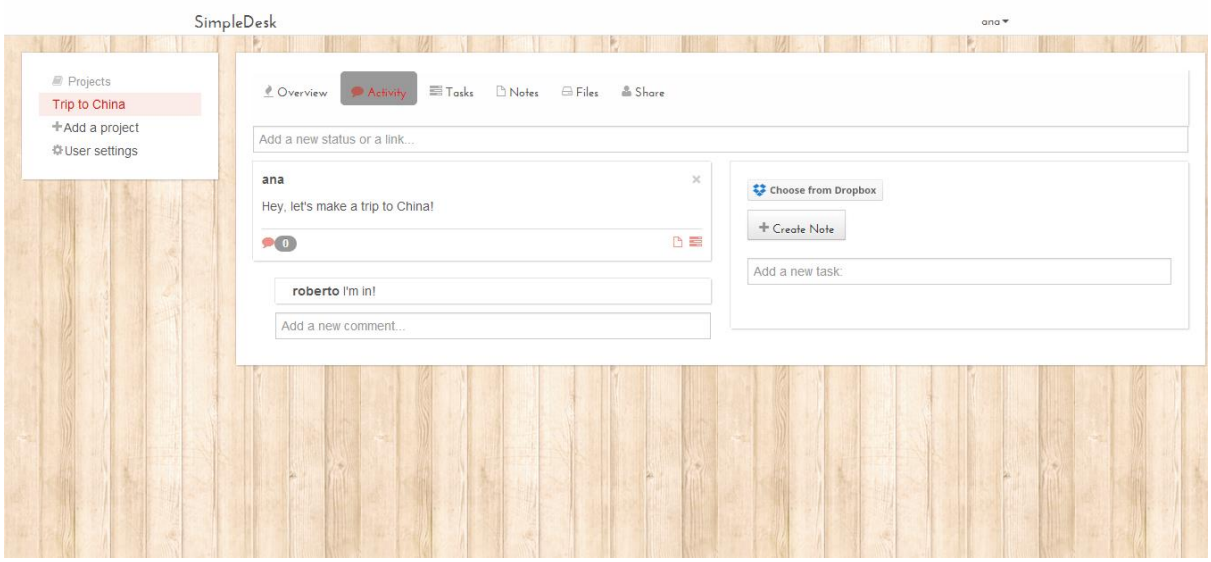
En la imagen superior se puede observar como se ha añadido un link a un video de YouTube, y se ha incluido el video para que pueda ser reproducido directamente.

Como compartir proyectos con otros usuarios

Para compartir un proyecto se pincha sobre la pestaña “Share”.

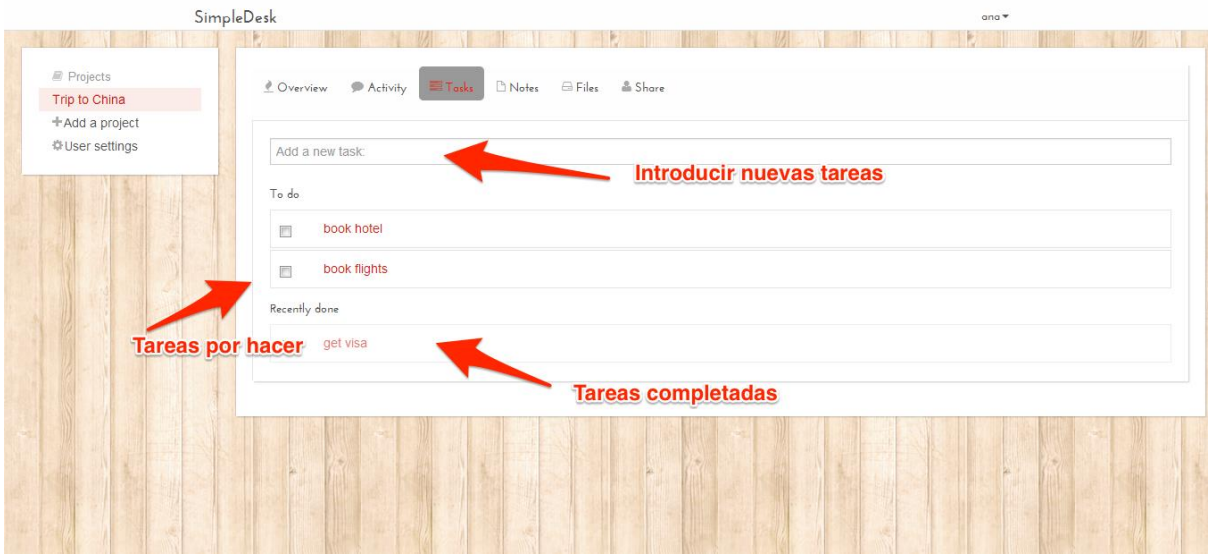


En esta pestaña se introducirá el email del usuario con el que se desea compartir el proyecto. Una vez compartido el usuario con el cual se ha compartido el proyecto podrá ver este en la lista de proyectos.



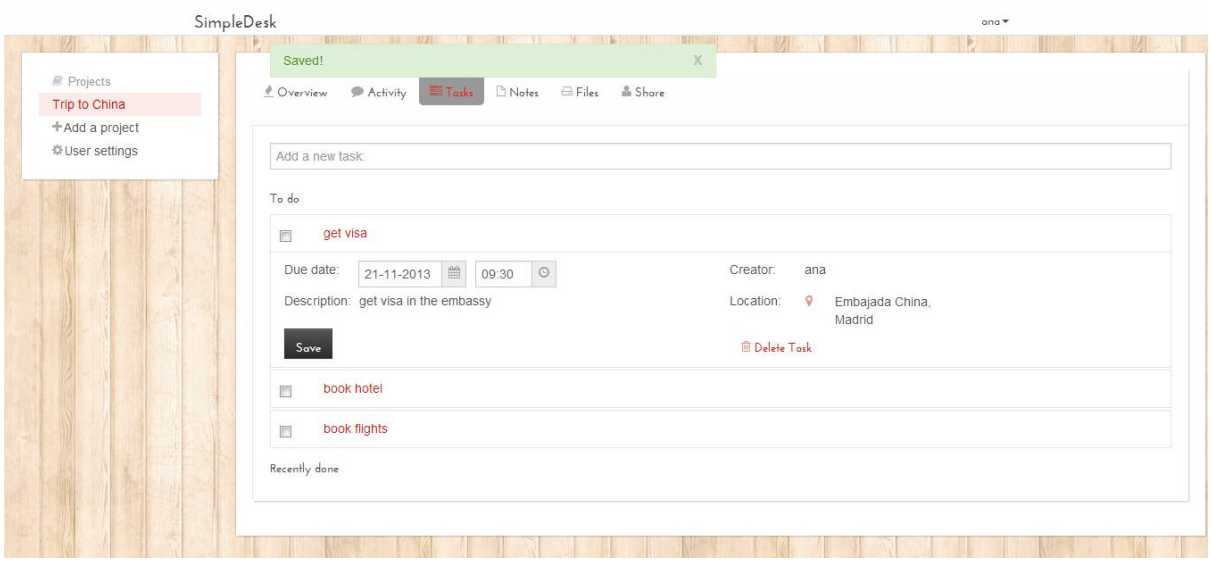
Aquí podemos ver como el usuario “Roberto” ha añadido un nuevo comentario.

Si se desean añadir tareas, el usuario se dirigirá a la pestaña “Tasks”. Desde ahí tiene la posibilidad de añadir una tarea escribiéndola y presionando intro.

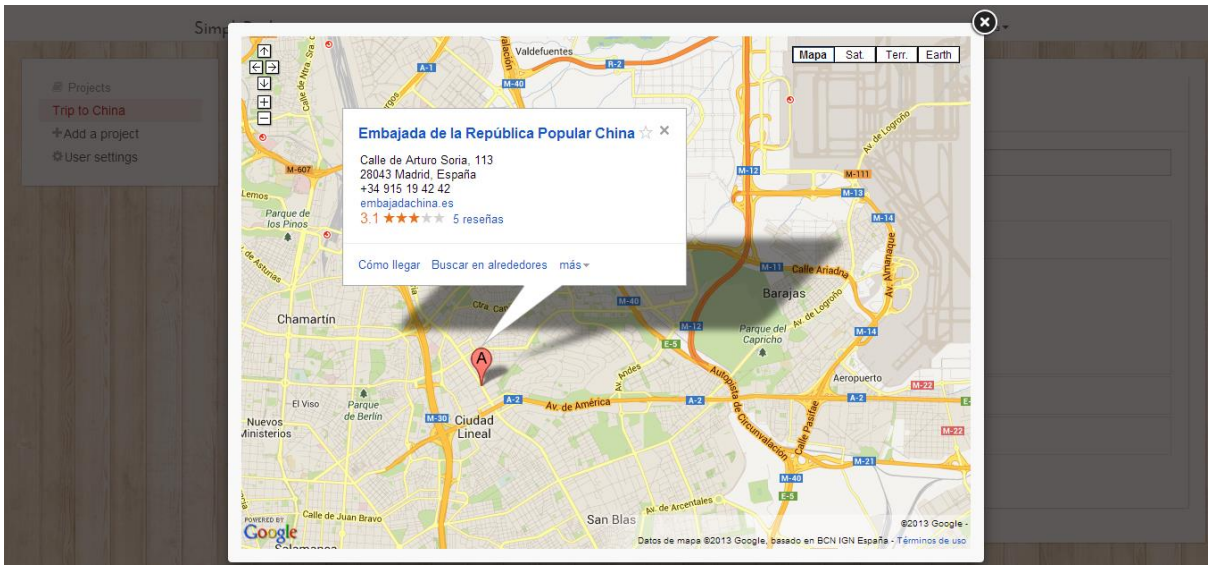


Cuando se considera por finalizada una tarea tan solo hay que presionar en el botón de la izquierda, y la tarea se desplazara a la lista de tareas finalizadas.

Si se desea especificar más información sobre una tarea, como puede ser la fecha de vencimiento o la localización de esta, se presiona sobre la tarea y esta desplegara todas estas opciones.

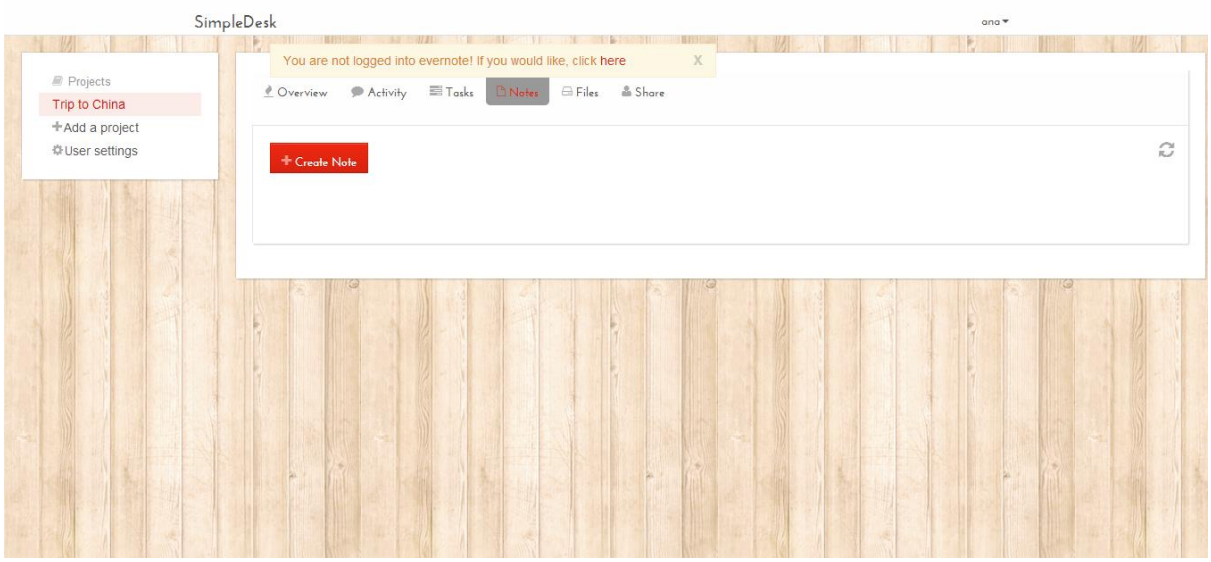


Si el usuario ha añadido una localización al presionar el icono se mostrara un mapa.

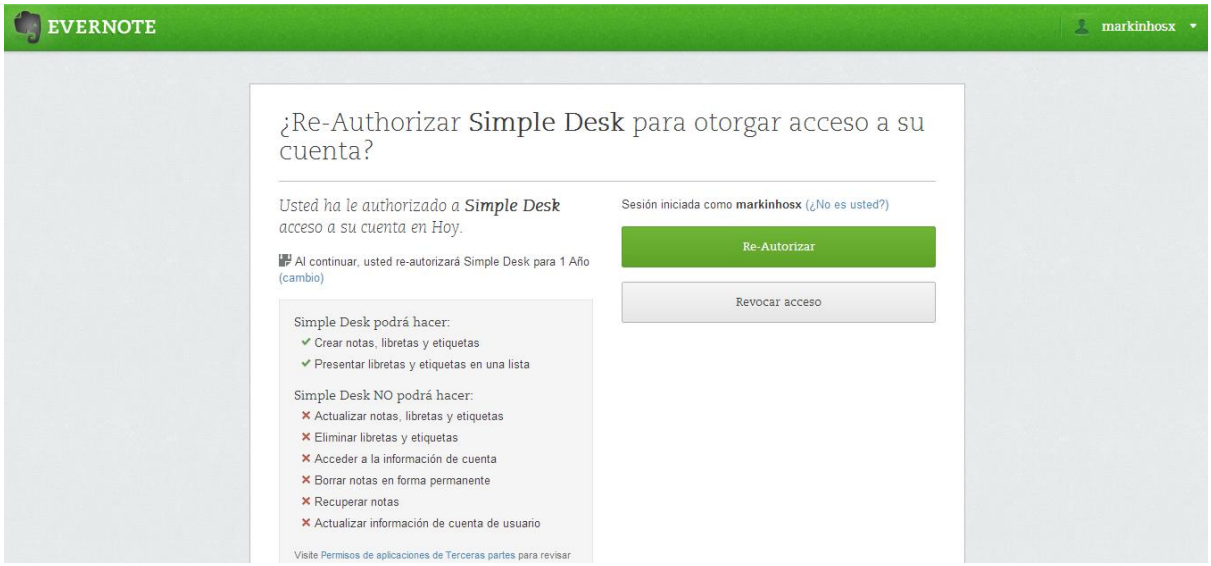


Como añadir notas

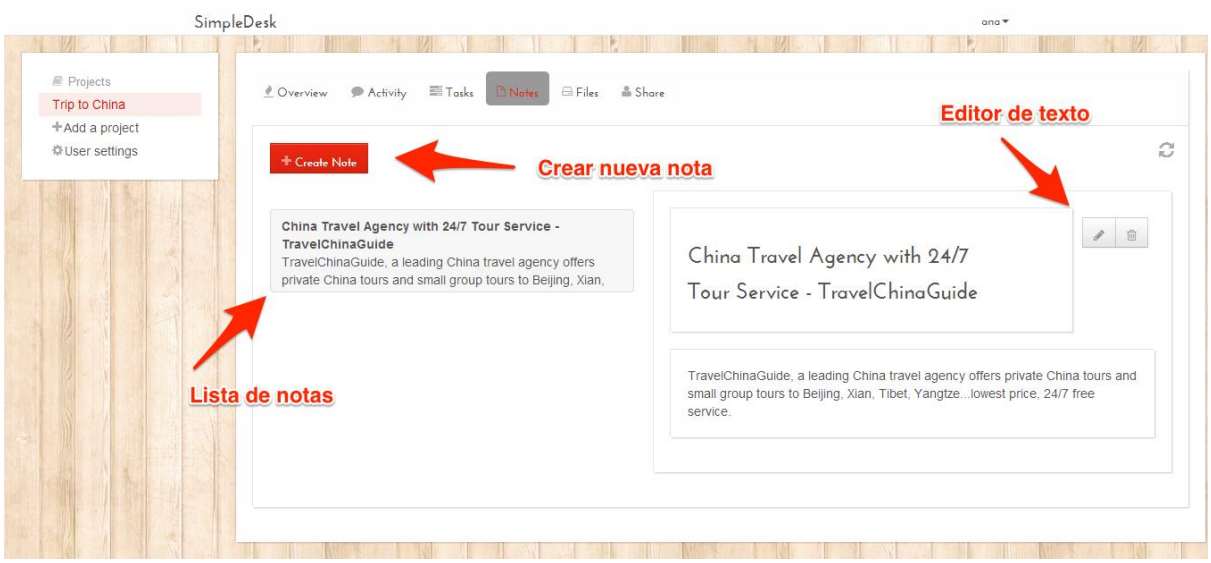
Para añadir o consultar notas, el usuario se dirigirá a la pestaña de notas. En esta pestaña podrá sincronizar su cuenta con **Evernote**, aunque no es obligatorio.

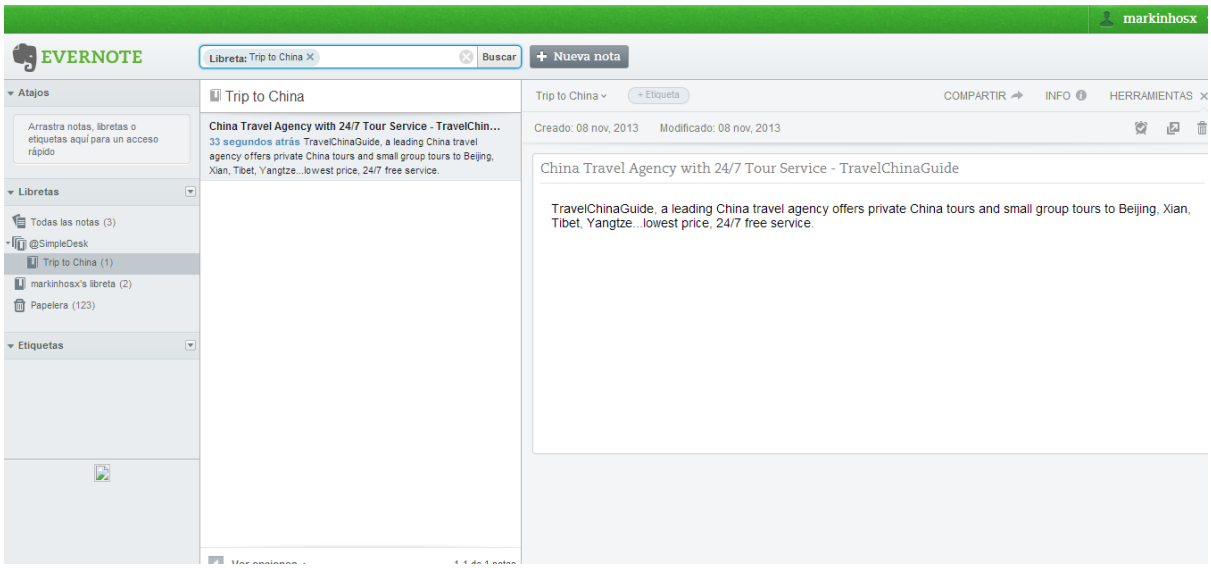


Para hacerlo se ha de presionar el link que aparece flotando. Este link le redirigirá a la página de **Evernote** que permite autorizar la aplicación.



Una vez autorizado el usuario podrá crear notas tanto en la aplicación desarrollada como en **Evernote** y el contenido será sincronizado. Para crear una nota basta con presionar el botón “Create Note” el cual mostrara un editor de texto. Una vez creada la nota se propagara a **Evernote**.

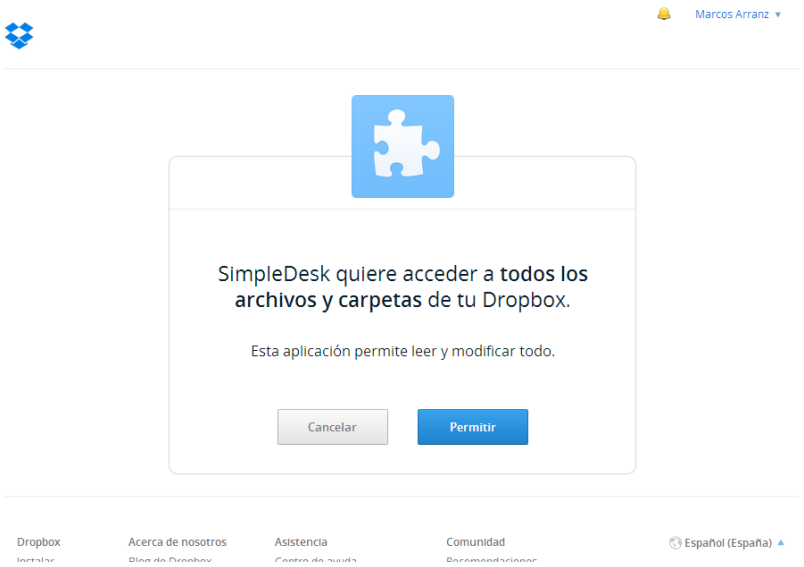




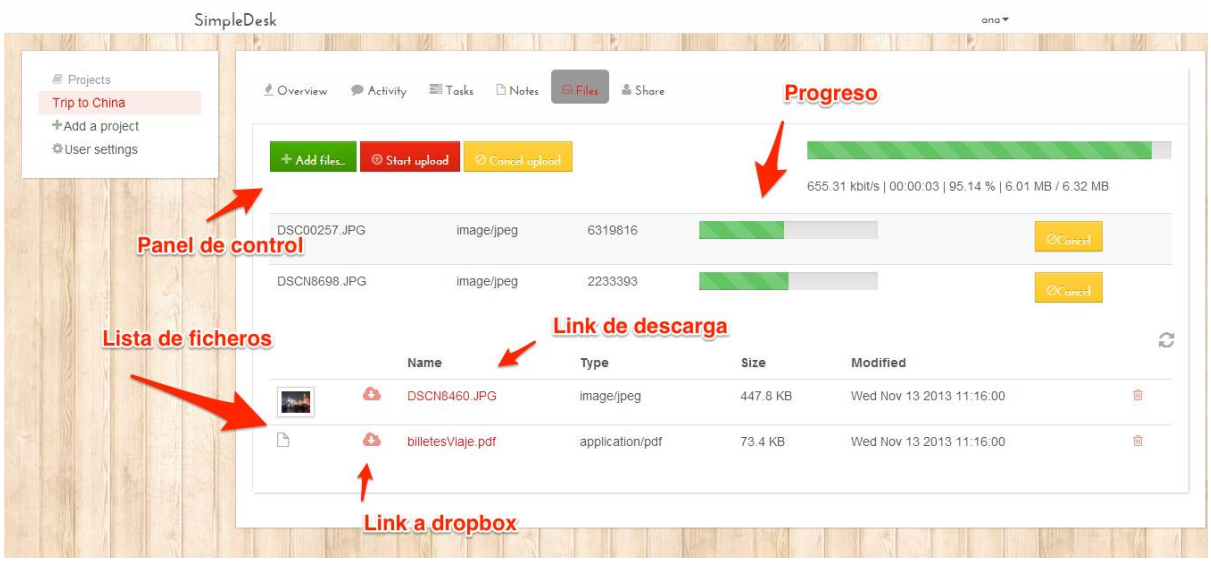
Como se puede ver las notas se crear en una libreta con el nombre del proyecto que esta sobre la pila "@SimpleDesk"

Como añadir ficheros

En la pestaña de ficheros el usuario podrá gestionar los ficheros que puede almacenar en la nube. En este caso es obligatorio sincronizar la aplicación con la cuenta del usuario de **Dropbox**, ya que es donde se van a almacenar todos los ficheros.

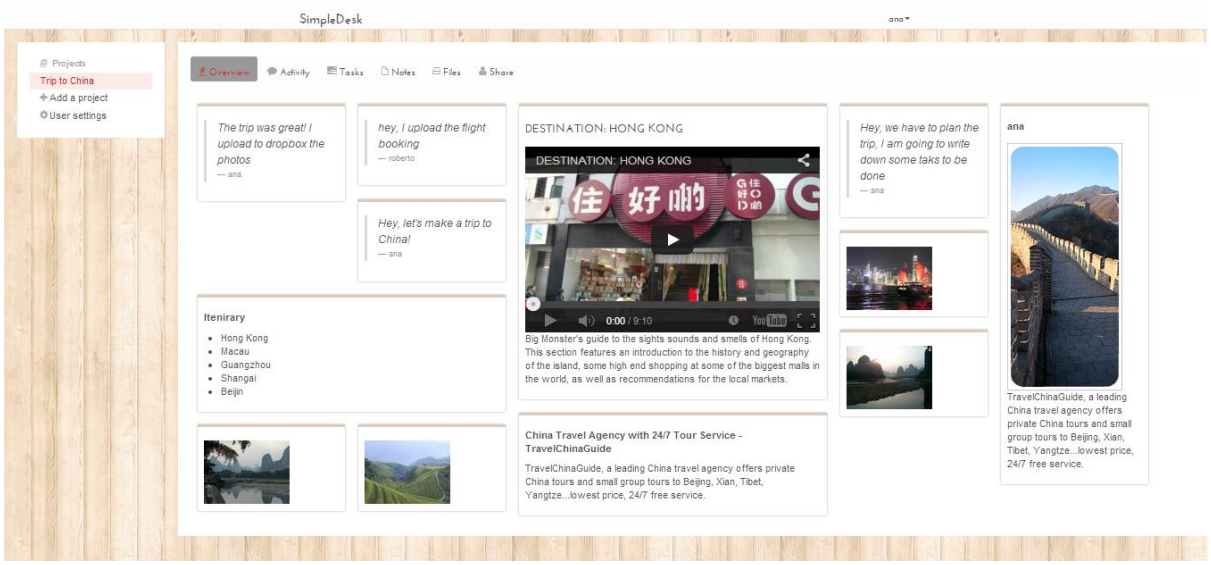


Una vez sincronizadas las cuentas, el usuario puede subir ficheros seleccionándolos o dejándolos caer en la interfaz.



Una vez subidos el usuario puede descargarlos, visualizarlos en caso de ser imágenes, o compartir un enlace directo a **Dropbox**. Todos los ficheros que se suban aparecerán en una carpeta con el mismo nombre que el proyecto bajo la carpeta "SimpleDesk" en la cuenta del usuario de **Dropbox**

Finalmente el usuario puede obtener de un vistazo los elementos del proyecto bajo la pestaña "Overview".



En esta pestaña se muestra un resumen de las fotos añadidas, los estados y las notas.

8 ANEXO II - PRESUPUESTO

1.- Autor:	Marcos Arranz
2.- Departamento:	Informática. Grupo de investigación ARCOS
3.- Descripción del Proyecto:	
- Titulo	SimpleDesk
- Duración (meses)	18
- Tasa de costes Indirectos:	0%
4.- Presupuesto total del Proyecto (valores en Euros):	82.830.198

5.- Desglose presupuestario (costes directos)					
PERSONAL					
Apellidos y nombre	N.I.F.	Categoría	Dedicación (hombres mes) [a)]	Coste hombre mes	Coste (Euro)
		Jefe de proyecto	1725	2.035,00	3.510.375,00
Arranz Mateo, Marcos		Analista	6899	2.035,00	14.039.465,00
		Diseñador	21845	2.035,00	44.454.575,00
		Programador	3450	2.035,00	7.020.750,00
					0,00
		Hombres mes	33919	Total	69.025.165,00

a) 1 Hombre mes = 131,25 horas. Máximo anual de dedicación de 12 hombres mes (1575 horas)
Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 hombres mes (1.155 horas)

EQUIPOS					
Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d)]
Ordenador Portátil	1.200,00	100	18	60	360,00
				Total	360,00

[d] Fórmula de cálculo de la Amortización:	
$\frac{A}{B}xCxD$	A = nº de meses desde la fecha de facturación en que el equipo es utilizado
	B = periodo de depreciación (60 meses)
	C = coste del equipo (sin IVA)
	D = % del uso que se dedica al proyecto (habitualmente 100%)

SUBCONTRATACIÓN DE TAREAS		
Descripción	Empresa	Coste imputable
N/A	N/A	0,00
	Total	0,00

OTROS COSTES DIRECTOS DEL PROYECTO [e]		
Descripción	Empresa	Costes imputable
Material fungible oficina		30,00
Material fungible informático		60,00
Electricidad		180,00
	Total	270,00

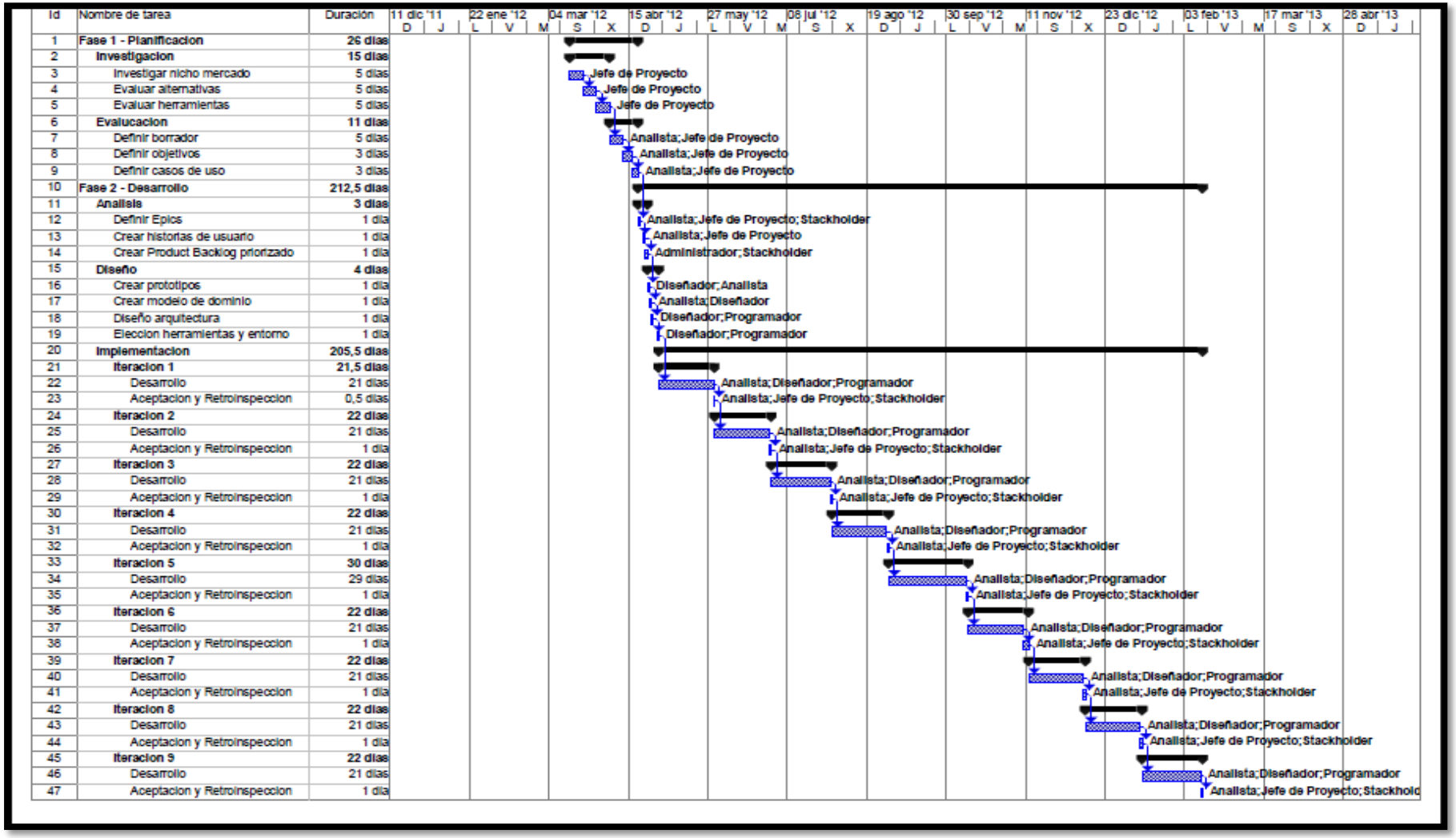
e) Este capítulo de gastos incluye todos los gastos no contemplados en los conceptos anteriores, por ejemplo: fungible, viajes y dietas, otros,…

6.- Resumen de costes	
Presupuesto Costes Totales	Presupuesto Costes Totales
Personal	69.025.165
Amortización	360
Subcontratación de tareas	0
Costes de funcionamiento	270
Costes Indirectos	0
Total parcial	69.025.165
Beneficio	13.805.033
Total	82.830.198

9 ANEXO III – HERRAMIENTAS EMPLEADAS

1 MockFlow – Creación de prototipos - http://www.mockflow.com/
2 Microsoft Word – Procesador de texto
3 Microsoft Project – Software de administración de proyectos.
4 Microsoft Visio – Software de dibujo vectorial.
5 VmWare Player – Software para virtualización.
6 Ubuntu – Sistema operativo basado en Linux.
7 PIP – Sistema de gestión de paquetes para Python.
8 Sublime Text 2 – Editor de texto
9 jQuery – framework para JavaScript para la manipulación del DOM - http://api.jQuery.com/
10 Django-MongoDB-engine – Versión adaptada para MongoDB del popular *framework* Django - http://Django-MongoDB-engine.readthedocs.org/en/latest/
11 Tastypie - Librería para Django para la construcción de interfaces REST - http://Django-Tastypie.readthedocs.org/en/latest/
12 Tastypie-non-rel – Código libre para adaptar Tastypie a Django-nonrel HTTPs://GitHub.com/andresdouglas/Django-Tastypie-nonrel
13 Embed.ly – Servicio comercial y librería JavaScript para contruir empotrar código HTML de otras páginas - http://embed.ly/
14 Dropbox SDK – Librería para la interacción con el API de Dropbox - HTTPs://www.Dropbox.com/developers/core/sdks/Python
15 Evernote SDK – Librería para la interacción con el API de Evernote - HTTPs://GitHub.com/Evernote/Evernote-sdk-Python
16 Gunicorn – Servidor de aplicaciones para Python basado en Unicorn - http://gunicorn.org/
17 Fancybox - Librería para el manejo de imágenes con JavaScript - http://fancybox.NET/
18 Bootstrap – *Framework* CSS y utilidades JavaScript - http://getbootstrap.com/2.3.2/
19 ICanHaz.js – Utilidad para JavaScript para crear plantillas Mustache.js en el lado cliente - http://icanhazjs.com/
20 jQuery-File-Upload – Utilidad para realizar múltiples subidas simultaneas - http://blueimp.GitHub.io/jQuery-File-Upload/
21 Wysihtml5-0.3.0.js – Utilidad JavaScript para crear editores de texto - http://xing.GitHub.io/wysihtml5/
22 Masonry.js - Utilidad JavaScript para crear grid dinámicamente - http://masonry.desandro.com/
23 Backbone.js – *Framework* para la creación de aplicaciones web complejos - http://backbonejs.org/
24 Backbone-relational.js – Complemento para Backbone.js para el uso de relaciones entre modelos y colecciones - HTTPs://GitHub.com/PaulUithol/Backbone-relational
25 Underscore.js – Librería de utilidades para JavaScript del que depende Backbone.js - http://underscorejs.org/
26 Bootstrap-timepicker.js – Selector de tiempo basado en Bootstrap - http://jdewit.GitHub.io/Bootstrap-timepicker/
27 Bootstrap-datepicker.js – Selector de fecha basado en Bootstrap - http://www.eyecon.ro/Bootstrap-datepicker/
28 Less – Preprocesador de CSS - http://lesscss.org/

10 ANEXO IV – DIAGRAMA DE GANTT



Id	Nombre de tarea	Duración
1	**Fase 1 - Planificacion**	**26 días**
2	**Investigacion**	**15 días**
3	Investigar nicho mercado	5 días
4	Evaluar alternativas	5 días
5	Evaluar herramientas	5 días
6	**Evalucacion**	**11 días**
7	Definir borrador	5 días
8	Definir objetivos	3 días
9	Definir casos de uso	3 días
10	**Fase 2 - Desarrollo**	**212,5 días**
11	**Analisis**	**3 días**
12	Definir Epics	1 día
13	Crear historias de usuario	1 día
14	Crear Product Backlog priorizado	1 día
15	**Diseño**	**4 días**
16	Crear prototipos	1 día
17	Crear modelo de dominio	1 día
18	Diseño arquitectura	1 día
19	Eleccion herramientas y entorno	1 día
20	**Implementacion**	**205,5 días**
21	**Iteracion 1**	**21,5 días**
22	Desarrollo	21 días
23	Aceptacion y Retroinspeccion	0,5 días
24	**Iteracion 2**	**22 días**
25	Desarrollo	21 días
26	Aceptacion y Retroinspeccion	1 día
27	**Iteracion 3**	**22 días**
28	Desarrollo	21 días
29	Aceptacion y Retroinspeccion	1 día
30	**Iteracion 4**	**22 días**
31	Desarrollo	21 días
32	Aceptacion y Retroinspeccion	1 día
33	**Iteracion 5**	**30 días**
34	Desarrollo	29 días
35	Aceptacion y Retroinspeccion	1 día
36	**Iteracion 6**	**22 días**
37	Desarrollo	21 días
38	Aceptacion y Retroinspeccion	1 día
39	**Iteracion 7**	**22 días**
40	Desarrollo	21 días
41	Aceptacion y Retroinspeccion	1 día
42	**Iteracion 8**	**22 días**
43	Desarrollo	21 días
44	Aceptacion y Retroinspeccion	1 día
45	**Iteracion 9**	**22 días**
46	Desarrollo	21 días
47	Aceptacion y Retroinspeccion	1 día