Escuela Politécnica Superior (Jaén)

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior (Jaén)

Trabajo Fin de Grado

Estudio y desarrollo de un prototipo para un sistema que facilite la colaboración entre alumnos en el campus universitario.

Alumno: Manuel José Castro Damas

Tutor: Prof. D. José Ramón Balsas Almagro
Dpto: Informática

Septiembre, 2015

Contenido

1.- Introducción

1.1.- Propósito

Siempre como estudiantes hemos tenido la necesidad de obtener alguna información no sólo académica, entendiendo académica como apuntes, esquemas, relaciones de ejercicios… sino también la necesidad de obtener información útil en la vida diaria ya sea buscar un piso, encontrar a alguien para realizar una permuta de turno, compartir coche y un largo etc.

Para suplir estas necesidades la solución más común es recurrir a Internet, pero **Internet plantea un problema dado a su tamaño descomunal**. Internet es una Red de redes que está compuesta por toneladas de información de diferentes tamaños y condiciones. Tomás González, periodista, publicó un artículo [1] que definía la búsqueda en internet como: "una enorme tienda de artículos usados". Resumiendo dice que puede ser divertido y esconde hallazgos sorprendentes, algunos útiles y muchos inservibles, además, nos movemos en un tamaño descomunal sin planos ni indicaciones hacia donde movernos.

Como sabemos se intenta encasillar cada necesidad en puntos diferentes para así obtener las funcionalidades necesarias, ayudando al usuario a obtener lo que necesita. Es decir, necesito compartir unos apuntes voy al dominio de patatabrava.com, necesito compartir coche utilizo blablacar.es… digamos que ésta solución **es una visión vertical**, me voy a una rama del árbol específica para así obtener su funcionalidad específica. Pero ésta rama está cargada de información y/o tal vez lo que necesite esté en otra.

Mi propósito en el trabajo que desarrollo a continuación es crear un prototipo de plataforma, inicialmente como aplicación web, en la cual los alumnos puedan colaborar entre ellos. El objetivo principal es permitir contactar a alumnos con intereses comunes para el intercambio de información variada, información de interés para la comunidad Universitaria como puede ser: crear un grupo de estudio, compartir apuntes, realizar permutas de grupos de teoría o prácticas, compartir piso, compartir vehículos, etc. Plataforma con **una visión más horizontal**, disponiendo así de una plataforma con la información específica de la universidad solucionando el problema de la inmensidad de Internet.

1.2.- Objetivos

Los objetivos siguientes son los que se marcaron en la propuesta presentada a la Escuela Politécnica superior de Jaén para TFG desarrollado a continuación.

● **Realizar un estudio de las necesidades de colaboración entre alumnos de un Campus Universitario.** En la introducción se comenta las necesidades de los alumnos, las cuales pueden ser muy diferentes e incluso pueden cambiar o surgir nuevas.
Por lo tanto nos debemos plantear no sólo crear una plataforma para compartir información o poner en contacto al alumno, sino una plataforma que pueda ser flexible de añadir o suprimir nuevas necesidades del modo más sencillo posible.

● **Diseñar una plataforma web colaborativa para el establecimiento de contactos e intercambio de información entre alumnos.** Los usuarios, en nuestro casos los alumnos no sólo suelen utilizar una única plataforma para acceder a la información por lo tanto, debemos pensar también en la flexibilidad para desarrollar nuestro prototipo en diferentes plataformas, navegador, móviles, etc. Nuestra plataforma como veremos más adelante será una plataforma web que se alimenta de un servicio.

● **Implementar un prototipo utilizando algún framework de desarrollo de aplicaciones web.** Como siempre hay que estudiar las necesidades y circunstancia del proyecto para decidir si usar framework o no. Por regla general suelen facilitar la vida al programador dado que permite realizar fácilmente tareas repetitivas o comunes en múltiples proyectos software de características similares. Además, un framework ya suele estar estudiado y testeado obteniendo una solución eficiente y robusta.

El problema surge cuando por querer facilitarnos la vida utilizamos frameworks complejos para tareas que requieren un diseño mucho más simple, convirtiendo una tarea sencilla en algo pesado e indescifrable. Un ejemplo sería cuando el estudio de un framework suponga mayor tiempo que implementarlo uno mismo. Por lo tanto más adelante describiremos los frameworks posibles a utilizar para nuestro proyecto y la explicación de su uso.

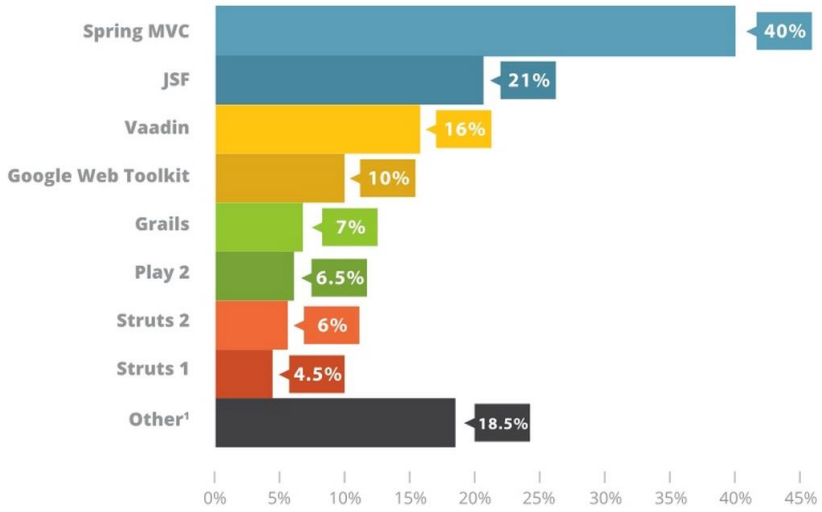


Fig. 1: Frameworks Web Java mas usados (Fuente: http://goo.gl/2px7xB)

En la comparativa anterior (Fig. 1) se muestra el uso en porcentaje de los frameworks de Java más usados [2], no se ha querido entrar en detalle, sólo es mostrada para observar que existen gran variedad y diferentes tendencias. En los apartados siguientes describiremos los diferentes frameworks y no sólo los de Java, sino los frameworks de los diferentes lenguajes que vamos a usar, y se expondrá una breve comparación entre ellos.

1.3.- Metodología

A continuación, pasamos a exponer las metodologías que emplearemos durante el desarrollo de nuestra plataforma:

● **Estudio de los requerimientos del sistema a desarrollar.** Debemos definir y desarrollar los objetivos básicos que nuestro proyecto debe cumplir. Para ello, aplicaremos los conocimientos de la ingeniería de requisitos, identificando así los actores que intervienen en nuestro sistema, los requisitos funcionales y los requisitos no funcionales.

● **Recopilación bibliográfica sobre la temática y tecnologías relacionadas.** Para comprobar la viabilidad, obtener un correcto desarrollo y terminar satisfactoriamente nuestro proyecto, será necesario un estudio de las herramientas y técnicas actuales que puedan servirnos de ayuda.

● **Estimación temporal y de costes del desarrollo.** Dichas estimaciones supone una de las actividades más importantes en el Proceso de Desarrollo de Productos de Software, dado que estas estimaciones nos ayudarán a saber una aproximación del tiempo y coste que tomará el desarrollo. Con ellas podremos ver la **viabilidad** de nuestro proyecto para no desperdiciar los recursos de los que disponemos.

● **Diseño del sistema utilizando una metodología de orientación a objetos utilizando el patrón de diseño Modelo-Vista-Controlador.** Usaremos el lenguaje de programación y las técnicas necesarias para aplicar correctamente una metodología orientada a objeto, que nos ayudará a encapsular los datos y el acceso a éstos. Existen diversos patrones de diseños, que siguen diferentes metodologías de diseño, haciendo nuestro proyecto más robusto como puede ser MVC[1]. Estudiaremos su utilización en nuestro proyecto viendo si el uso de algún **framework** nos puede ayudar a su utilización **tanto en el lado del cliente como del servidor**.

● **Implementación del prototipo del sistema.** Ya solo nos queda desarrollar el software según el diseño realizado, el cual viene desarrollado en la memoria.

1.4.- Estructura del documento

En los siguientes apartados continuaremos con un análisis, diseño e implementación como podemos observar en la tabla de contenido donde además se incluyen los subapartados. Están desarrollado de forma secuencial, pero el desarrollo real del diseño y la implementación han sido de forma iterativa dado que hemos seguido una metodología SCRUM para **un desarrollo ágil** (Todo esto y otros conceptos se irán aclarando más adelante).

En el apartado 2 trataremos sobre el análisis donde daremos una propuesta de solución, seguida de la obtención de requisitos junto una planificación de tarea y el modelado del proyecto. Es decir definiremos totalmente el **qué** debe hacer nuestro proyecto. A comentar que la planificación seguirá algunas técnicas de SCRUM.

[1] MVC: Hace referencia al patrón Modelo-Vista-Controlador.

El siguiente, el apartado 3 es dedicado al diseño donde se especificará el **cómo** debe hacerse, determinaremos la estructura y funcionamiento que tendrá nuestro software a partir de las herramientas obtenidas en el análisis. Este apartado se ha desarrollado de forma iterativa, por lo tanto comentaremos los cambios que han podido surgir en su desarrollo.

Por último antes de las conclusiones tenemos un apartado de implementación, numerado como el cuarto. En él comentaremos los elementos que se incluyen en la implementación, cómo se relacionan y la función que cumplen. El siguiente paso es entrar en detalle de implementación donde hablaremos en profundidad sobre los mismos y describiremos las iteraciones en las que se han llevado a cabo.

Como último apartado, el apartado 5 donde daremos las conclusiones tanto del cumplimiento de los objetivos, como reflexiones personales. Sin olvidarnos, de las posibles mejoras tras su desarrollo.

Finalmente adjuntamos un anexo del manual de instalación, manual de usuario y descripción de contenidos suministrados junto a la memoria, a los que llamaremos apéndice I, II y III correspondientemente.

2.- Análisis

En la introducción donde explicaba mi propósito, se comentó el problema de buscar información y cómo se suele actuar para suplir las necesidades que surgen con ayuda de Internet. En este apartado desarrollaremos un análisis algo más detallado, para así, poder ver el mejor modo para resolver estos problemas comunes y ayudar al estudiante a una colaboración más simple. Daremos una propuesta de solución, seguida de la obtención de requisitos y el análisis.

2.1.- Análisis preliminar

En este apartado desarrollaremos un poco más las necesidades de los estudiantes universitarios en el uso de las tecnologías de la información y comunicación (TIC), tanto en su frecuencia, como también en modo de uso. Viendo las ventajas y debilidades en cada caso.

El siguiente informe [3] nos muestra que los españoles somos los hispanohablantes que más usamos Internet para temas relacionados con estudios. También podemos ver que su uso en éste campo es para obtener documentos tales como apuntes, guías (tutoriales), exámenes, resúmenes y ejercicios. Siendo los apuntes el documento más demandado y los ejercicios el que menos. En cuanto a dispositivos utilizados, como es lógico el más común es el **ordenador con un 98.3%** frente a los móviles y tabletas.

Pero los estudiantes no sólo tienen que lidiar con problemas meramente formativos como apuntes, sino que también surgen problemas administrativos, como puede ser, un cambio de turno y también problemas personales como búsqueda de piso o compartir coche. Podemos ver que **los problemas son variados** y de diferentes clasificaciones, además, tenemos que tener en cuenta que **puede surgir uno nuevo en cualquier momento**.

Cada necesidad puede suponer un problema complejo con infinidad de funcionalidad asociada para ayudar al usuario a obtener lo que busca. Por lo tanto, **siempre se ha ramificado**, y hemos obtenido aplicaciones específicas para que así puedan ser robustas y seguras. Lo que se puede ver como una **vertiente vertical**, desciendes el árbol para ir a la rama donde se satisface tu necesidad. Esto tiene su ventaja pues puedes ofrecer una funcionalidad completa, pero también, su desventaja con el problema intrínseco de internet, su gran volumen. Veamos un ejemplo, supongamos que buscamos compartir piso con estudiantes y que esté cerca de la universidad, una buena opción sería el dominio pisocompartido.com. Dispones de todos los datos bien organizados y el modo de contactar, pero los problemas que podemos encontrar son por ejemplo: la oferta del piso que más te puede interesar esté en otro dominio, necesitas registrarte, puede tener un coste por la página inmobiliaria, anuncios…

Todo esto nos lleva a pensar en **obtener una solución más horizontal**, ya no ser algo tan específico sino algo más centrado en el campus universitario. La ventaja está clara, quieres algo relacionado con el campus universitario, vas a la aplicación de tu campus y la información obtenida será la correspondiente a tu campus.

En este punto, ya tenemos definida la motivación de realizar una aplicación de colaboración en el campus universitario, pero también podría ser útil en cualquier otro campo similar en el que se necesite compartir información o abastecer una demanda. Ahora veamos cuál sería el mejor modo:

1. Deberíamos tener en cuenta el factor de cambio y que la aplicación pueda ser extensible.
2. Que pueda ser añadida una nueva demanda de forma rápida y sencilla.
3. También, que pueda ser utilizado en distintos dispositivos.

Para hacer posible los puntos anteriores o facilitar la obtención de la mejor solución debemos estudiar las siguientes tecnologías o frameworks.

2.1.1. Aplicaciones orientadas a servicios web

Un servicio web es un método de comunicación entre dos dispositivos electrónicos a través de la World Wide Web (Red informática mundial) mediante el uso de una serie de estándares abiertos, como puede ser XML[2], SOAP[3] y WSDL[4] sobre los protocolos de Internet. Gracias a estos estándares, es posible el intercambio de información entre dos sistemas, que no tienen por qué conocer los detalles de sus respectivos sistemas de información.

En cuanto a arquitectura para montar nuestros servicios, tenemos dos grandes vertientes como son SOAP y REST (Representational State Transfer). El punto identificativo de **SOAP** es que la comunicación se realiza con operaciones definidas como puertos de WSDL. Siendo aconsejables en entornos donde se establece un contrato formal, donde se puede definir todas las funciones y los tipos de datos utilizados tanto en la entrada como en la salida.

REST a diferencia de la arquitectura anterior se centra en el uso de los estándares de HTTP (HyperText Transfer Protocol) y XML para transmitir los datos, las operaciones se solicitan con los métodos GET, POST, PUT y DELETE, sin la necesidad de requerir ninguna implementación especial para consumir los servicios. Esta arquitectura es aconsejable cuando buscamos mejorar el rendimiento, o disminuir el consumo de recursos como puede ser en dispositivos móviles.

Como impedimentos de los servicios web podemos mencionar que no ofrece una interfaz de usuario, un usuario no podrá acceder directamente al servicio proporcionado. Son programas conectados entre sí, por lo tanto es tarea del programador el realizar una interfaz de usuario para la conexión de éste con el servicio. Cierto es que hay otros modos de probar un servicio como puede ser *Advanced Rest Client* [4] para los servicios con arquitectura REST. Advanced Rest Client es una aplicación de Google Chrome que te proporciona una interfaz muy simple para utilizar los métodos de conexión proporcionados por ésta arquitectura.

[2] XML: Viene del inglés eXtensible Markup Language, siendo un lenguaje de marcas utilizado para almacenar datos de forma legible.
[3] SOAP: Simple Object Access Protocol.
[4] WSDL - Web Services Description Language: Está en lenguaje XML describiendo la forma de comunicación y los mensajes necesarios para interactuar con los servicios.

Para SOAP podemos utilizar el estándar de Java que es *JAX-WS, basadas en anotaciones con poca o ninguna configuración adicional.* Una medida muy extendida es utilizar *Spring Framework* que hace un gran trabajo para la gestión de transacciones, inyecciones de dependencias, persistencia y creación de servicios web entre otras cosas.

Para REST también se puede utilizar el estándar *JAX-RS* o integrarlo con *Spring Framework* pues en REST no hay una implementación del estándar. Un buen framework de Java para Servicio Web REST es *Jersey* [5] que si utiliza el estándar JAX-RS que se incluye con GlassFish y siendo de código abierto, ha sido diseñado para facilitar la construcción de un servicio web Restful utilizando Java y la máquina virtual de Java.

2.1.3. Frameworks de programación en el cliente

Como vimos en aplicaciones orientadas a servicio web, un servicio no puede ser usado directamente por el usuario, suelen ser usados por otro programas que si toman contacto con el cliente. A estos programas se le suele denominar la programación del lado del cliente o programación en el cliente, y surge la pregunta que todo programador se hace siempre ¿Qué lenguaje utilizar?

Un posible enfoque podría ser buscar algo que sea común y que te permita hacer aplicaciones multiplataforma, que puedan ejecutarse sobre la mayoría de sistemas operativos y dispositivos. Podríamos decantarnos por HTML5 + Javascript compitiendo con las aplicaciones nativas que, como es lógico, siempre será mejor en rendimiento y acceso en cada una de las funcionalidades de los dispositivos. Pero aun utilizando estos dos lenguajes juntos, surgen algunas dificultades para desarrollar grandes aplicaciones de negocios. Un buen enfoque para facilitar estas dificultades es buscar algún framework que nos ayude a hacer un buen diseño en el cliente. El framework nos debe permitir utilizar el patrón MVC, que nos ayudará a responder a los cambios que irán surgiendo en el servicio. Haciendo posible que esta parte de la aplicación también pueda adaptarse a los cambios y complejidad que surgirán a lo largo del tiempo.

Si nos decantamos por HTML5 y Javascript muchos programadores coinciden que los cincos mejores frameworks [6] que además proporcionen MVC son *Backbone*, *Ember*, *Angular*, *Knockout*, *JavaScript MVC*. Por elección personal consideraremos que los tres primeros son los mejores, siendo los que explicaremos y compararemos a continuación. En común tienen como he dicho antes que siguen un patrón MVC, son de código abierto permitiendo usarse bajo la MIT license[5] y pueden resolver el problema de crear aplicaciones web de una página (Single Page Web Applications).

Backbone se hizo popular rápidamente debido a que era una buena opción a los pesados frameworks de la época de ExtJs, es decir, es un framework muy ligero. Pero la ligereza tiene su punto débil, haciéndolo repetitivo, viéndose evidente en grandes aplicaciones.
Ember su origen se remonta al 2007 con el nombre *SproutCore MVC* pero fue tomado su relevo en 2011 llegando a lo que ahora es. Asume múltiples opciones de configuración siguiendo ciertas conversiones con el fin de acelerar el desarrollo, posee un gran motor de plantillas. Pero aún no dispone de una buena documentación que facilite su aprendizaje.

[5] Licencia MIT: Es una licencia software que ha empleado el Instituto Tecnológico de Massachusetts.

Angular es mantenido por Google, destacamos que tiene la habilidad de agrupar tecnologías aun cuando estas sean incompatible gracias a su sistema de inyección de dependencia. Dispone de una gran documentación y su sintaxis es muy fácil de aprender, como defecto podremos decir que es bastante pesado su código base y no muy modular.

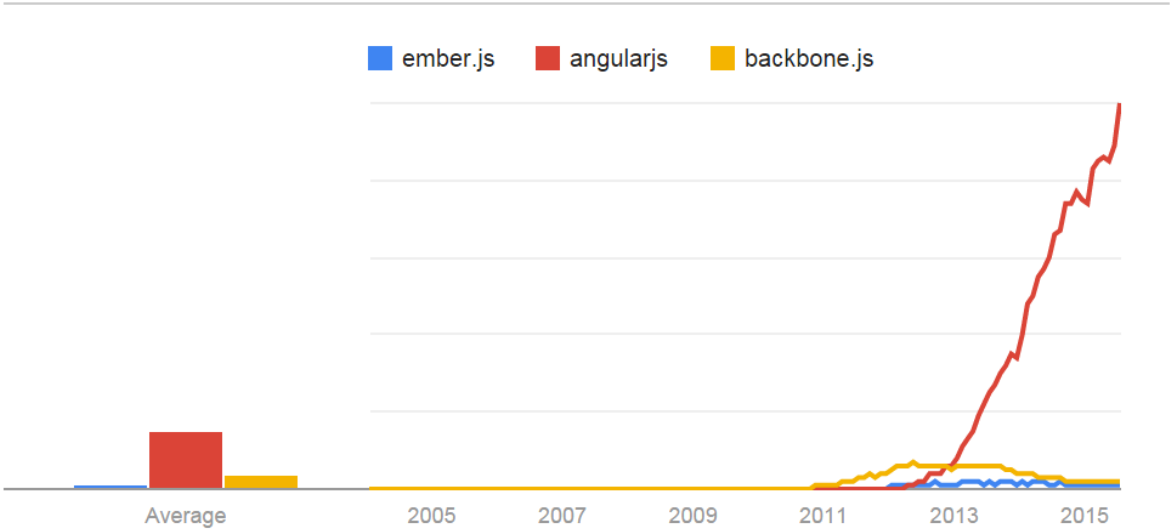


Fig. 2: Comparativa frameworks Javascript (Fuente: https://www.google.es/trends/)

2.1.3. Patrones de diseño en aplicaciones web

Uno de los criterios de la metodología de desarrollo de la aplicación (Punto 1.3.) era el uso del patrón de arquitectura ***Modelo-Vista-Controlador*** **(MVC)** [7]. Con éste patrón dividimos el sistema en tres capas donde tenemos, la encapsulación de los datos, la interfaz o vista por otro lado y por último la lógica interna o controlador.

El patrón de arquitectura "modelo-vista-controlador", es una filosofía de diseño de aplicaciones, compuesta por:

- Modelo: Encapsula los datos y las funcionalidades. El modelo es independiente de cualquier representación de salida y/o comportamiento del controlador o la vista.
- Vista: Muestra la información a través de una interfaz de usuario. Pueden existir múltiples vistas del modelo. Cada vista tiene asociado un componente controlador con el que interactúa.
- Controlador: Reciben las entradas de las vistas, usualmente como eventos que codifican los movimientos o pulsación de botones del ratón, pulsaciones de teclas, etc.

En el Modelo como vemos, tenemos una encapsulación de la funcionalidad y datos, pero los datos están en la base de datos, por lo tanto, vamos a dar un paso más en el encapsulamiento y en la forma de acceder a la fuente de datos con el **patrón DAO** [8]. Digamos que un DAO (Data Access Object) define la relación entre la lógica de representación y la de negocio por una parte y por otra la relación con la capa de datos. Ofreciendo una interfaz común, sea cual sea el modo y la fuente de acceso a datos.

Cómo adicional explicaré por encima el modo de usar el patrón MVC con *AngularJS,* definiendo las componentes de cada encapsulación del modelo:

- Vistas: Será el HTML y todo lo que represente datos o información.
- Controlador: Se encargará de la lógica de la aplicación y sobre todo de las llamadas "Factorías" y "Servicios" para mover datos contra servidores o memoria local en HTML5.
- Modelo de la vista: En angular el "Modelo" es algo más de aquello que se entiende habitualmente en MVC tradicional, las vistas son algo más que el modelo de datos. En una pantalla concreta es posible que tengas que ver otra información además de la principal de la pantalla, útil sin formar parte del modelo de negocio, es a lo que llamamos el "Scope" que es el modelo en Angular.

Todo esto es una breve introducción, un análisis preliminar, más adelante lo veremos mejor y en mayor detalle en la implementación.

2.2.- Requerimientos del sistema

En éste apartado se define los requisitos que debe cumplir el prototipo a desarrollar siendo el inicio del análisis y el posterior diseño. Diferenciaremos entre los **requisitos funcionales** que describen los servicios (funciones) que se esperan del sistema y los **requisitos no funcionales** siendo las restricciones sobre los funcionales.

Para la obtención de requisitos de un proyecto, dado que los errores en esta fase son un punto crítico durante el desarrollo del proyecto, debemos aplicar una base de ingeniería denominada **ingeniería de requisitos**. La cual nos da las técnicas necesarias para una correcta obtención de los requerimientos del software a realizar, con las fases de captura, definición y validación. El cliente "nunca sabe lo que quiere" y con todo éstos nos aseguramos mediante entrevistas, cuestionarios, tormentas de ideas,… capturar todos los requerimientos. A continuación con, por ejemplo, las revisiones y prototipos vamos validando los requerimientos.

Pero en nuestro caso no será necesario aplicar una ingeniería de requisito en sí, el cliente seré yo mismo, dado que soy estudiante y he podido durante años tratar con otros estudiantes universitarios. La obtención de requisito se basará principalmente en la experiencia como estudiante, experiencia académica y experiencia laboral que he obtenido durante estos años. A continuación paso a expresar los requerimientos del problema descrito hasta ahora en la memoria.

2.2.1. Requisitos funcionales

Los requerimientos funcionales son declaraciones de los servicios que debe proporcionar el sistema, de la manera en que éste debe reaccionar a entradas particulares y de cómo se debe comportar en situaciones particulares. En algunos casos, también pueden declarar explícitamente lo que el sistema no debe hacer.

Para nuestro caso se definen los siguientes requisitos funcionales:

- El sistema debe permitir la gestión de información a compartir (noticias, apuntes...) y poder realizar con ella las siguientes operaciones:
    - Gestionar la información, permitiendo operaciones de creación (Create), lectura (Read), actualización (Update) y borrado (Delete), conocido en inglés como CRUD.
    - Ver la información que otro usuario haya puesto para compartir.
    - Un usuario no podrá editar o borrar la comparticiones de otro usuario.
- El sistema debe permitir operaciones de registro:
    - Si el usuario no está registrado debe tener un modo para poder hacerlo.
    - Si el usuario ya está registrado esta opción no debería volverse a repetir.
    - Debe tener la opción de darse de baja al sistema.
- No se podrá realizar ninguna operación con la información compartida sin loguearse (identificarse) para ello:
    - Se dispondrá de un punto de acceso.
    - Si ya se ha identificado no será necesario volver a hacerlo.
    - Existirá la opción de logout.
- Los usuarios identificado como administrador tendrán permisos total en el sistema siendo:
    - Se le permite una gestión CRUD de la información que es propietario y borrar o editar la información compartida por otro usuario.
    - Se le permite una gestión CRUD de su información de registro y borrar o editar la información de registro de cualquier otro usuario.
- La aplicación inicialmente compartirá información y ofrecerá la funcionalidad de compartir noticias y compartir apuntes.

2.2.1. Requisitos no funcionales

Los requerimientos no funcionales son restricciones de los servicios o funciones ofrecidos por el sistema. Como su nombre sugiere, son aquellos requerimientos que no se refieren directamente a las funciones específicas que proporciona el sistema, sino a las propiedades emergentes de éste como la fiabilidad, el tiempo de respuesta y la capacidad de almacenamiento.

Para nuestro caso se definen los siguientes requisitos no funcionales:

- La aplicación no solo tendrá la información compartida inicialmente, conforme las necesidades de los estudiantes cambie, ésta deberá poder cambiar de un modo sencillo, rápido y eficaz.
- La información compartida tendrá unos datos comunes y otros adicionales de todo tipo, deberá unificar estos datos comunes para un mejor uso de los recursos y una gestión más fácil de las nuevas funcionalidades.
- La aplicación se dividirá en servicio web y aplicación de cliente web, disponiendo así de la capacidad de funcionar en distintos dispositivos con distintos sistemas operativos, además de la capacidad de desarrollar una posible aplicación nativa si fuese necesario.
- La aplicación será accesible desde cualquier dispositivo con un navegador web compatible y por distintos usuarios simultáneamente.

- La interfaz se desarrollará usando los principios de adaptabilidad para una mejor visualización en los distintos dispositivos en el que pueda ser accesible.
- Se debe validar que todos los campos obligatorios de los formularios estén completos para evitar errores innecesarios, disminuyendo así la carga en el servidor.

2.3.- Propuesta de solución

Como hemos visto en puntos anteriores, el mejor modo de realizar la solución debería cumplir:

Punto 1 Tener en cuenta el factor de cambio y que la aplicación pueda ser extensible.

Punto 2 Que pueda ser añadida una nueva demanda de forma rápida y sencilla.

Punto 3 Que pueda ser utilizado en distintos dispositivos.

Un buen modo de que se pueda utilizar en distintos dispositivos, es crear un Servicio Web REST utilizando algún framework o biblioteca que permita un desarrollo ágil y robusto del servicio. Con el servicio REST podremos elegir una programación del lado del cliente en infinidad de plataformas y dispositivos cumpliendo el Punto 3. Pero el Punto 3 no estaría completo sin la programación del lado del cliente, y como es lógico, el lenguaje a utilizar dependerá del caso demandado. Ya vimos en el análisis preliminar que el mejor modo es elegir una Aplicación Web que con ayuda de algún framework, ya sea de lenguaje nativo o no, nos ayudará a la aplicación del patrón de diseño MVC y así tener una arquitectura flexible y robusta en la programación del cliente.

Respecto a los puntos 1 y 2, ambos están muy ligados y estos puntos debería tenerse en cuenta en el diseño, además de las tecnologías a usar en ella, no hacerlo puede perjudicar el resultado final. Por lo tanto, buscaremos el modo de programar lo más sencillo posible, y así cumplir estos puntos, para ello usaremos el **patrón de diseño MVC** (tanto en el servicio como en la aplicación cliente). Vuelvo a repetir que no debemos olvidar estos puntos en el diseño, será crucial para conseguir lo que buscamos.

2.4.- Planificación de tareas

Antes de la planificación de tareas he de decidir la metodología a utilizar. Es decir, una metodología tradicional en cascada como se ha estado haciendo normalmente hasta ahora, o utilizar una metodología ágil como una tendencia creciente en estos últimos años. Para ellos yo me he decidido desde el primer momento por la metodología ágil y expondré brevemente el porqué de mi decisión.

Las metodologías tradicionales son aconsejables para proyectos grandes, con requisitos estables, grandes equipos de desarrollo y con equipos distribuidos. Pero en mi caso tengo un proyectos de pocas horas de trabajo (300 horas aproximadamente), distribuidas en unos 4 o 5 meses donde solo hay única persona encargada para el análisis, diseño e implementación.

La metodología ágil [9] es más apropiada para el presente proyecto, ya que proporciona un ambiente más dinámico para un equipo de desarrollo de menor tamaño. La metodología a utilizar será Scrum desarrollando el proyecto por medio de iteraciones (Spring),

permitiendo cambiar los requerimientos rápidamente si fuese necesario y donde me "reuniré" con el tutor a lo largo del proyecto. Yo estaré formando parte como actor en todos los actores posible de esta metodología: Propietario del producto, usuario del producto, scrum master y equipo scrum.

2.4.1. Product Backlog

En la metodología ágil trabajamos con puntos de historias y días ideales, los puntos de historias se suelen utilizar para la estimación al expresar el tamaño de trabajo utilizando valores relativos. Empezamos con la planificación, en primer lugar definimos el Product Backlog del proyecto, en él se define los requisitos del proyectos a través de las historias de usuarios con su complejidad anotada como puntos de historia. Además se ordenan según el valor de negocio que representa la historia de usuario.

Para la identificación de las historias de usuario, se agruparán por sus características obteniendo así el id por su propiedad y situación. Además diferenciaremos entre servicio web (SW) o aplicación web (AW).

1. Acceso

ID	Historia de usuario	Criterio de aceptación	Puntos
AC1SW	**Como** desarrollador **Quiero** modelar el perfil de usuario **Para poder** gestionar su información e identificar al usuario	a) Clase con información de persona. b) DAO correspondiente. c) Mapeado correspondiente.	4.5
AC2SW	**Como** desarrollador **Quiero** disponer de un servicio personas **Para poder** hacer un CRUD sobre la información de persona	a) Respetar la arquitectura REST.	3.5
AC3SW	**Como** desarrollador **Quiero** modelar los tokens **Para poder** gestionar la seguridad de las peticiones al servidor	a) Clase con información del token. b) DAO correspondiente. c) Mapeado correspondiente.	6.0
AC4SW	**Como** desarrollador **Quiero** disponer de un servicio acceso **Para poder** logearme y hacer logout	a) Respetar la arquitectura REST. b) Comprobar datos del usuario y enviar el token, error en caso contrario. c) Borrar el token en logout.	5.0

AC5AW	**Como** usuario **Quiero** registrarme **Para poder** logearme y entrar al sistema	a) El usuario y el correo debe ser de la universidad y por lo tanto coincidir en el nick. b) Las contraseñas deben coincidir. c) Todos los campos obligatorios han de estar rellenos.	6.0
AC6AW	**Como** usuario **Quiero** logearme **Para poder** identificarme y acceder al sistema	a) Todos los campos obligatorios rellenos antes de enviar la comprobación. b) Si es correcta accede al sistema. c) Si no es correcta se mostrará un mensaje de error.	5.0
AC7AW	**Como** desarrollador **Quiero** guardar el token **Para poder** disponer de él en las llamadas al servidor	a) Guardarla como cookie para disponer del token en todo momento.	12.0
AC8AW	**Como** usuario **Quiero** hacer logout **Para poder** salir de la aplicación	a) Se borra el token de la cookie. b) Se elimina en el servicio. c) Mostramos por pantalla el login.	4.0

2. Interfaz

ID	Historia de usuario	Criterio de aceptación	Puntos
INT1AW	**Como** usuario **Quiero** tener una interfaz que siga como base la IPO **Para poder** optimizar mi tiempo de aprendizaje y uso de la aplicación	a) Mantener al usuario el estado en todo momento y darle opción de volver al paso anterior. b) Estudio de los colores a usar.	15.0
INT2AW	**Como** desarrollador **Quiero** tener una interfaz bien estructurada con los controladores **Para poder** gestionar las vista de un modo eficaz		15.0
INT3AW	**Como** usuario **Quiero** tener una interfaz vistosa **Para poder** disfrutar de su uso	a) Usar en la mayor medida los principios de Material Design.	10.0

3. Arquitectura

ID	Historia de usuario	Criterio de aceptación	Puntos
ARQ1SW	**Como** desarrollador **Quiero** disponer de una arquitectura que sea estructurada y ordenada **Para poder** mantener un orden en el proyecto en todo momento	a) Buenas prácticas en la programación.	5.5
ARQ2SW	**Como** desarrollador **Quiero** disponer de una arquitectura que sea flexible **Para poder** introducir nuevas funcionalidades de diferentes tipos	a) Que sea modular y encapsulada. b) Con poco esfuerzo pueda añadir una nueva funcionalidad.	12.5

4. Colaboraciones (Noticias y apuntes)

ID	Historia de usuario	Criterio de aceptación	Puntos
COL1SW	**Como** desarrollador **Quiero** modelar un aporte simple como noticia **Para poder** hacer un CRUD sobre la información del aporte	a) Clase con información de aporte (noticia). b) DAO correspondiente. c) Mapeado correspondiente.	8.0
COL2SW	**Como** desarrollador **Quiero** disponer de un servicio para las noticias **Para poder** disponer de una interfaz del CRUD y otras funcionalidades sobre la información de las noticias	a) Respetar la arquitectura REST.	6.0
COL3SW	**Como** desarrollador **Quiero** modelar un aporte complejo como apuntes **Para poder** hacer un CRUD sobre la información del aporte complejo	a) Clase con información de aporte complejo (apunte). b) DAO correspondiente. c) Mapeado correspondiente.	9.0
COL4SW	**Como** desarrollador **Quiero** disponer de un servicio para los apuntes **Para poder** disponer de una interfaz del CRUD y otras funcionalidades sobre la información de los apuntes	a) Respetar la arquitectura REST.	7.0

COL5AW	**Como** usuario **Quiero** ver todas las noticias **Para poder** encontrar las noticias en las que esté interesado	a) Se visualizarán las más recientes primero. b) Se dispondrá de un breve resumen de cada una de ellas. c) Debería existir una paginación.	12.0
COL6AW	**Como** desarrollador **Quiero** gestionar la paginación **Para poder** facilitar la visualización al usuario y reducir el peso de las peticiones	a) Dos noticias por página. b) Evitar las páginas sin noticias. c) Disponer de elección de página en el pie del listado.	8.0
COL7AW	**Como** usuario **Quiero** abrir una noticia **Para poder** acceder a la noticia entera	a) Visualizarla en una nueva página con toda la información oportuna.	4.0
COL8AW	**Como** usuario **Quiero** crear una noticia **Para poder** colaborar con información de interés	a) Todos los campos obligatorios han de estar rellenos. b) Se pueden subir imágenes directamente desde tu ordenador.	12.0
COL9AW	**Como** usuario **Quiero** ver la lista de mis noticias **Para poder** editarlas o borrarlas	a) Mostrarla en una tabla resumen. b) Ver las opciones disponibles con fácil acceso.	7.5
COL10AW	**Como** usuario **Quiero** editar mi noticia **Para poder** modificar un dato erróneo	a) Todos los campos obligatorios han de estar rellenos. b) Ofrece la posibilidad de cancelar.	5.5
COL11AW	**Como** usuario **Quiero** eliminar mi noticia **Para poder** hacer que no se visualice más	a) Mostrar un modal de aceptación.	10.0
COL12AW	**Como** usuario **Quiero** buscar mi carrera **Para poder** encontrar apuntes sobre mis estudios	a) Se mostrarán todas las carreras. b) Con un buscador podremos agilizar la búsqueda.	12.0
COL13AW	**Como** usuario **Quiero** ver los cursos disponibles de mi carrera **Para poder** acceder a las asignaturas del curso que me interese	a) Se mostrarán todos los cursos con información.	4.0

COL14AW	**Como** usuario **Quiero** ver las asignaturas disponibles de mi carrera y curso seleccionado **Para poder** ver los apuntes sobre esta selección	a) Se mostrarán todas las asignaturas con información	4.0
COL15AW	**Como** usuario **Quiero** ver los apuntes disponibles de mi selección **Para poder** hacer uso de ellos	a) Se ofrecerá la información introducida por el colaborador. b) Se dispondrá del link donde esté el archivo. c) No se almacenará en el servidor el documento	6.0
COL16AW	**Como** usuario **Quiero** ver mis colaboraciones de apuntes **Para poder** borrarlos	a) Se mostrarán todas las colaboraciones de apuntes. b) Tendrá la opción de borrarla.	8.0
COL17AW	**Como** usuario **Quiero** borrar mis apuntes **Para poder** deshacer mis colaboraciones	a) Se mostrará un modal de confirmación.	2.0

5. Seguridad

ID	Historia de usuario	Criterio de aceptación	Puntos
SEG1SW	**Como** desarrollador **Quiero** filtrar el acceso a los servicios ofrecidos **Para poder** disponer de una lógica de negocio segura y robusta	a) Cuando se quiera acceder a ciertas peticiones te obliguen a autenticarte para ver tus permisos de autorización.	22.0
SEG2SW	**Como** desarrollador **Quiero** controlar la autorización del acceso a los servicios **Para poder** disponer de una lógica de negocio segura y robusta	a) Que cumpla los requisitos del sistema en cuanto al acceso de cada perfil de usuario.	14.0

Para una vista rápida del Product Backlog obtendré una tabla resumen de las historias de usuarios anteriores, así como de los puntos de historias asignados. Para ello me apoyaré en su agrupación y sumaré sus puntos de historia.

Servicio Web	Aplicación Web	Nº de historias de usuario	Nº de puntos de historia
Acceso	**-**	4	19.0
-	**Acceso**	4	27.0
-	**Interfaz**	3	40.0
Arquitectura	**-**	2	18.0
Colaboraciones	**-**	4	30.0
-	**Colaboraciones**	13	95.0
-	**Seguridad**	2	36.0
Total		32	265

Tabla resumen del Product Backlog

2.4.2. Planificación de los Sprint

En Scrum los proyectos avanzan en unas series de "Sprint". La duración de cada sprint depende del equipo de desarrollo pero, están en el rango de 2 a 4 semanas, nunca excederán el mes de trabajo. En cada sprint de Scrum no hay cambio, los cambios se hacen al final de cada sprint y será cuando me reúna con el tutor (a no ser que surja alguna duda relacionada con el uso de las tecnologías que utilice y no me deje avanzar).

Tomaremos como duración del sprint 15 días (dos semanas) distribuida a comodidad del equipo, si fuese una empresa debería establecerse dentro del horario laboral. Según la experiencia y del tiempo que dispone el equipo de desarrollo sabemos que puede mantener una velocidad de 30 a 40 puntos de historia por sprint. Con esto podremos saber el número de sprint que necesitamos:

$$\frac{265\ puntos\ de\ historia}{40\ puntos\ de\ historia/sprint} = 6{,}625 \text{sprints} \Rightarrow 7 \text{ sprints}$$

Lo que supondrá tres meses y medio dejando tiempo para limar detalles y organizar la memoria del TFG que estamos leyendo. A continuación, deberemos estructurar cada punto de historia en el sprint correspondiente teniendo en cuenta las precedencias y el tiempo que disponemos. Se lo resumiré en la siguiente tabla:

Sprint	Historias de usuario	Nº de historias	Puntos de historia
Sprint 1	AC1SW, AC2SW, AC3SW, AC4SW, INT1AW, AC5AW	6	40
Sprint 2	INT2AW, INT3AW, AC6AW,ARQ1SW	4	35.5
Sprint 3	AC7AW, AC8AW, ARQ2SW, COL1SW, COL2SW	5	42.5
Sprint 4	COL5AW, COL6AW, COL7AW, COL8AW	4	36
Sprint 5	COL3SW, COL4SW, COL9AW, COL10AW, COL11AW	5	39
Sprint 6	COL12AW, COL13AW, COL14AW, COL15AW, COL16AW, COL17AW	6	36
Sprint 7	SEG1SW, SEG2SW	2	36

Tabla de planificación por sprint

Como vemos el Sprint 2 y el Sprint 3 se desvía un poco de la media de los 40 puntos de historia. Pero no supone mucho problema dado que, cuando se termine el sprint 2 podemos avanzar un poco del siguiente. Además, si éste no fuese suficiente, disponemos de un margen un poco amplio para jugar con los tiempos.

2.4.3. Seguimiento de la planificación

Una herramienta para el seguimiento de la planificación Scrum son los diagramas Burn Down también conocidos como diagramas de quemado, son una representación del trabajo por hacer del proyecto en el tiempo. Generalmente representaremos el trabajo que queda por hacer conocido como trabajo remanente o backlog en el eje vertical y el tiempo del que disponemos en el eje horizontal.

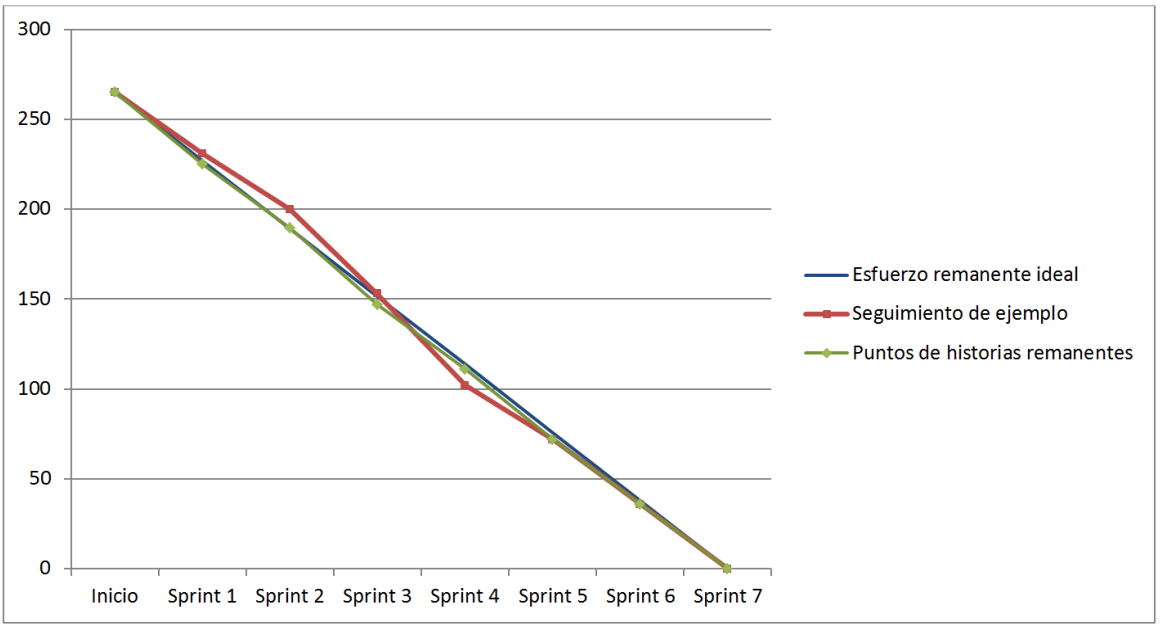


Fig. 3: Diagrama Burn Down Chart o de quemado

Como podemos ver, el esfuerzo ideal (línea azul) nos muestra la velocidad media que debemos llevar para completar nuestro proyecto en el tiempo indicado. Por otro, lado la línea verde son los puntos de historia que hemos acordado por sprint y como se observa difiere de forma ínfima al esfuerzo ideal.

Como línea roja puse un ejemplo de lo que suele ocurrir, aunque sólo es un ejemplo, en él muestro como al principio del proyecto no se lleva la velocidad adecuada ya sea por motivo de adaptación a las tecnologías utilizada o porque el equipo no se conocen lo suficiente para trabajar aún bien en equipo. Pero a partir del Sprint 3 ya se obtiene un ritmo bueno, tanto que en el Sprint 4 se ha avanzado en el trabajo del siguiente sprint… luego se estabiliza la velocidad y se mantiene acorde a la planificación.

Aunque no es mi caso, pues solo hay una persona trabajando en el proyecto, una buena manera de realizar el seguimiento es utilizando una tabla de tareas o Taskboard. El cual se puede comparar como el espejo del trabajo diario del equipo, es un elemento visible físicamente por todo el mundo y muestra la información relevante relacionada con el proyecto y/o el equipo. Por comodidad se suele utilizar una pizarra física accesible para todos, donde cada miembro del equipo tiene y se responsabiliza de una fila de historias de usuario que en Scrum será Sprint backlog. El propietario de la fila será el responsable de trabajar e ir organizando las historias de usuario de su Sprint backlog aunque, también se puede dar un backlog único donde trabaje todo el equipo.

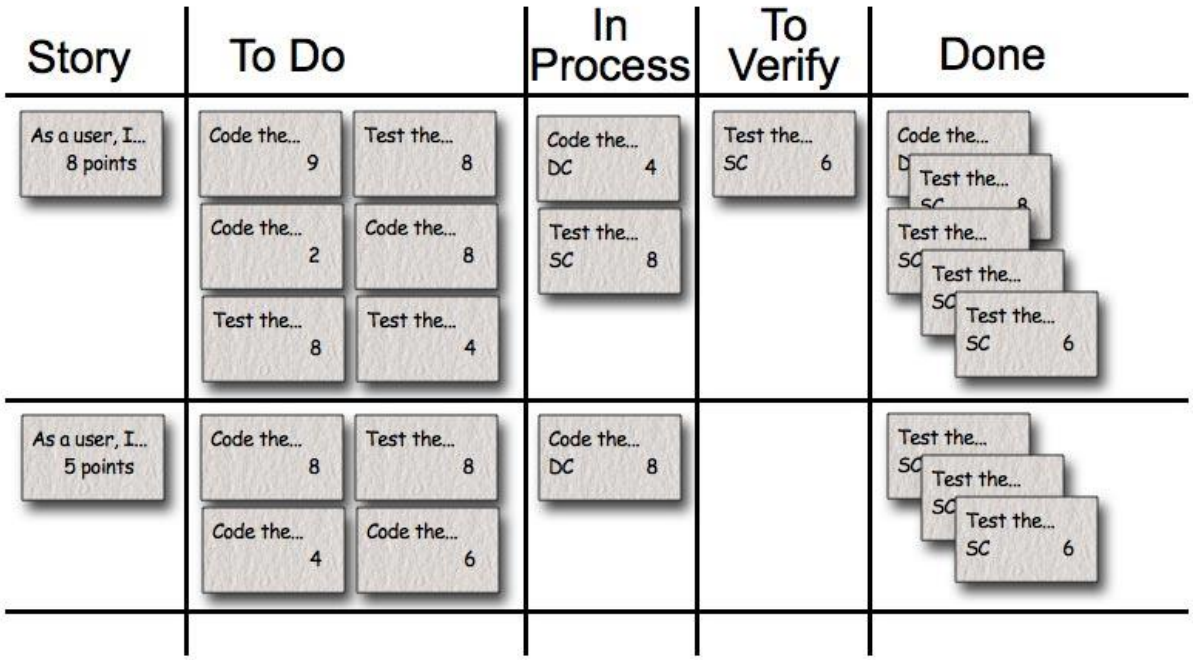


Fig. 4: Imagen ejemplo de Taskboard(Fuente: https://goo.gl/sGii4B, Abril 2014)

2.5.- Estudio de viabilidad

Uno de los puntos más importante en un proyecto software es la estimación de lo que costará su desarrollo. Es una de las actividades de planificación que reviste especial importancia, ya que una de las características que debe tener un producto de software es que su costo sea adecuado, de lo contrario no sería viable y puede fracasar.

Existen técnicas para la estimación de costos, pero para ello se requiere experiencia, acceso a una buena información histórica y coraje para confiar en medidas cuantitativas, cuando todo lo que existe son datos cualitativos. Nosotros hemos realizado una planificación bien detallada que, gracias a la metodología ágil, nos ayudará a seguir sin mucha dificultad. Los datos iniciales obtenidos para la planificación han sido que, el tamaño del equipo de desarrollo es de un único miembro y el tiempo límite de entrega se ha establecido en tres meses y medio (correspondiente a los 7 sprints x 15 días/sprint).

Pasamos al cálculo del costo, tendremos que considerar ciertos elementos que vamos a agrupar en: gastos hardware, gastos de licencia de software, servicios externos, gastos de transporte y gastos de personal.

2.5.1. Gastos hardware

Para el desarrollo del proyecto se va a utilizar dos ordenadores: un ordenador de sobremesa valorado en 780€ en el año 2010 y un ordenador portátil de 835.90€ en el 2012. Para el ordenador portátil, dado su uso, se estimó una vida útil de unos 4 años. Sin embargo la vida útil cumplió en el 2014 y se le hizo unas mejoras que costó 235.68€, prolongando su vida otros dos años más. Se incluirá también un monitor del 2010 valorado en 299.99€ con vida útil calculada en 6 años. Por último, teclado de 22.5€ y ratón 15.20€ del 2014 que se calcula dos años de vida útil.

Se resume a continuación al completo los gastos de hardware con el precio de amortización que supondrá su uso durante este proyecto:

Recurso	Precio	Años de utilidad	Coste mes	Total 3.5meses
Ordenador sobremesa: - Intel Core(TM)2 Duo - RAM 2GB - Almacenamiento 512 GB - NVIDIA GeForce GTX 260 Mejoras: - Intel Core i5 4460 - KIT 8GB RAM DDR3	En 2014 quedó amortizado 780€ Queda: 235.68	Son 6 de los cuales aún quedan 2 años por amortizar	9.80€/mes	34.3€
Ordenador portátil: - Intel Core i7 2º generación - RAM 4GB - Almacenamiento 512 GB - NVIDIA GeForce 610M	835.90€	4	17.42€/mes	60.97€
Monitor-Televisor LG 22” LED	299.99€	6	4.17€/mes	14.6€
Teclado multifunción HP	22.5€	2	0.94€/mes	3.29€
Ratón óptico Logitech	15.20€	2	0.64€/mes	2.24€
Total	**-**	**-**	**32.97€/mes**	**115.4€**

Tabla de gastos hardware

2.5.2. Gastos de licencia de software

La mayoría de licencias son gratuitas al ser software libre o universitario (acuerdo software con Microsoft), las incluyo pero con un coste de cero euros:

Licencia	Precio	Coste
Sistema operativo Window 7 Professional	Universitario 0€	0.00€
NetBeans IDE 8.0.2	Software libre	0.00€
Gestor de base de datos XAMPP	Software libre	0.00€
Repositorio GitHub para control de versiones	Plan Micro 5.27€/mes	18.45€
Frameworks utilizados mencionados anteriormente	Software libre	0.00€
Microsoft office	Universitario 0€	0.00€
Avanced REST Client	Aplicación gratuita	0.00€
Licencia Visual Paradigm	Universitario 0€	0.00€
Total	**-**	**18.45€**

Tabla de gastos de licencia de software

2.5.3. Servicios externos

Servicios contratados a terceros, ya sea gastos inmateriales (como Internet y electricidad) o servicios de auditorías, administración,… que en nuestro caso no hemos necesitado. Podemos incluir los siguientes servicios externos:

Servicio	Precio	Coste
ADSL Ocean's Network hasta 20Mb	36.3€/mes	127.05€
Endesa - Luz, uso aproximado de 320 horas	0.26€/hora	83.2€
Total	**-**	**210.25€**

Tabla de gastos por servicios externos

2.5.4. Gastos de transporte

Después de cada sprint se ha de utilizar un vehículo para reunirme con el tutor, el vehículo consume 4.5 litros cada 100 km, el precio medio del litro en estos meses ha estado a 1.22€/litro. La distancia a recorrer por cada reunión ha sido 52 km que, con las 7 reuniones, hacen un total de 364 km. Se resume a continuación:

Transporte	Distancia	Coste
4.5litros/100km *1.22€/litro = **5.49€/100km**	364 km	19.98€

Tabla de gastos de transportes

2.5.5. Gastos de personal

Para los gastos de personal hemos tomado como referencia la tabla salarial del convenio colectivo de Telefónica que es publicado en BOE. El enlace de referencia puede consultarse en [10], donde miraremos la categoría de Programador senior y Analista.

Categoría	Sueldo	Coste
Analista.	2555.29€/mes(½ mes)	1277.65€
Programador senior	2166.79€/mes(3 meses)	6500.37€
Total	**3.5 meses**	**7778.02€**

Tabla de gastos de personal

2.5.6. Gastos totales y viabilidad

El gasto total que supondrá nuestro proyecto será la suma de los gastos calculados en los puntos 2.5.X. anteriores, sumando un total de:

Gastos	Coste
Gastos de hardware	115.4€
Gastos de licencia software	18.45€
Servicios externos	210.25€
Gastos de transporte	19.98€
Gastos de personal	7778.02€
Total	**8142.1€**

Tabla de gastos totales

Para la viabilidad del proyecto buscaremos financiación pública pues, se trata de un proyecto libre, que no tendrá un precio de venta y no se desea uso de publicidad (o al menos para obtener ningún tipo de ingresos relacionados con ella).

No sería el primer caso, como ejemplo, tenemos el caso de la financiación por parte del gobierno alemán de desarrollos de GnuPG orientados a facilitar su uso para los usuarios no experimentados. Su idea era favorecer el uso de correo seguro entre sus ciudadanos.

Nuestra financiación será buscada por la Universidad de Jaén, pidiendo ayuda también a otras universidades andaluzas o incluso recurrir a la Junta de Andalucía.

Fig. 5: Ejemplo de campaña viral. (Fuente: http://goo.gl/sRiwft, 2014)

Se estima que inicialmente la aplicación se irá conociendo por campaña online, pero conforme avance las funcionalidades (no sólo las básicas proporcionadas en el prototipo) y aumente las aportaciones de los usuarios, se convertirá en una campaña viral por diferentes dominios y redes sociales.

2.6.- Modelado

En éste punto de la memoria con el análisis anterior, ya tenemos información suficiente para ver si el proyecto es viable y disponemos de la "idea" de solución. Con la aprobación del cliente, ahora comenzamos a analizar con más detalle la información y funciones del sistema para poder realizar un buen diseño.

Para una mejor comprensión del sistema a desarrollar, hemos modelamos mediante notación UML [11]. Podemos definir el modelado como la creación de modelos del sistema, con el fin de entender mejor el flujo de datos y control, el tratamiento funcional, el comportamiento operativo y el contenido de la información.

Ahora vamos a centrarnos en la creación de diagramas y modelar el análisis. Pero como las metodologías ágiles están orientadas a entregas y construcciones tempranas, cabe pensar que el modelado no es indicado si seguimos estas metodologías. El diseño y modelado en las metodologías ágiles se desarrolla iterativamente de forma indirecta, sin ser un proceso concreto, no obstante, existen ciertas ocasiones en la que son recomendables. Dean Leffingwell afirma en su libro sobre software ágil [12] que el modelado como los casos de

uso rara vez se usan en proyectos ágiles, aunque sí es muy recomendable su uso en proyectos que adquieren cierta envergadura.

Por lo tanto y para facilitar el aprendizaje y comprensión del proyecto en ésta memoria trabajaremos también con la burocracia que conlleva las metodologías tradicionales.

Los modelos necesarios para un análisis completo se describen en la Fig. 6, como podemos observar son Modelo del dominio (o Modelo Conceptual), Modelo de casos de uso (o Diagrama de Casos de Uso) y los escenarios principales de casos de uso.

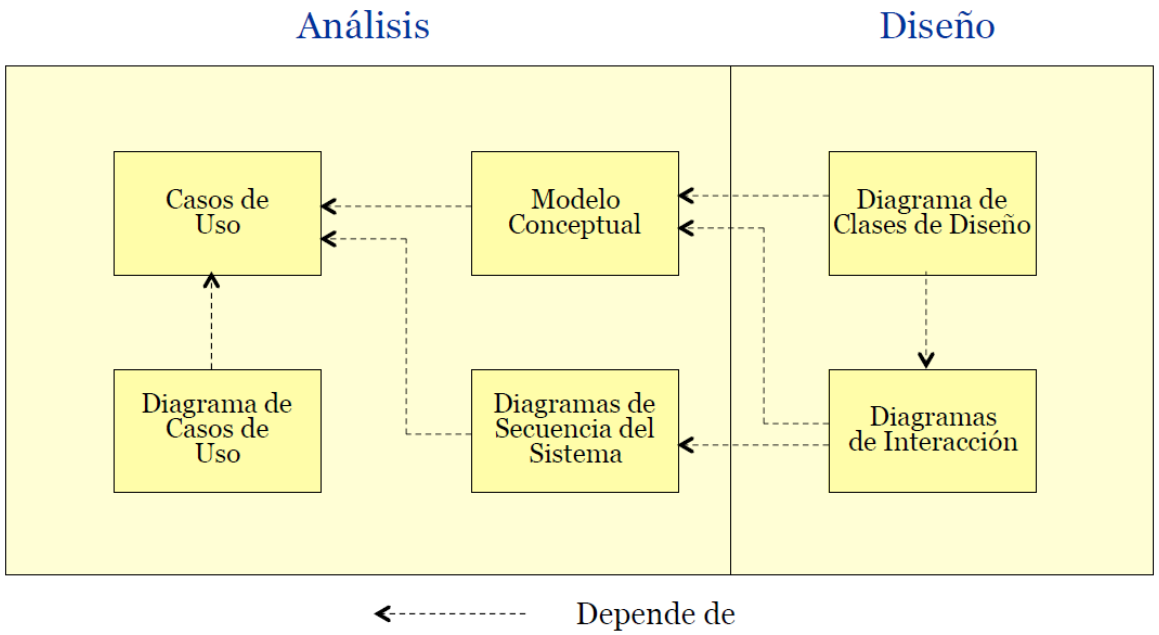


Fig. 6: Transición del modelo de análisis al de diseño

A continuación, en los siguientes apartados, vamos a desarrollar el modelado del sistema a crear utilizando la notación UML. El modelado del análisis consistirá en la obtención del modelo del dominio para obtener las entidades, atributos y relaciones del mundo real que rigen nuestro problema a resolver. A continuación, definiremos el diagrama frontera donde observaremos las relaciones entre los actores y los casos de uso de nuestro sistema. Por último, describiremos los casos de uso principales dando un mayor nivel de detalle, y para terminar, analizaremos la interfaz viendo las opciones que nos pueden guiar en su futuro diseño.

2.6.1.- Modelo del dominio

Un modelo de dominio es un modelo conceptual de todos los temas relacionados con un problema específico. En él se describen las distintas entidades, sus atributos, papeles y relaciones, además de las restricciones que rigen el dominio del problema.

Suele confundirse a primera vista con los diagramas de clase de diseño, pero a diferencia de éstos, definen conceptos del mundo real más que entidades software. También es cierto que suponen la base para el diseño, pero no ha de confundirse. Tras mostrar la imagen del

modelo, explicaré por encima las características básicas para la gente que no conozca la notación UML.

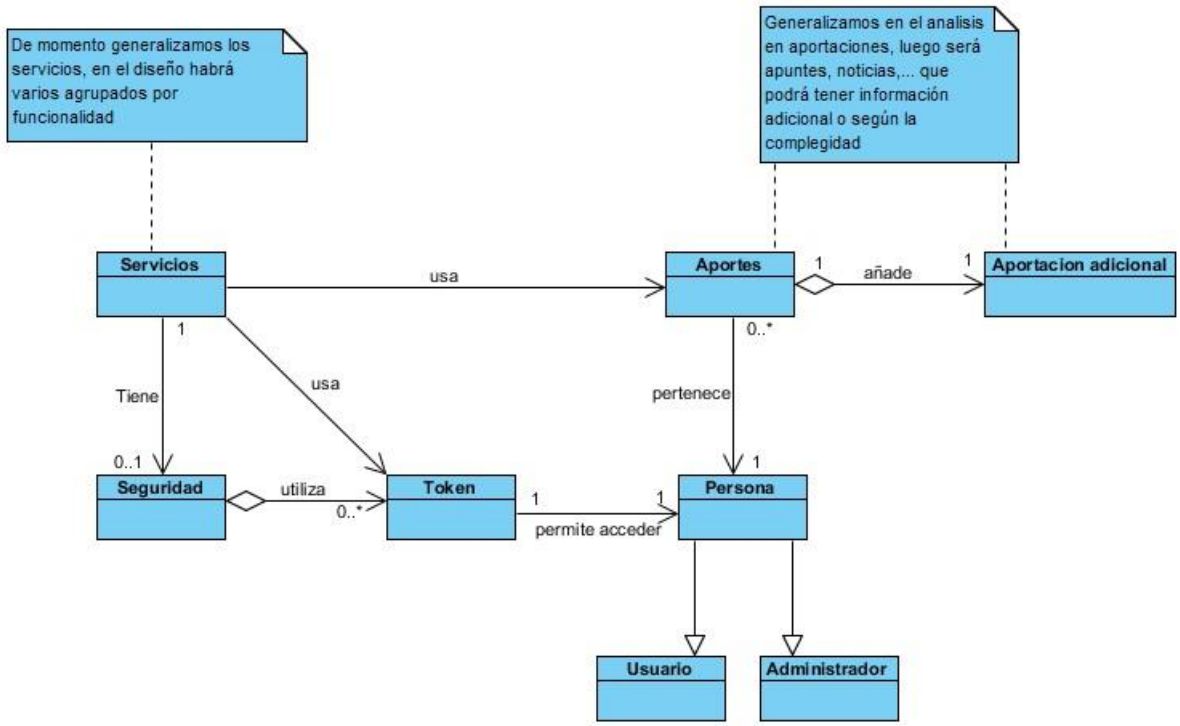


Fig. 7: Modelo del dominio

En el diagrama del modelo del dominio se puede ver los objetos que van a formar parte en el proyecto, y aun no conociendo la terminología UML se puede entender fácilmente. Dispondremos de unos “Servicios” aún no detallados para realizar las consultas que demandemos, pero estos servicios tendrán una “Seguridad” que utilizará los “Token” para controlarla. Además éstos “Token” corresponderá a una “Persona” que puede ser “Usuario” o “Administrador”.

Los “Servicios” usarán los “Aportes” (aún no definidos tampoco) como pueden ser noticias o apuntes. A éstos últimos, por ejemplo, se le deberá añadir una “Aportacion adicional” como datos no comunes como pueden ser asignatura, curso,... Otra información importante es que un aporte pertenece a una “Persona”.

2.6.2.- Diagrama de casos de uso

En este apartado, mostraremos el diagrama frontera del proyecto que estamos desarrollando. No confundir con los casos de uso, los cuales, son más detallados que los diagramas y se definen en el siguiente apartado. Solamente se resumen algunas de las relaciones entre casos de uso, actores y los sistemas.

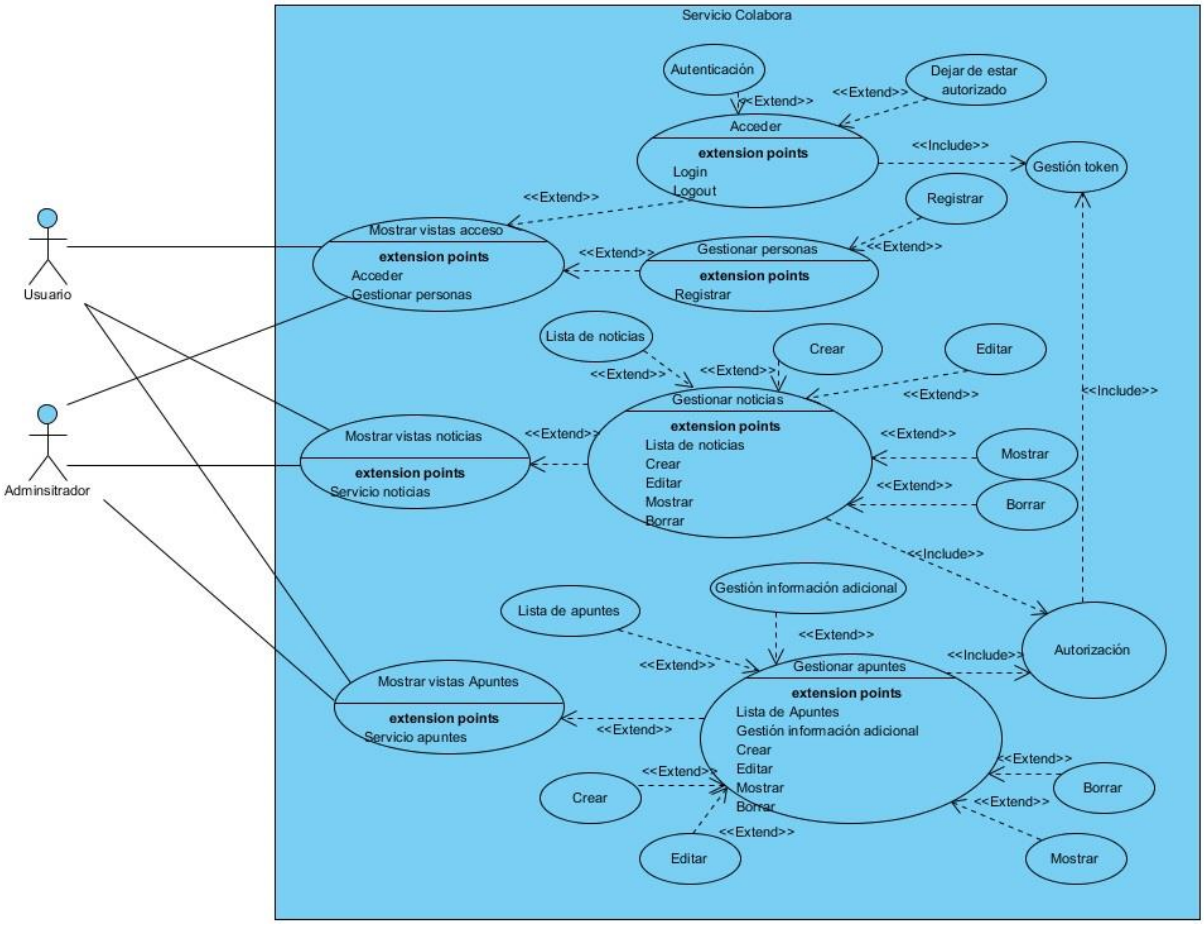


Fig. 8: Diagrama Frontera

Como podemos ver sólo describe sin entrar en detalle los requisitos funcionales del sistema, representando las relaciones entre los requisitos y los usuarios. Para comunicarnos con el sistema disponemos de tres casos de usos que nos muestran las vistas correspondientes.

La primera “Mostrar vistas acceso” puede extender del caso de uso “Acceder” o “Gestionar personas” según la función que vaya a realizar: registrar un usuario en el primer caso o autenticarse y dejar de estar autenticado en el segundo. Destacando que el caso de uso “Acceso” incluye “Gestión token” que gestionará el token que dará acceso al usuario a ciertas funcionalidades del sistema.

El resto de casos de uso que se comunican con los actores en el caso fronteras son “Mostrar vista noticias” y “Mostrar vistas apuntes”, que extienden de “Gestionar noticias” y “Gestionar apuntes” correspondientemente. Ambos incluyen un caso de uso denotado como “Autorización”, que a su vez incluye “Gestión token” que cumplirán la función de la

seguridad de la aplicación. Seguridad que consistirá en comprobar que el usuario está autorizado a realizar esa función.

En cuanto a “Gestionar noticias” y “Gestionar apuntes” tienen la posibilidad de listar su información, crearla, editarla, mostrarla y borrarla. Una diferencia entre estos dos casos de uso son que los apuntes dispone de información adicional y por lo tanto “Gestionar apuntes” extenderá de “Gestión de información adicional”.

2.6.3.- Escenarios principales de casos de uso

Lo realmente útil de los casos de uso no son los diagramas, pero son necesarios cuando empiezas a tener un número considerable de casos de uso como nuestro proyecto. Pues, cuando tenemos un buen número de casos de uso, no resultan nada fácil situarlos y relacionarlos. Lo realmente útil, es el documento que describe el caso de uso. En este documento se explica la forma de interactuar entre el sistema y el usuario.

Ahora se trata de concretar qué hacen los casos de uso. Aquí hay que ser bastante sistemáticos. El trabajo que realizaremos ahora redundará en una mejora de los siguientes pasos hasta la implementación.

En los siguientes apartados se detalla la documentación de los casos de usos principales, que viene a estar relacionado con las historias de usuario de nuestra metodología ágil al describir la funcionalidad de nuestro proyecto. Solo trataremos los casos de uso más representativos, como son la operación de registro, loguearse y hacer logout en la aplicación, la autenticación. También trataremos la operación CRUD (creación, mostrar, modificar y borrar) y el listado de forma general pues, son operaciones comunes tanto en noticia como en apunte.

2.6.3.1. Casos de uso de la operación de registro

Nombre:	Registrarse
Autor:	Manuel J. Castro
Fecha:	09/03/2015

Descripción:
Registrarse mediante la aplicación web.

Actores:
Usuario sin registrar.

Precondiciones:
- El usuario no está identificado.

Flujo Normal:

1. El actor pulsa la opción de registrarse.
2. Se muestra un formulario con los datos.
3. El actor rellena el formulario.
4. El sistema comprueba la validez de los campos.
5. El sistema realiza la petición de registro.
6. Se muestra un mensaje: “Se ha registrado correctamente”.

Flujo Alternativo:

4. Si los campos no son correcto, el sistema muestra el error y vuelve al punto 3.
6. Si la petición no se realiza correctamente, muestra el error.

Poscondiciones:
El usuario ha sido registrado en el sistema con permiso de estudiante.

2.6.3.2. Caso de uso de la operación de logueo

Nombre:	Loguearse
Autor:	Manuel J. Castro
Fecha:	09/03/2015

Descripción:
Loguearse en la aplicación para tener acceso.

Actores:
Usuario registrado.

Precondiciones:
- El usuario no está identificado.

Flujo Normal:

1. Se muestra un formulario con los datos (es la vista por defecto al sistema sin identificación).
2. El actor rellena el formulario y lo envía.
3. El sistema comprueba el usuario y contraseña.
4. El sistema realiza la petición de acceso creando el token.
5. Se le muestra al actor la vista por defecto para usuarios identificados.

Flujo Alternativo:

3. Si se produce un error en la comprobación, el sistema muestra el mensaje de error.

Poscondiciones:
Se dispone del token válido para el acceso, el cual se guarda también en la BBDD.

2.6.3.3. Caso de uso de la operación de logout

Nombre:	Logout
Autor:	Manuel J. Castro
Fecha:	09/03/2015

Descripción:
Hacer logout en la aplicación para terminar la sesión.

Actores:
Usuario logueado.

Precondiciones:
- El usuario está identificado.
- La sesión es válida (no ha caducado).

Flujo Normal:

1. El actor pulsa la opción de logout.
2. El sistema realiza la petición.
3. El sistema muestra al actor la vista por defecto para usuarios no identificados (login).

Flujo Alternativo:

2. Si se produce algún error, muestra un mensaje con el error correspondiente.

Poscondiciones:
Tras este proceso, el usuario no estará identificado.

2.6.3.4. Caso de uso de la operación seguridad

En ciertas peticiones saltará el filtro para comprobar que estás autorizado, a continuación veamos cómo sería su caso de uso.

Nombre:	Operaciones de seguridad
Autor:	Manuel J. Castro
Fecha:	09/03/2015
Descripción: Se realiza las operaciones de seguridad.	
Actores: El sistema.	
Precondiciones: Se dispondrá de un token de acceso.	
Flujo Normal: 1. Se detiene temporalmente la petición para comprobar si es permitida. 2. Se obtiene el usuario correspondiente al token. 3. Se mira su rol y se comprueba si está autorizado. 4. Se continúa con la petición.	
Flujo Alternativo: 3. Si no tiene permiso para esta petición se le devuelve el mensaje de que no está autorizado.	
Poscondiciones: Sólo podrán realizar dicha petición los usuarios con permiso, los cuales están definidos en los requisitos.	

2.6.3.5. Caso de uso de la operación CRUD

Se describe el caso de uso de las operaciones CRUD tanto de noticias como de apuntes, pues será similar en ambos casos, para ellos sustituimos noticia o apunte por aportación. El caso de uso de la seguridad controlará el permiso si es administrador o usuario normal.

Nombre:	Operaciones de Creación
Autor:	Manuel J. Castro
Fecha:	09/03/2015
Descripción: Se realiza la creación de una aportación.	
Actores: Usuario logueado.	
Precondiciones: - El usuario está identificado. - La sesión es válida (no ha caducado).	
Flujo Normal: 1. El actor pulsa que quiere realizar una nueva aportación. 2. El sistema muestra el formulario correspondiente a dicha aportación. 3. El actor rellena el formulario y lo envía. 4. El sistema comprueba la validez de los datos si fuese necesario. 5. El sistema almacena la información en la BBDD. 6. Se muestra al actor la vista correspondiente de su creación.	
Flujo Alternativo: 4. Si la información rellenada no es correcta, se muestra el problema correspondiente. 5. Si se produce un error en algún punto del proceso se devuelve el error.	
Poscondiciones: El aporte estará disponible para cualquier consulta tras la finalización de la petición.	

Nombre:	Operaciones de Lectura
Autor:	Manuel J. Castro
Fecha:	09/03/2015

Descripción:
Se pide al sistema que te muestre un aporte a través de su id.

Actores:
Usuario logueado.

Precondiciones:
- El usuario está identificado.
- La sesión es válida (no ha caducado).

Flujo Normal:

1. El actor pide ver una aportación concreta**.**
2. El sistema recibe la petición y es pasado por el filtro de seguridad.
3. Se realiza la petición a la BBDD**.**
4. Se muestra la vista con objeto obtenido en la BBDD.

Flujo Alternativo:

2. Si no está autorizado, se muestra el mensaje correspondiente.
3. Si se produce un error en algún punto del proceso se devuelve dicho error.

Poscondiciones:
Se muestra la información que se ha pedido y las opciones de dicha información si la hubiese.

Nombre:	Operaciones de Actualización
Autor:	Manuel J. Castro
Fecha:	09/03/2015

Descripción:
Se modifica una aportación dada.

Actores:
Usuario logueado.

Precondiciones:
- El usuario está identificado.
- La sesión es válida (no ha caducado).

Flujo Normal:

1. El actor pide modificar una aportación concreta**.**
2. El sistema muestra la información en su formulario correspondiente.
3. El actor modifica la información que necesita cambiar**.**
4. El actor envía el formulario.
5. El sistema comprueba la validez de los datos si fuese necesario.
6. El sistema realiza la operación de actualización en la BBDD.
7. El sistema muestra una vista donde se pueda comprobar que los datos han sido modificados.

Flujo Alternativo:

2. Si se produce un error en la obtención del aporte a modificar, se notificará el error.
5 y 6. Si la información rellenada no es correcta o si se produce un error en algún punto del proceso de edición, se muestra el problema correspondiente.

Poscondiciones:
El aporte modificado estará disponible para cualquier consulta tras la finalización de la petición.

Nombre:	Operaciones de Borrado
Autor:	Manuel J. Castro
Fecha:	09/03/2015

Descripción:
Se elimina una aportación dada.

Actores:
Usuario logueado.

Precondiciones:
- El usuario está identificado.
- La sesión es válida (no ha caducado).

Flujo Normal:

1. El actor pide borrar una aportación concreta**.**
2. El sistema muestra un modal o una advertencia de confirmación.
3. El actor confirma el borrado.
4. El sistema recibe la petición y es pasado por el filtro de seguridad.
5. Se realiza la eliminación en la BBDD.
6. El sistema muestra una vista donde se pueda comprobar que el aporte ha sido borrado.

Flujo Alternativo:

4. Si no está autorizado, devuelve el error correspondiente.
5. Si se produce un error en algún punto del proceso se devuelve el error.

Poscondiciones:
El aporte ya no estará disponible en ninguna consulta tras la finalización de la petición.

2.6.3.6. Caso de uso de la operación listado

Del mismo modo que hicimos en el apartado anterior, entenderemos aportación como noticia o apunte puesto que la operación será similar.

Nombre:	Operaciones de Borrado
Autor:	Manuel J. Castro
Fecha:	10/03/2015
Descripción: Se listan las aportaciones existentes en la BBDD.	
Actores: Usuario logueado.	
Precondiciones: - El usuario está identificado. - La sesión es válida (no ha caducado).	
Flujo Normal: 1. El actor desea ver un listado concreto y realiza dicha petición**.** 2. El sistema recibe la petición y es pasado por el filtro de seguridad. 3. Se realiza la consulta a la BBDD. 4. Se muestra el listado en la vista correspondiente.	
Flujo Alternativo: 2. Si no está autorizado, devuelve el error correspondiente. 4. Si se produce un error en algún punto del proceso se devuelve el error.	
Poscondiciones: Se obtiene la vista con el listado deseado.	

2.7.- Análisis de la interfaz

No nos extenderemos mucho hablando sobre el análisis de la interfaz, puesto que ya se detalla mejor en el diseño de la interfaz más adelante. Pero es importante, a la vez de interesante mencionar ciertos aspectos a analizar antes del diseño, dado que la interfaz es la superficie de contacto entre las persona y el sistema que desarrollamos. También es importante en nuestro caso porque estamos buscando facilitar la colaboración entre estudiantes universitarios. Y sin una buena Interacción Persona Ordenador, esto podría perjudicarnos después de tanto trabajo en lógica de negocio.

Según McIntyre[6], "la interfaz constituye entre el 47% y el 60% de las líneas de código" [14] y además, existe una tendencia de un diseño gráfico cada vez más exigente. Todo esto ha supuesto que hayan ido surgiendo diferentes frameworks y templates (gratuitos y de pago) para facilitar la vida al programador.

Un framework muy extendido es *Bootstrap,* surgió inicialmente por Twitter que permite crear interfaces web usando CSS y JavaScript. Como fuerte es su capacidad de adaptar la interfaz al tamaño del dispositivo en el que se vaya a visualizar.

Para ayudarme a satisfacer la experiencia del usuario, recientemente han surgido como tendencia unas series de principios de diseño. Los principios lo proporciona Google en su *Material Design* [15], son 9 principios pero, los principales son:

1. **El material es la metáfora:** Una metáfora material es la teoría unificadora de un espacio racionalizado y un sistema de movimiento. Nuestro material está basado en la realidad táctil, inspirado en nuestro estudio del papel y la tinta, pero abierto a la imaginación y la magia.
2. **Color, la superficie, y la iconografía destacan las acciones:** La acción del usuario es la esencia del diseño de la experiencia. Las acciones principales son puntos de inflexión que transforman el diseño del conjunto. Se enfatiza en la funcionalidad y proporciona puntos de interés para el usuario.
3. **Movimiento proporciona significado:** El movimiento es significativo y apropiado, sirve para centrar la atención y mantener la continuidad. La regeneración es sutil pero clara. Las transiciones son eficientes pero coherentes.

[6] Jean McIntyre: Es un analista programador en Interacción Persona Ordenador (IPO) de Philadelphia.

3.- Diseño

En el **análisis** indicamos **qué** debe hacer el sistema, con el análisis podemos entender la organización, los objetivos a cumplir y los requisitos.

El siguiente paso es comenzar con el **diseño**, donde especificamos el **cómo** debe hacerse, determinaremos la estructura y funcionamiento que tendrá nuestro software a partir de las herramientas obtenidas en el análisis. El diseño es un proceso iterativo, al principio se representa con un grado alto de abstracción. Y a medida que avanzan las iteraciones, con el refinamiento sucesivo, conduce a la representación del diseño con un menor nivel de abstracción.

Uno de los primeros pasos para comenzar el diseño, es obtener el diagrama de Entidad-Relación en el que se puede observar las entidades y las relaciones necesarias para la persistencia de la BBDD. Después desarrollaremos los diagramas de clases de diseño, definiendo cómo organizamos el código. A continuación, los diagramas de interacción como el diagrama de secuencia, que nos ayudará a entender cómo funciona e interacciona los elementos que contiene el sistema.

Además, recordemos que estamos trabajando con una metodología Scrum que sigue una metodología ágil y no una tradicional en cascada, donde los pasos de análisis, diseño e implementación son secuenciales. En nuestro caso se trata de pasos incrementales, divididos en diferentes iteraciones que denominamos sprint. En cada sprint se analiza, diseña e implementa los casos de usos pertenecientes a las historias de usuario que incluye en la iteración correspondiente.

Sin embargo, de igual modo que en el análisis, realizaremos un diseño completo como aconsejaba Leffingwell [12] en el modelado de grande proyectos software. Y como trabajamos con una metodología ágil, si fuese necesario, de una iteración a otra, podré cambiar el diseño sin suponer un gran coste de tiempo. Gracias a que la metodología ágil mueve el análisis y el diseño a la implementación.

Fig. 9: Coste del cambio en la vida de un proyecto software (Fuente: http://goo.gl/Q1CJxN)

A continuación, por lo tanto, mostraré el diseño obtenido y en caso de haber realizado algún cambio en el diseño de un sprint a otro, éste será reflejado junto a su justificación.

Nos falta comentar que, también aplicaremos los patrones de diseños de los que ya hablamos en el análisis preliminar: el patrón MVC y el patrón DAO.

3.1.- Diagrama Entidad-Relación

El siguiente diagrama de entidad-relación (también conocido por sus siglas en inglés, E-R) es una herramienta que permite representar las entidades más relevantes de un sistema informático, así como sus interrelaciones y las propiedades que contienen. Es decir, utiliza el mundo real para representar una serie de objetos, que llamamos entidades, los cuales tienen atributos que lo definen y pueden existir relaciones entre ellos.

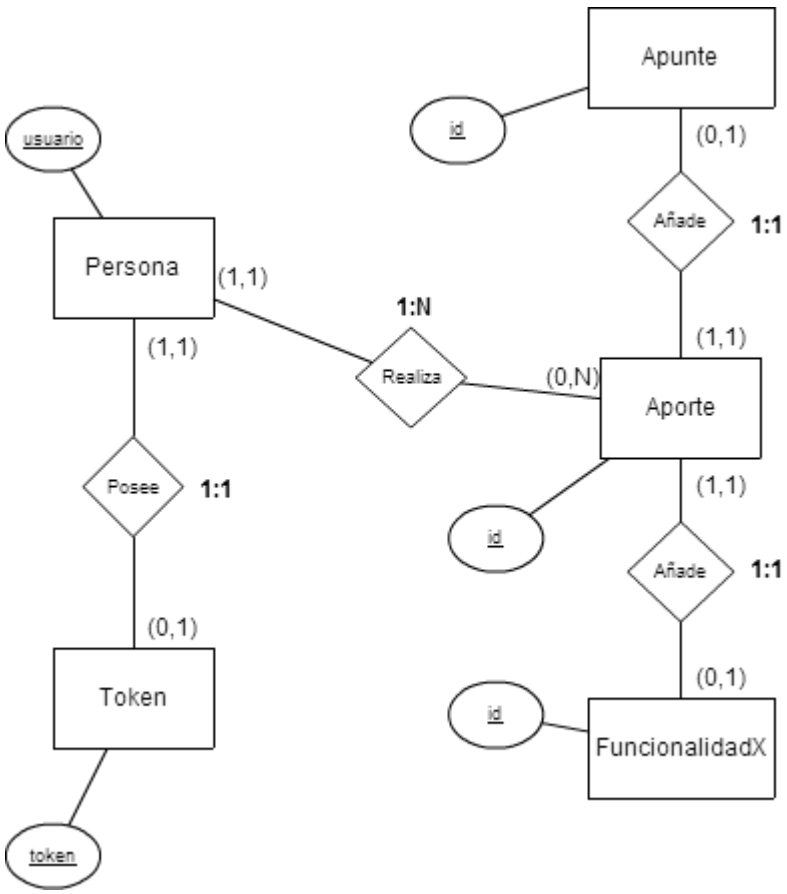


Fig. 10: Diagrama Entidad-Relación

El elemento central sería *Aporte*, que puede añadir cierta funcionalidadX, como es el caso de los atributos específicos de *Apunte* o una funcionalidadY como es compartir piso. O incluso ser un *Aporte* que no añade ninguna funcionalidad específica, siendo el caso de *Noticia*.

Un *Aporte* es realizado por una única *Persona*, pudiendo una *Persona* realizar desde ningún aporte hasta N, definiendo N como muchos *Aportes*. Además, en cuanto a *Personas,* para acceder a los aportes como ya sabemos debe poseer un *Token* que lo autorice, siendo éste único e intransferible. También puede no poseer ningún *Token* porque no esté autenticado en la aplicación.

Las claves, definen cada una de las entidades inequívocamente de una colección a la cual pertenece y suelen ser un subconjunto del conjunto de atributos en una colección de entidades. Son llamadas comúnmente claves primarias y son diferenciadas del resto de entidades al estar subrayadas. En nuestro diagrama decidimos mostrar solo los atributos que definen las claves por simplicidad al lector y por ello todas están subrayadas.

Las claves son el nick de usuario para *Persona*, la cadena de caracteres que define el token para *Token* y un identificador que generamos definido como id para el resto de entidades.

En teoría de bases de datos relacionales, es necesario proporcionar una base de datos que cumpla unos criterios que determine que el grado de vulnerabilidad es aceptable. El criterio lo proporcionan las **formas normales** (FN), debiendo de cumplir hasta la tercera forma normal (3FN). Nuestras tablas cumplen la tercera forma normal y por definición cumple las formas normales inferiores. Haremos un resumen de las formas normales:

- La primera forma normal (1FN): Una tabla se encuentra en primera forma normal si impide que un atributo de una tupla pueda tomar más de un valor.
- La segunda forma normal (2FN): Si cumple la FN anterior y toda la clave principal hacen dependiente el resto de atributos, si hay atributos que dependen sólo de una parte de la clave, éstos formarán otra tabla.
- La tercera forma normal (3FN): Si cumple la FN anterior y no hay atributos que dependen funcionalmente de atributos que no son claves.

El diagrama de Entidad-Relación constituye un elemento importante para obtener las entidades y relaciones necesarias para la persistencia de nuestra base de datos relacional. Pero nuestro diagrama se obtendrá automáticamente a partir de un **mapeador objeto-relacional** que nos creará una base de datos orientada a objeto virtual, sobre la base de datos relacional. De esta manera se construirá el esquema de la base de datos a partir de las relaciones definidas en el modelo del dominio. Además como veremos más adelante, se comparará en la implementación (en el punto 4.2.1.) el modelo generado y el diagrama anterior figura 10, y que se encuentran en tercera forma normal para evitar inconsistencia y duplicidades.

3.2.- Diagrama de Clases

En ésta fase nos ayudaremos de la información anterior para determinar las clases que modelan nuestro diseño y completar el conjunto de atributos de cada clase. Aunque mostramos el diagrama completo, el diseño ha sido un proceso iterativo, donde han surgido nuevas clases de diseño, se han ido añadiendo las operaciones conocidas como métodos y diseñando las relaciones entre las clases.

Recordemos nuevamente que estamos trabajando con SCRUM, una metodología ágil la cual mueve el diseño hacia la implementación con entregas más temprana, diseñando para el futuro próximo en diferentes iteraciones llamadas sprint. Y como podremos leer al final de este subapartado, hemos tenido que realizar un cambio de diseño, junto a ella se describe la motivación y justificación que nos han llevado al cambio.

En la propuesta a solución se describió que nuestro proyecto iba a consistir en un servicio web REST y su aplicación web donde el cliente utilizará el servicio. **Dividiremos los diagramas** en servicio web y aplicación cliente, de igual modo como también se hizo al obtener las historias de usuario en la planificación.

En el diagrama de **clases del servicio web** (fig. 11) hemos situado las clases en sus paquetes correspondiente, teniendo la funcionalidad otorgada que marcan los patrones MVC y DAO. Para facilitar la visualización he dividido el paquete controlador donde están los servicios en dos paquetes (marcados por el color rosa) pero en la práctica es el mismo paquete.

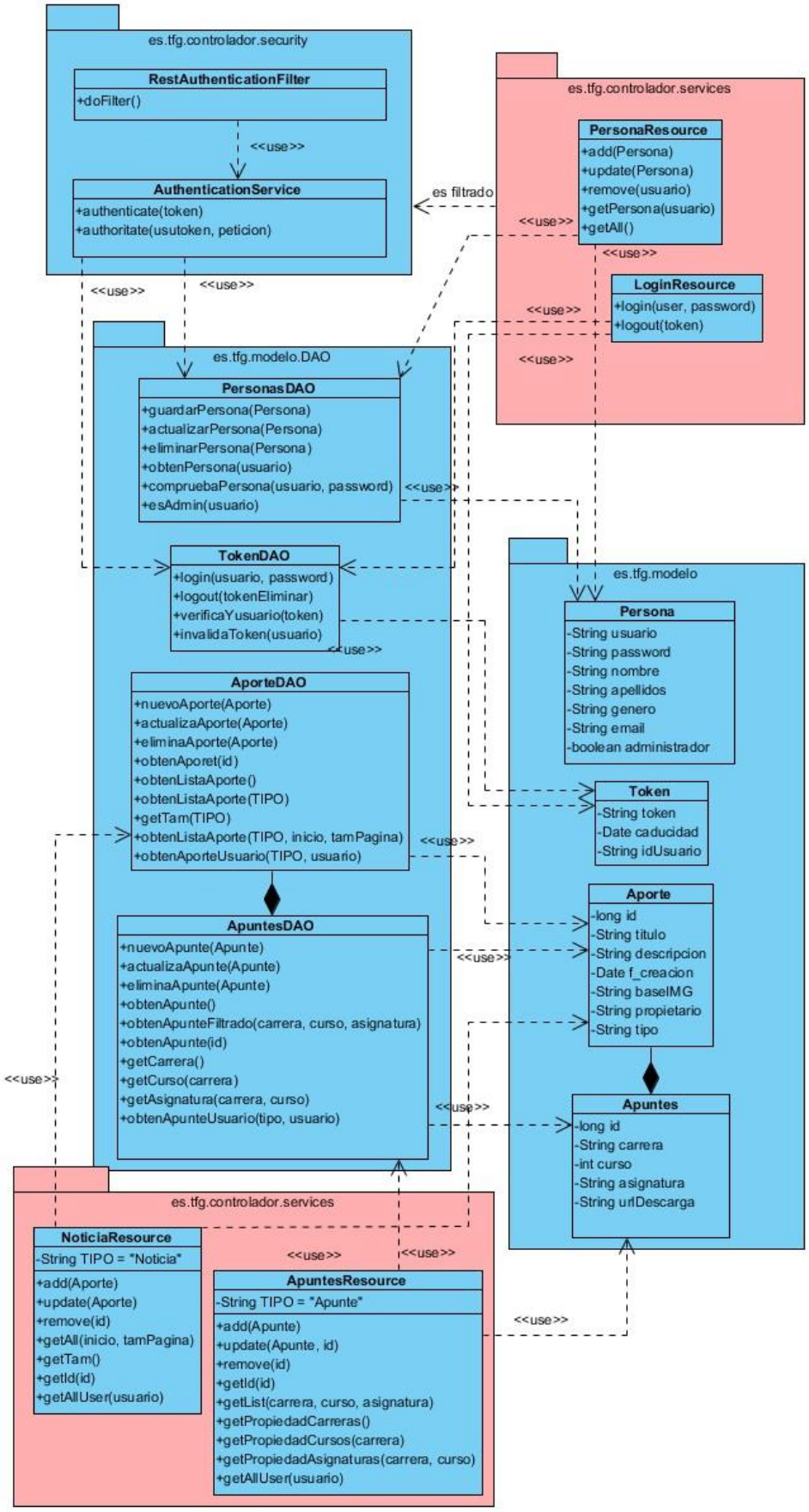


Fig. 11: Diagrama de clases del servicio web

En cuanto a las decisiones tomadas, durante el diseño podemos destacar que hemos eliminado la herencia de “Persona” como se mostraba en el modelo del dominio, esta decisión fue tomada dado que las clases que usaban la herencia no disponían de una diferenciación clara, solo de la propiedad de ser o no administrador. Por lo tanto se usa un booleano para mantener esta única diferenciación.

Optamos por agrupar los datos comunes de una colaboración (noticias, apuntes,...) en una clase llamada “Aporte”, diferenciaremos una de otra por el atributo tipo. Así si deseamos crear un aporte simple como es el caso de noticias, sólo tendríamos que crear el servicio con su seguridad, sin necesidad de crear la clase y el DAO correspondiente, solo tendríamos que usar en nuestro servicio el ya creado como “Aporte”.

Además, a la hora de tener una nueva funcionalidad compleja, podemos usar la estructura Aporte como base. Creando así la clase con la funcionalidad y propiedades que necesite como es el caso de Apunte, y mediante una relación de composición ya tenemos la base de nuestra funcionalidad común montada.

También se puede observar que divididos las clases en diferentes paquetes para encasillar cada clase donde le corresponde en el uso de los patrones MVC y DAO, es importante mantener la arquitectura bien organizada.

Podemos decir que el diseño del servicio web mostrado en la figura 11, se ha podido mantener durante los sprints 1, 2, 3, 4, 5 y 7 donde se ha requerido, sin necesidad de realizar alguna modificación. Es decir, el diseño inicial obtenido se ha mantenido durante todo el desarrollo.

En el diagrama de **clases de la aplicación cliente** (fig. 12) como podemos observar, tiene bastante similitudes en las operaciones al diagrama de clases del servicio web y es normal porque cumplen las mismas funcionalidades pero de diferente modo. El servicio ofrece la lógica de negocio mientras que la aplicación usa ésta lógica de negocio para visualizarla y trabajar con ella.

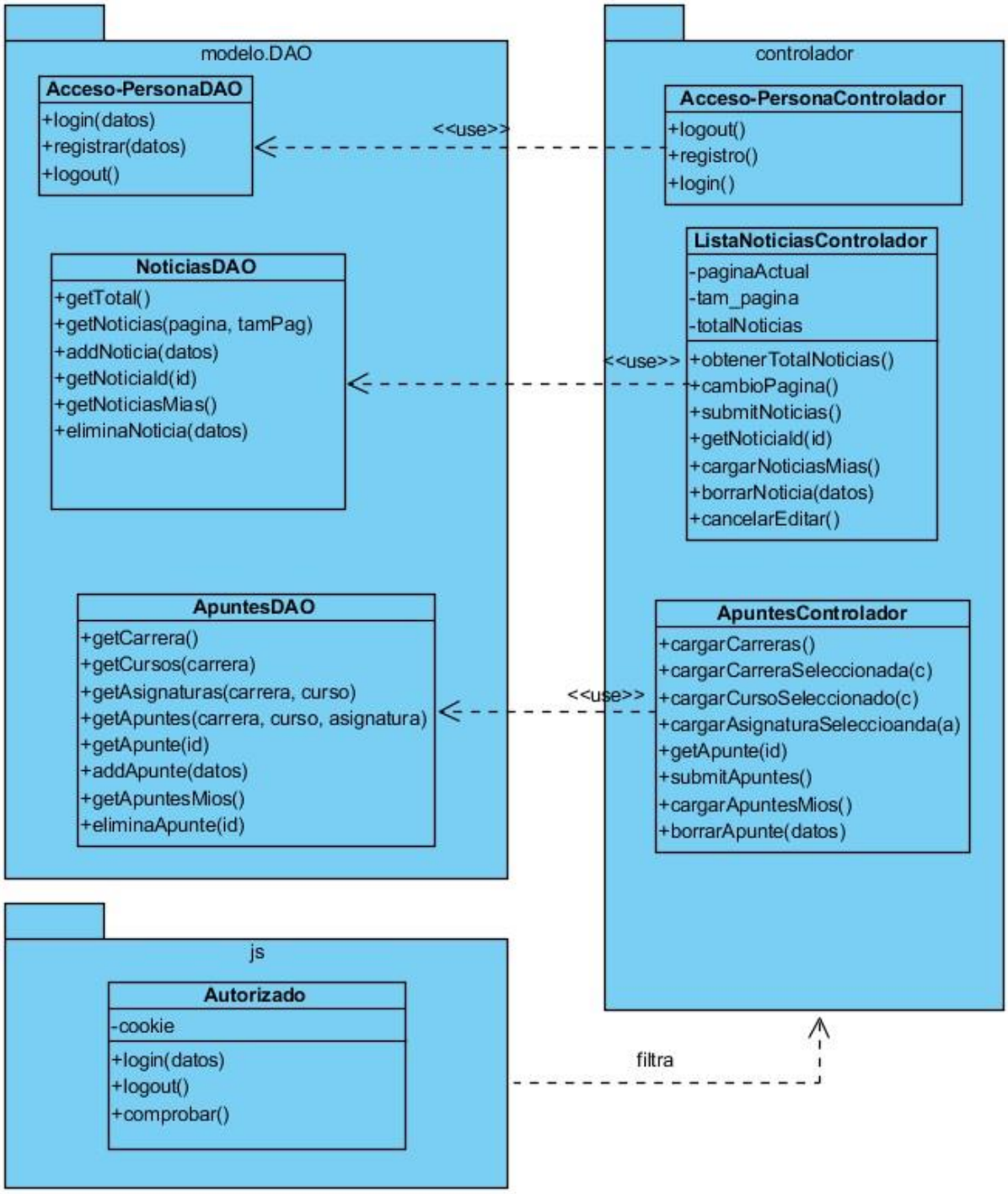


Fig. 12: Diagrama de clases de la aplicación cliente

Podemos ver también que se ha aplicado el patrón de diseño MVC y el patrón DAO, estando las clases dividida de forma similar al diagrama de clase del servicio web. Por un lado tenemos los controladores que a diferencia de los anteriores (Fig. 11), éstos sí trabajan sobre las vistas correspondientes por eso es una división más amplia. Es decir cada controlador dispondrá de una vista en la que trabaja, por ejemplo el controlador “NoticiaEditaControlador” trabajará sobre la vista “NoticiaEdita”, cuyas funciones son obtener la noticia a editar (getNoticiaId(id)), subir la modificación realizada (submit()) y cancelar la edición (cancelar()).

Los DAOs son usados por los controladores para comunicarse con el servicio, por lo que también podríamos verlos como objetos que se comunican con las APIs[7] del servicio.
Es importante mencionar que se dispone de un elemento que gestiona el acceso a la aplicación comprobando que estés autorizado mediante el uso de las cookies. Disponemos de una cookie que tendrá los datos del token siendo el identificador del token, fecha de caducidad y el usuario propietario, al loguearse en la aplicación se creará y en cada operación se comprobará la cookie y actualizará. Al salir de la aplicación, se borrará la cookie y el token en el servidor, impidiendo el acceso a la aplicación.

Sin embargo, el diseño mostrado inicialmente en la figura 12 no ha resultado ser el conveniente trabajando con *AngularJS*. *AngularJS* más que trabajar fácilmente con un MVC, trabaja como lo llaman muchos con un MVVM [13] (Model-View-View-Model). MVVM es llamado así por muchos porque son los datos más, lo que necesita para que estos datos se muestren de manera adecuada.

Terminando el primer sprint, cuando empezamos a trabajar en la aplicación web, al montar los primeros pasos del MVC nos dimos cuenta que crear el controlador por entidad no iba a ser tan natural como se esperábamos.

El controlador que llaman algunos en AngularJS, no son controladores en sí, más bien son funciones controladoras inicializadas en el *$scope*. El *$scope* se considera un objeto de *JavaScript* que puede extenderse creando propiedades que pueden ser datos y funciones, por lo tanto el controlador que considera algunos en *AngularJS* son funciones añadidas a este objeto que ayuda al control de la vista.

Por lo tanto, para obtener algo parecido a un MVC tuvimos que replantearnos el diseño para el sprint 2, obteniendo un controlador que inicializarse y manejara estas funciones controladoras del $scope. El resultado fue el siguiente:

[7] Una API(*Application Programming Interface*) representa la capacidad de comunicación entre componentes de software.

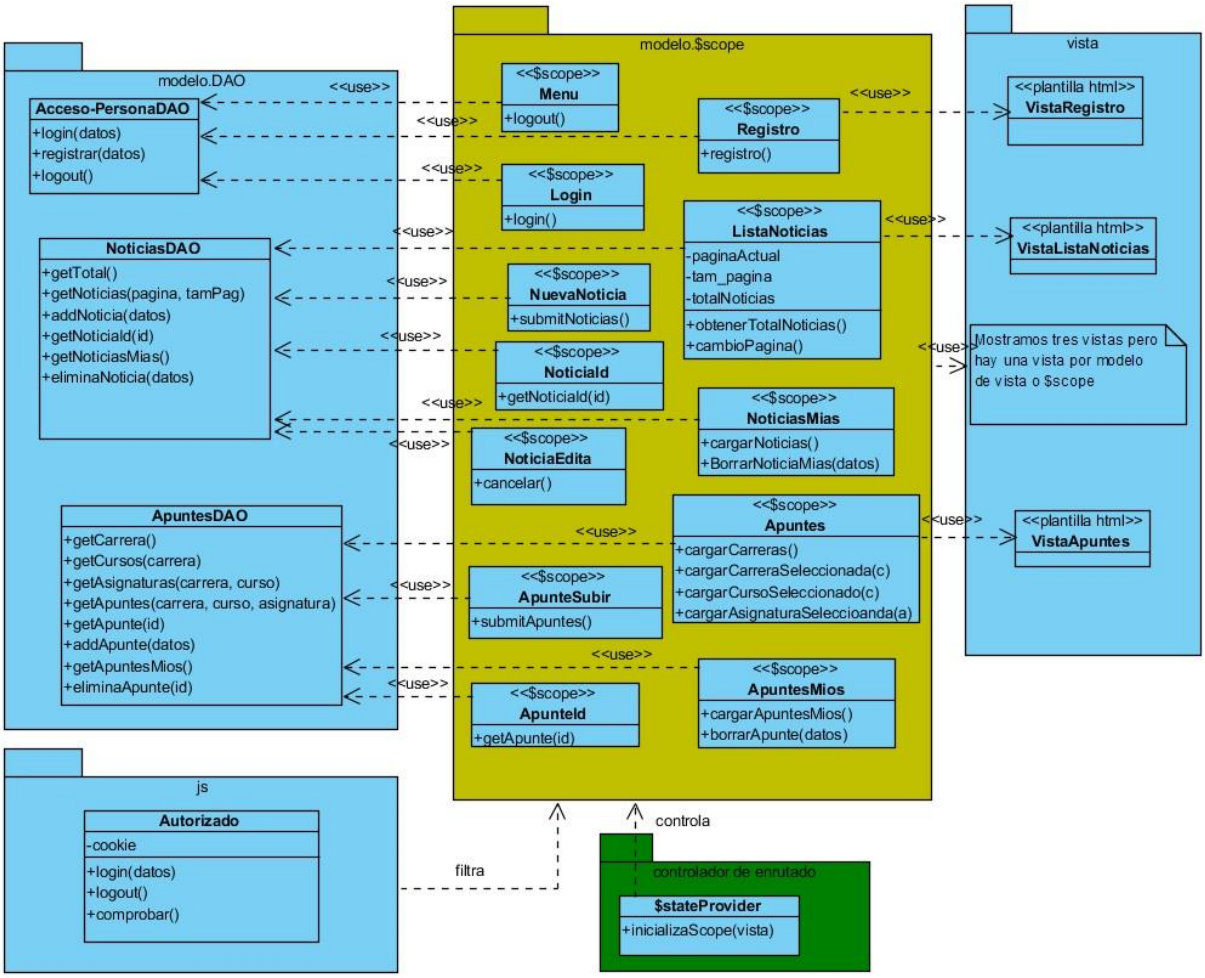


Fig. 12*: Diagrama de clases de la aplicación cliente en el sprint 2

Puede parecer un diseño complejo, pero en realidad su implementación es realmente simple. Mantenemos el diseño de los paquetes con el fondo azul, hemos cambiado el del fondo amarillo y ha surgido uno nuevo marcado de verde.

El problema estaba en el anterior controlador, no actuaba como controlador sino como funciones inicializadora y era otro elemento el encargado de la interacción con la vista, por lo tanto, los controladores anteriores por entidad se han dividido en funciones inicializadoras siendo mostradas en el paquete amarillo. El paquete amarillo ya no se llamara controlador, ahora la llamaremos modelos de vista, así tenemos nuestro MVVM con su DAO y sus modelos de vista o $scope en terminología Angular.

Como nuevo controlador, mostrado en el paquete verde, tenemos el proveedor de estado ($stateProvider) del enrutador, es el encargado de inicializar las funciones controladoras perteneciente al $scope de una vista.

Con esta forma de trabajar como lo hace *AngularJS,* tendremos una implementación directa de este diseño. Además, se adapta mejor al desarrollo ágil puesto que no son entidades tan amplias y se van implementando a razón de las historias de usuario.

3.3.- Diagramas de secuencia

En el siguiente apartado, veremos algunos diagramas de secuencias para definir cómo funciona el sistema y observar las interacciones entre los elementos que contiene. Para ello, haremos referencia a las interacciones más relevantes como pueden ser acceder a una noticia (que debe incluir loguearse para poder hacer las peticiones como listar y ver la noticia elegida) o borrar un apunte (comprobar que tiene permiso y realizar la petición). Hay muchas más operaciones para poder definir su interacción en diagrama de secuencia, pero con esto será suficiente para comprender el comportamiento del diagrama de clases junto con el modelo conceptual.

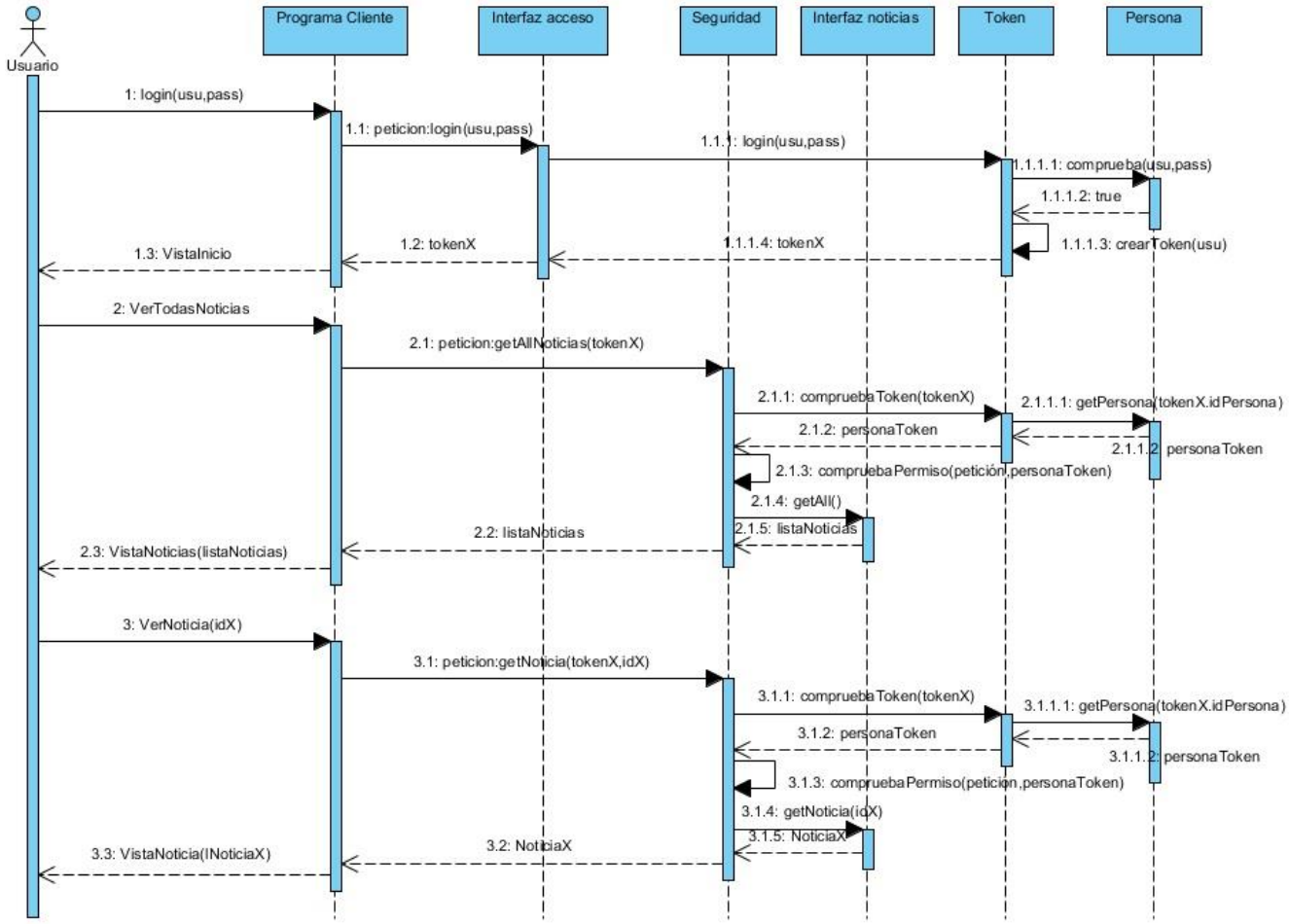


Fig. 13: Diagrama de secuencia para ver una noticiaX

Aquí ya se puede observar las comunicaciones y las clases necesarias para obtener una noticiaX antes de haberse logueado. En definitiva, son tres peticiones que hace el programa que se comunica con el cliente en diferentes vistas, siendo la petición de logueo, petición de ver las noticias disponibles, petición de ver una noticia de dicha lista.

En el diagrama de secuencia, se obtiene la interacción entre clases en un marco temporal definida por la notación numérica y la situación horizontal en el diagrama. De esta interacción sacamos los métodos que necesitaremos para la comunicación entre clase y los atributos necesarios a enviar.

Sigamos con el siguiente diagrama pero, para simplificarlo, el usuario ya está logueado con un token válido.

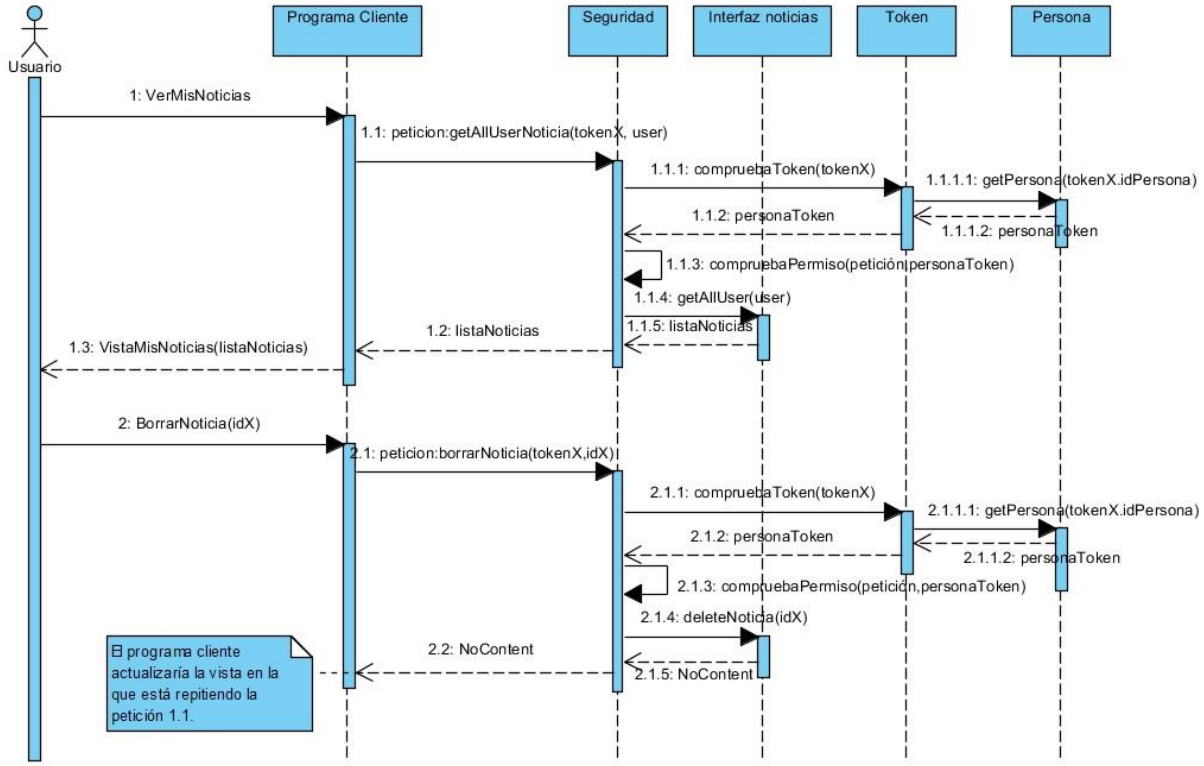


Fig. 14: Diagrama de secuencia para borrar una de tus noticias

Tenemos dos nuevas peticiones para un usuario ya logueado, el de listar sus noticias (noticias de la que es propietario) y borrar una de ellas, debería de existir una tercera petición pero sería la misma que la primera, en la que se mostraría la mismas noticias pero ya no estaría visualizada la eliminada.

Podríamos realizar y en algunos casos sería necesario, el diagrama de secuencia de cada una de las funcionalidades a proporcionar. Pero, al realizar una metodología ágil disponemos de las historias de usuario que nos guiarán en el proceso iterativo de construcción del problema.

3.3.- Diseño de la Interfaz

Seguiremos como definimos en los requisitos no funcionales los principios de adaptabilidad, notificaremos visualmente errores y campos no validados. Todo esto teniendo en cuenta la teoría básica de la interacción persona-ordenador y los principios de material design visto en el análisis de la interfaz.

Un aspecto importante también en el diseño gráfico es el uso y la elección de los colores, diferentes colores pueden generar diferentes sensaciones y emociones, incluso recuerdos. Para ello existen estudios y bases teóricas sobre su aplicación [16]. Es posible usar un **generador de paletas de colores** como éste [16] de *material design*. Para la elección de los colores de nuestro diseño, usaremos la base del colorido de la página de la Escuela

Politécnica de la Universidad de Jaén. Siendo los colores más similares en el generador de colores anterior: un azul normal y un verde azulado denominado “teal”.

Para el diseño de la interfaz, crearemos los wireframes de las vistas típicas que tendrá nuestra aplicación y nos ayudaremos de los storyboards para ver cómo se comunican unas vistas con otras, definiendo por la funcionalidad del caso de uso *ver una noticia sin estar logueado*.

El storyboard será realizado con ejemplos reales para así mostraros los resultados obtenidos y poder comparar entre los wireframe y las vistas reales.

3.3.2. Wireframes

Con los wireframe representamos el esqueleto del sitio web, en lo que se incluye los elementos de la interfaz y sistema de navegación. Se utilizan para ver la funcionalidad y definir la jerarquía de contenido por lo que habitualmente, por tiempo, carece de estilo y color. Nosotros le daremos algo de color para ayudar a un mejor diseño, al incorporar el color, mejoramos la visualización del diseño deseado.

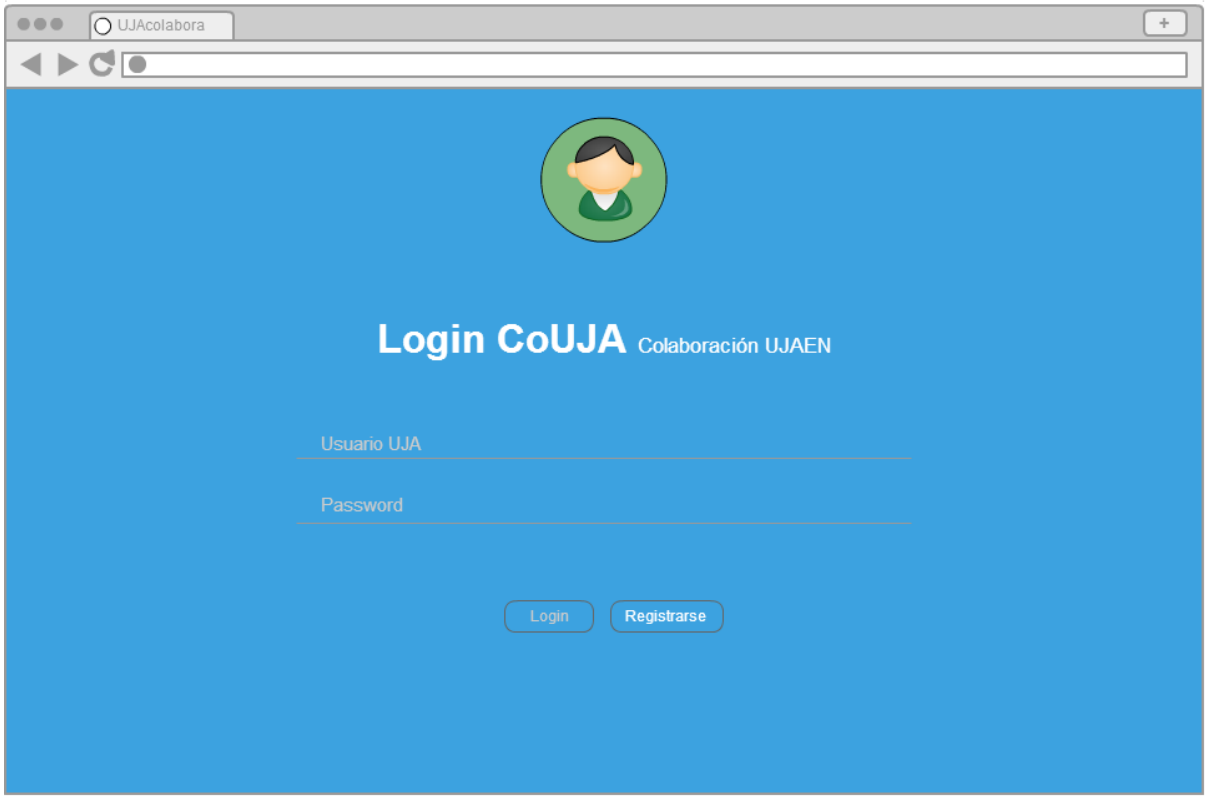


Fig. 15: Wireframe pantalla login.

Fig.16: Wireframe pantalla registro.

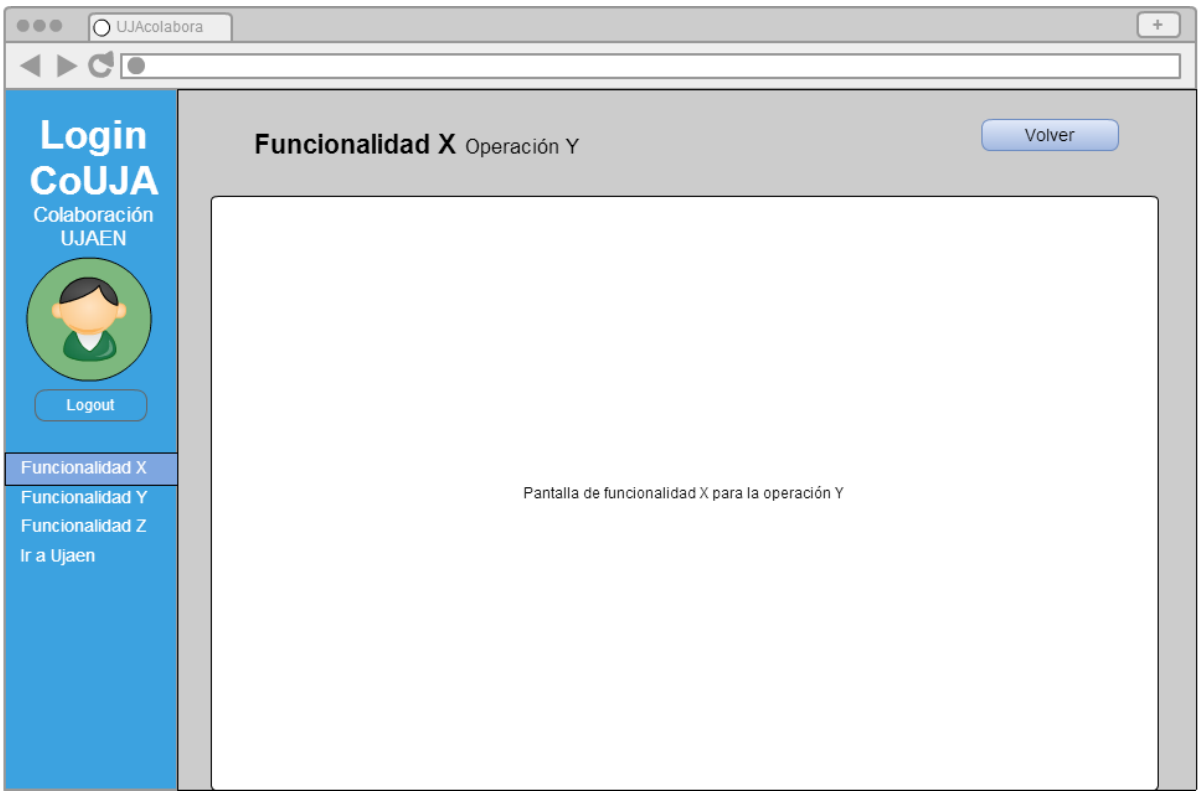


Fig.17: Wireframe pantalla general.

Hemos generalizado la mayoría de los wireframes en el último (Fig. 17), ya que la estructura que se muestra es la general para todos los casos. En la izquierda dispondremos del menú siempre visible, el cual dispondrá de las opciones de salir de la aplicación e ir a la página de la universidad, además de las funcionalidades que dispondremos como son “Noticias” y “Apuntes”. Al ir a una de las funcionalidades nos mostrará un inicio con la descripción de ella y las operaciones disponibles de la funcionalidad escogida como: crear un nuevo aporte, ver los aportes… Para tener el usuario constancia de donde se encuentra, arriba dispondrá visiblemente de esa información. Por último, para moverse de forma eficiente por la aplicación dispondrá de un botón en la esquina superior derecha que le llevará a la vista anterior.

Todas las vistas usarán los principios de adaptabilidad, en el caso de registro y login no supone mucha dificultad pues son simples formularios pero el caso general si y su resultado será el siguiente:

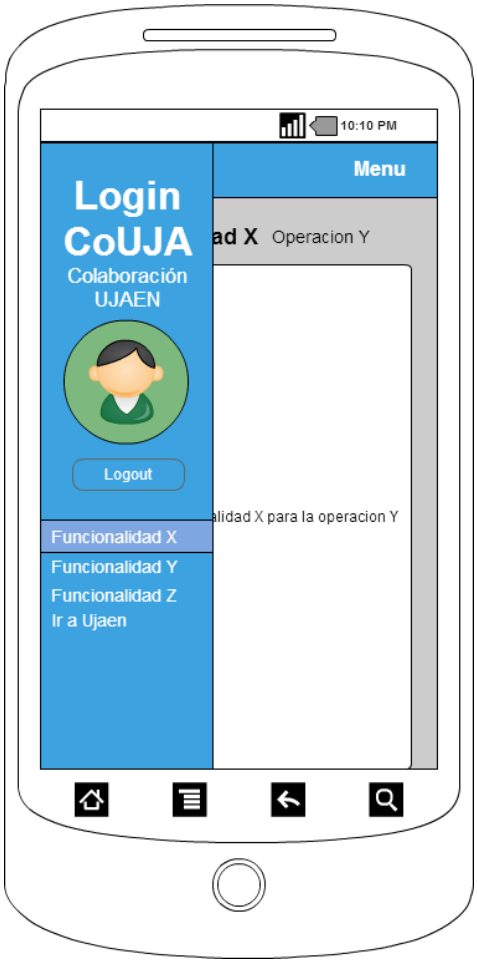
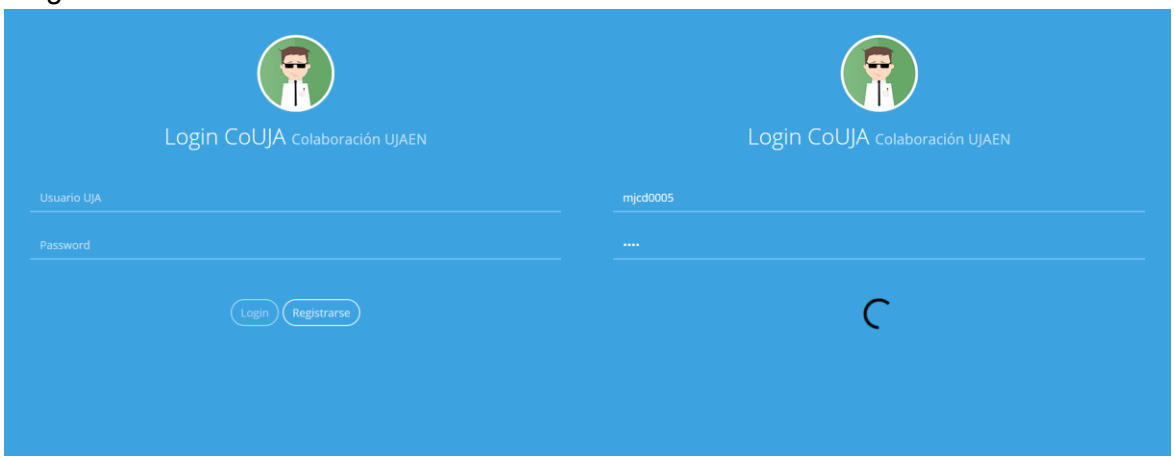


Fig.18: Wireframe pantalla general (adaptado).

3.3.2. Storyboard con ejemplos reales

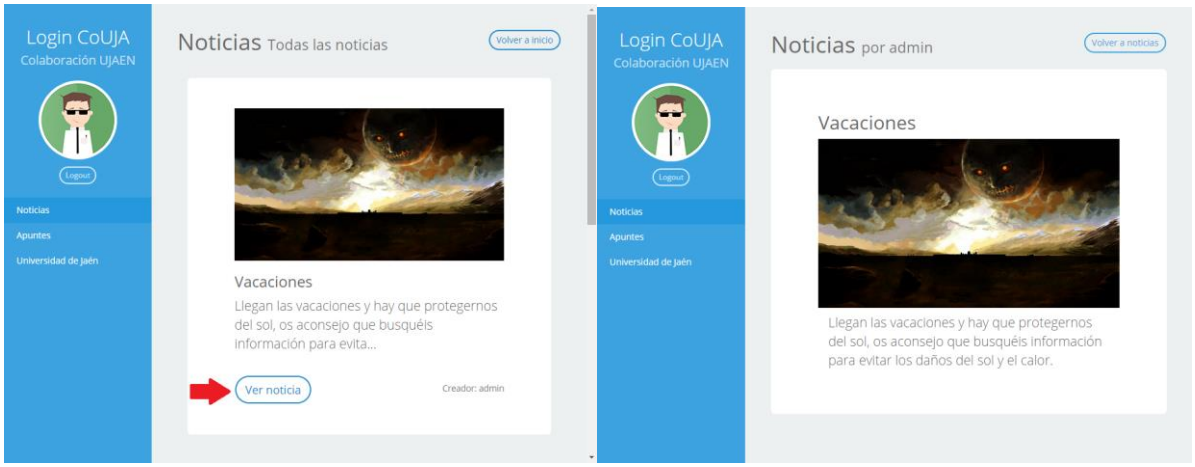
Como comentamos, definiremos el caso de uso ver una noticia desde la vista del login, para ello describimos los storyboards oportuno con ejemplos reales de la aplicación, y además también narraremos los pasos realizados.

Primero en la vista login debemos rellenar los campos del formulario, y pulsar el botón “Login”:



A continuación, nos aparece la pantalla general que comentamos en los wireframes, por defecto nos aparece en la primera funcionalidad que es “Noticias” (que casualmente es la que deseamos). Si quisiéramos otra funcionalidad deberíamos pulsar el botón del menú con la funcionalidad a elegir. Como se describió nos muestra como “facilidades” las operaciones que podemos realizar. La operación que buscamos es ver “Todas las noticias” para así, buscar y abrir la noticia que buscamos:

La siguiente vista nos lista las noticias (dos por página), en el título de la vista nos muestra en qué vista estamos, y en la esquina superior derecha un botón para volver a inicio. Seleccionamos la primera noticia por ejemplo, para ello pulsamos “Ver noticia”, completando así nuestro storyboard.



4.- Implementación

Llegamos al punto de la realización de nuestro proyecto, la implementación. En las metodologías tradicionales esta fase utiliza como punto de partida el modelo de la fase anterior, procediendo a programar o implementar los diseños especificados en el modelo de diseño.

Sin embargo estamos utilizando una metodología ágil y no es 100% necesario la fase de diseño completa anterior. En el desarrollo ágil basta con tener diseñado lo que se va a programar en el sprint actual, siendo el futuro cercano (2 semanas). Permitiendo así un cambio de diseño en los sprints siguientes, aun así nosotros hemos realizado un diseño completo usado como guía, el por qué ya lo vimos en el modelado (Punto 2.6.).

Por lo tanto, la implementación ha seguido la planificación ágil, diseñando cada sprint y luego implementando según las historias de usuario especificadas. Comentaremos que se ha implementado en cada sprint y si fuese necesario, las modificaciones obtenidas tras el sprint.

Pasamos a ver la arquitectura general, comentaremos en ella los elementos, cómo se relacionan y la función que cumplen. El siguiente paso es entrar en detalle de implementación, donde hablaremos en detalle sobre los mismos y describiremos los sprints en los que se han llevado a cabo.

4.1.- Arquitectura

Para el prototipo a implementar dispondremos de dos arquitecturas básicas con implementación “independientes”, el término independientes lo entrecomillado porque ambas arquitecturas trabajan juntas. Es decir dispondremos de:

- Arquitectura Cliente (Aplicación Web): Encargada de la interfaz y comunicación con el cliente, sólo dispone de la lógica de la aplicación, necesita del servicio web para obtener la lógica de negocio.
- Arquitectura Servicio (Servicio WEB): Contiene la lógica de negocio y ofrece unos servicios mediante, como veremos más adelante, el protocolo HTTP. Necesita de un sistema de comunicación con el cliente para ser usada, como es el caso de la aplicación web.

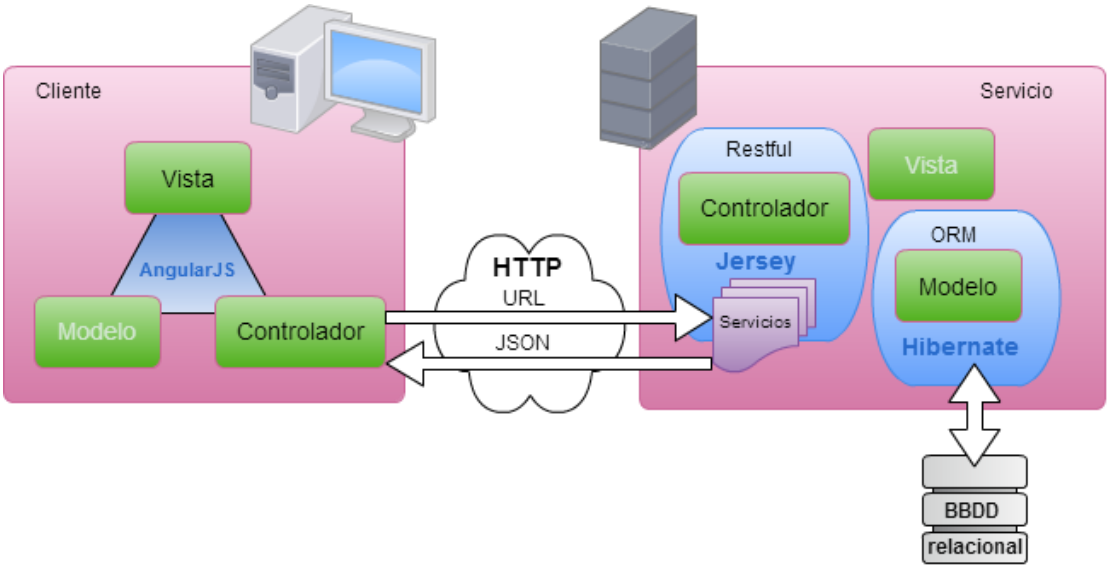


Fig. 19: Infraestructura general.

4.1.1. Arquitectura Servicio

A tener en cuenta principalmente en esta arquitectura, es como ofrecer el servicio web. Existen dos posibilidades ya mencionadas en el análisis, SOAP o REST. La tendencia actual es crear el servicio web con un framework REST, las que están sustituyendo rápidamente a los tradicionales SOAP como podemos ver en la fig. 20.

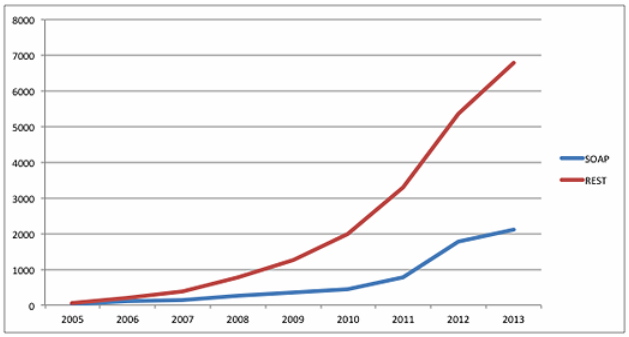


Fig. 20-Número de SOAP vs. REST.

Para el lado del servidor programaremos en el lenguaje orientado a objeto más usado durante toda la carrera, Java y por lo tanto usaremos un framework de Java. Nos queda discutir qué frameworks utilizar para crear el servicio, nuestras opciones estudiadas anteriormente eran *Spring Framework* y *Jersey*.

El debate de usar Spring o Jersey y otros frameworks como los de persistencia, acaba rápidamente si decimos que buscamos la sencillez. Al buscar la sencillez, usaremos los framework específico. Es decir, dado a que no necesitaremos gestión de transacciones complejas, caché y a que busco la sencillez, para así añadir nuevos servicios fácilmente utilizaré framework específicos para la creación de servicio Restful como es el caso de ***Jersey***. Recordamos que *Jersey* es un framework de código abierto de Java que proporciona soporte JAX-RS APIs.

Otro aspecto importante en esta arquitectura es la persistencia, hay infinidad de posibilidad y es un tema que puede llevar una memoria entera debatir sobre ella en su totalidad. Básicamente disponemos de dos grandes grupos, la tradicional de las bases de datos relacionales y las bases de datos orientados a objetos.

Pese a la nueva moda de usar sistemas de gestión de base de datos orientada a objetos, yo me decanto por las tradicionales bases de datos relacionales y la creación asequible y fiable de MySQL, siendo de código abierto de mayor aceptación mundial. Para poder integrarla en nuestro proyecto necesitaremos las siguientes herramientas:

- Usaremos el sistema de gestión de bases de datos MySQL con la plataforma ***XAMPP*** [18], para reducir la curva de aprendizaje de otra plataforma, la vimos en alguna asignatura ofrecida por la universidad y era ampliamente usada en ESIJA Informática S.L. donde realicé mis prácticas de empresa.
- Para establecer la conexión con nuestro proyecto en Java con la base de datos MySQL, debemos incluir ***mysql-connector-java*** que es un driver proporcionado por MySQL.
- Por últimos y no menos importante, debemos saber que sin utilizar ninguna herramienta para la creación y las peticiones con la base de datos relacional estando trabajando con Java, un lenguaje orientado a objeto, puede ser algo tedioso y propenso a errores. Por ello utilizo la herramienta ***Hibernate*** [19], un framework que me soluciona todos estos problemas mediante el Mapeo objeto-relacional (ORM) a partir de archivos declarativos (XML) o anotaciones en los beans de las entidades que permite establecer estas relaciones.

A comentar sobre el framework ***Hibernate,*** es que nos permitirá la creación automática de las tablas, hecho importante al no tener una base de dato fija. El número de tabla puede ir incrementando según la necesidad de los estudiantes. Además la comunicación con la base de datos gracias a Hibernate y utilizando el patrón DAO, no supondrá mucha dificultad.

4.1.2. Arquitectura Cliente

Es la arquitectura encargada de hacer posible la comunicación entre el cliente y el servicio donde está la lógica de negocio. Pero su función no es simplemente actuar de interfaz entre el cliente y el servicio, también dispondrá de la lógica de aplicación que se encargará de crear una interacción agradable y fácil al usuario.

Para nuestro caso desarrollaremos una aplicación web, que se abastece de un servicio REST permitiendo así también poder ampliar fácilmente, por ejemplo a Android o iOs y con un bajo consumo de datos en las peticiones. Me decanté por plataforma web porque casi todos los dispositivos disponen de navegadores y por lo tanto haremos un diseño adaptativo.

A estos programas, como es el caso de nuestra aplicación web, que hace posible la comunicación del servicio con el cliente se le suele denominar la programación del lado del cliente. Y surge la pregunta que todo programador se hace siempre ¿qué lenguaje usar para su creación?

Podríamos decantarnos por HTML5 + Javascript compitiendo con las aplicaciones nativas que, como es lógico, siempre será mejor en rendimiento y acceso en cada una de las funcionalidades de los dispositivos. Sin embargo, eligiendo HTML5 + Javascript abarcamos el gran abanico de dispositivos con navegador, pero surgen grandes dificultades a la hora de hacer grandes aplicaciones… los problemas de rendimiento por la sobrecarga de procesamiento en el servidor, además de los problemas ya mencionados en las aplicaciones en el lado del cliente. Para resolver estos problemas o dificultades surgen diferentes tecnologías y frameworks que nosotros usaremos.

Debemos destacar entre todas estas tecnologías a ***AngularJS*** [20] siendo de código abierto, realizado en Javascript. *AngularJS* contiene un conjunto de librerías útiles para el desarrollo de aplicaciones web y propone una serie de patrones de diseño para llevarlas a cabo.

Nosotros nos decantamos por utilizar *AngularJS*, no entraremos más en detalle de lo que ya hemos comentado, sólo mencionaré la justificación de su uso. Estamos diciendo de añadir nuevas demandas cuando la aplicación se extienda, de forma rápida y sencilla. Tenemos que tener además en cuenta, que la programación de aplicaciones web puede ser una verdadera tortura sin ningún framework o patrones. Por lo tanto, debemos encontrar un modo de disponer de patrones que nos facilite la programación, y para ello el framework *AngularJS* es la mejor opción además de la más usada como se puede ver en Fig. 2.

4.2.- Detalles sobre implementación

A continuación trataremos cuestiones concretas sobre los elementos mencionados de la arquitectura, comentando el sprint donde ha sido desarrollado, además de añadir los puntos de la implementación que deban ser destacados ya sean metodológicos o técnicos. Adicionalmente resumimos como el proyecto ha sido estructurado. Para el desarrollo del apartado volveremos a diferenciar entre la arquitectura servicio y la arquitectura cliente.

4.2.1. Implementación Servicio

Como puntos importantes en la implementación tenemos las librerías necesarias y los aspectos a tener en cuenta en la persistencia y el servicio. Sin olvidarnos también de los aspectos que tomados en la seguridad.

Desde la implementación del primer sprint hay que atender a los aspectos de la persistencia y aspectos de los servicios, y para ello, tenemos que añadir librerías de *Hibernate* junto con driver de conexión con MySQL y *Jersey*, mostradas en la siguiente figura:

Fig. 21: Librerías necesarias para la creación del servicio.

- Persistencia -

En la persistencia de datos optamos por realizar el mapeado mediante su definición del Mapeo objeto-relacional (ORM) a partir de archivos declarativos (XML). A continuación, mostramos unos ejemplos de mapeo de clases para comentar algunas decisiones tomadas sobre el diseño. Haremos todo lo posible para facilitar la comprensión de la implementación de *Hibernate*, explicando brevemente los puntos donde puedan surgir dudas. Para una mejor comprensión del framework aconsejamos seguir el curso [21].

Token.hbm.xml

Source History

```
1  <?xml version="1.0"?>
2  <!DOCTYPE hibernate-mapping PUBLIC
3          "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4          "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
5
6  <hibernate-mapping>
7      <class name="es.tfg.modelo.Token" table="TOKEN">
8          <id name="token" column="TOKEN"></id>
9          <property name="caducidad" type="timestamp" column="F_CADUCIDAD" />
10         <property name="idUsuario" type="string" column="IDUSUARIO" />
11     </class>
12 </hibernate-mapping>
```

Fig. 22: Fichero Token.hbm.xml.

En el sprint 1 tuvimos que crear la persistencia de “Persona” y “Token”, en ella tuvimos que tomar una decisión importante en tema de implementación, en la Fig.22 se puede ver que la entidad Token difiere del modelo entidad-relación de la Fig. 10, en el modelo disponemos de una relación uno a uno y nosotros hemos creado esta relación añadiendo el id del usuario como columna. La diferencia de haberla creado con una relación en Hibernate a como se muestra en el modelo es que de nuestro modo, al enviar el token en una petición, solo va el id del usuario y no el usuario completo como resultaría de la relación con *Hibernate*. La ventaja de esta decisión es doble, el envío del token es continuo y reduciendo el peso de cada envío se reduce el consumo de datos considerablemente, además evitamos enviar continuamente la información del usuario que tal vez pueda ser privada. En caso de necesitar la información de éste, disponemos del id para realizar la petición y si fuese privada se comprobaría si hay autorización.

De igual modo lo implementamos en el sprint 3 (con la historia de usuario COL1SW) para la relación Persona-Aporte, la relación se mantiene pero no es proporcionada por Hibernate.

En el sprint 2, dedicamos una historia de usuario para plantearnos una implementación que fuese con una arquitectura estructurada y organizada, haciendo posible que la funcionalidades fueran extensibles fácilmente. Pero creando la persistencia de apuntes en el sprint 6 tuvimos que tomar una decisión en la implementación que comentaremos tras observar las siguientes figuras.

```
<id column="ID" name="id">
  <generator class="identity"/>
</id>
```

Fig. 23: Generación automática de id.

Apunte.hbm.xml

Source History

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
3  <hibernate-mapping>
4      <class name="es.tfg.modelo.Apunte" table="Apuntes">
5          <id column="ID" name="id"></id>
6          <one-to-one name="datosGenerales" class="es.tfg.modelo.Aporte"
7                      cascade="all"></one-to-one>
8          <property column="CARRERA" name="carrera" type="string"/>
9          <property column="CURSO" name="curso" type="int"/>
10         <property column="ASIGNATURA" name="asignatura" type="string"/>
11         <property column="URLDESCARGA" name="urlDescarga" type="string"/>
12     </class>
13 </hibernate-mapping>
```

Fig. 24: Fichero Apunte.hbm.xml.

En la relación de uno a uno fig. 24 entre “Apunte” y “Aporte” para hacerla bidireccional sin añadir información en “Aporte”, rompiendo el esquema de flexibilidad de extensión de una nueva funcionalidad, se ha seguido el siguiente procedimiento. Se mantiene una relación de uno a uno unidireccional como se muestra y la obtención del id no será automática como en la fig. 23 sino que será el mismo identificador del “Aporte” creado. Así desde “Aporte” conocemos la información del “Apunte” que lo contiene por tener el mismo id y desde “Apunte” sabremos su información común en “Aporte” por la relación.

Y por último sobre persistencia, mencionamos la creación de una clase que utiliza el patrón singleton[8] estático que nos ayuda a acceder a la *SessionFactory* desde los DAOs. La *SessionFactory* es aquella que se encarga de decir al sistema, donde se encuentra todos los ficheros de mapeo, el dialecto de Hibernate a utilizar, y también asocia los DAOs dentro de las Fachadas. La clase tiene el nombre *HibernateUtil.java*, al iniciar una operación de persistencia en un DAO, se obtendrá la *SessionFactory* a partir del método getSessionFactory() de la clase y no creando un nuevo objeto como se haría normalmente.

Se puede observar que con nuestro esquema ORM, se obtiene el diseño Entidad-Relación presentado en la fase del análisis. Como se comentó en la fase mencionada cuando desarrollamos el modelo Entidad-Relación, tras la implementa de la persistencia con el esquema ORM, hemos comprobado que si cumple los criterios de aceptación proporcionados por las formas normales.

- Servicio -

Para la implementación del servicio introduciremos teóricamente el modo de trabajar de los servicios REST, expondremos los servicios necesarios y en que sprint han sido creado, viendo si han supuesto algún problema. Y por último, mostraremos algo de código que siempre ayuda a una mejor comprensión de cómo han sido implementados.

Nuestro servicio REST no necesita ningún WSDL (Web Services Description Language) para definir las funciones y los tipos de datos utilizados, como es el caso de los servicios SOAP.

[8] Patrón Singleton (instancia única): está diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto.

Pero bien es cierto que necesitamos una base teórica para trabajar correctamente con ellos, siguiendo los principios establecidos.

REST (REpresentational State Transfer) es considerado un patrón de diseño, un punto clave es que es *stateless*, lo que significa que no mantiene estados. Esto conlleva que en diferentes llamadas el servicio, requiere todos sus datos. Usa el principio HATEOAS[9], donde los mensajes enviados al cliente dan información para conducir el estado del cliente.

Por otro lado, REST se diferencia por estar orientado a recursos, recursos accesible mediante URLs, usando un conjunto pequeño y sencillo de operaciones para manejar los recursos basados en los métodos HTTP: PUT, GET, POST, DELETE. El principio principal para saber si usamos REST correctamente es que "parezca" que los elementos a los que accedemos existen estáticamente en el servidor, donde una URL es un identificador unívoco. Las operaciones para manejar los recursos se resume en:

- GET: Es la operación de lectura, no modifica el estado interno del servidor.
- POST: Usada para crear un nuevo recurso en el servidor.
- DELETE: Modifica el estado interno al eliminar recurso.
- PUT: Para la actualización de un recurso, también modifica el estado interno.

Al realizar una petición al servicio, ésta suele tener una respuesta que sigue los códigos estándares de HTTP [22], como por ejemplo un recurso no encontrado su código es 404.

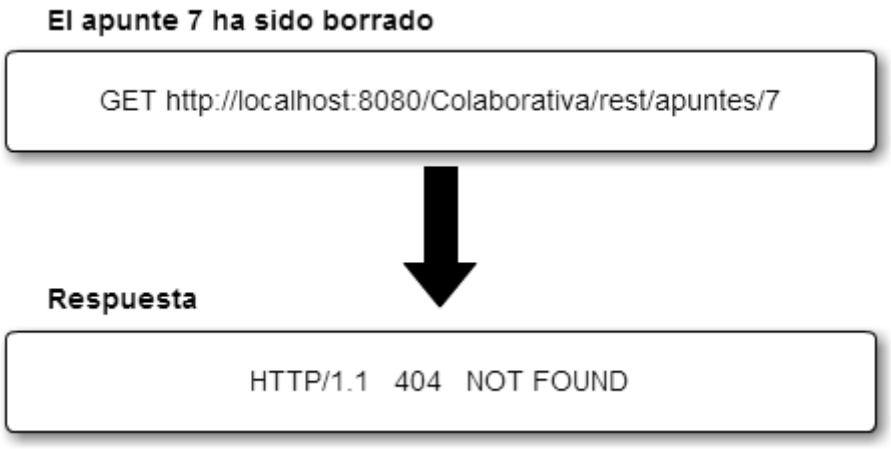


Fig. 25: Petición de lectura el apunte con id = 7.

Comenzaremos con la implementación del servicio realizado, para ello expondremos las URLs de los recursos creados y por último el código de uno de los servicios para explicar la técnica empleada para su creación. Recomendamos a los lectores que nunca hayan creado un servicio web REST ver la conferencia de Ricardo Borillo sobre "Servicio REST con Jersey JAX-RS" [23].

[9] HATEOAS (Hypermedia As The Engine Of Application State).

Para empezar dispondremos de los siguientes servicios:

- **Servicio Persona** que se encarga de gestionar la información de los usuarios del sistema, se implementó en el sprint 1:
  - **Añadir una nueva persona**.

  POST http://localhost:8080/Colaborativa/rest/personas

  - **Modificar a una persona**, donde {usuario} es el id del usuario a modificar.

  PUT http://localhost:8080/Colaborativa/rest/personas/{usuario}

  - **Eliminar a una persona**, donde {usuario} es el id del usuario a eliminar.

  DELETE http://localhost:8080/Colaborativa/rest/personas/{usuario}

  - **Obtener una lista** de todas las personas.

  GET http://localhost:8080/Colaborativa/rest/personas

  - **Obtener una personas**, donde {usuario} es el id del usuario a obtener.

  GET http://localhost:8080/Colaborativa/rest/personas/{usuario}

- **Servicio Login** que se encarga del acceso al sistema, se implementó junto al anterior servicio en el sprint 1:
  - **Login**, para obtener el token de acceso, donde también enviamos en la cabecera el usuario y la contraseña cifrados en *Base64*.

  POST http://localhost:8080/Colaborativa/rest/acceso/login

  - **Logout**, para eliminar el token de acceso (como cerrar sesión, solo que no hay estado) donde {token} es el token de acceso.

  DELETE http://localhost:8080/Colaborativa/rest/acceso/logout/{token}

- **Servicio Noticias** encargado de la funcionalidad noticias, se implementó en el tercer sprint:
  - **Añadir una nueva noticia**.

  POST http://localhost:8080/Colaborativa/rest/noticias

  - **Modificar una noticia**, donde {id} es el id de la noticia a modificar.

  PUT http://localhost:8080/Colaborativa/rest/noticias/{id}

  - **Eliminar una noticia**, donde {id} es el id de la noticia a eliminar.

  DELETE http://localhost:8080/Colaborativa/rest/noticias/{id}

  - **Obtener una noticia**, donde {id} es el id de la noticia a obtener.

  GET http://localhost:8080/Colaborativa/rest/noticias/{id}

  - **Obtener una lista** de las noticias de la página {inicio} con un tamaño de página {tamPagina}.

  GET http://localhost:8080/Colaborativa/rest/noticias?inicio={inicio}
  &tamPagina={tamPagina}

- **Obtener el número de noticias** que hay, para ayudar en la paginación.

GET http://localhost:8080/Colaborativa/rest/noticias/tam

- **Obtener una lista** de las noticias de un, donde {usuario} es el usuario seleccionado.

GET http://localhost:8080/Colaborativa/rest/noticias/propietario?usuario={usuario}

➢ **Servicio Apuntes** encargado de la funcionalidad apuntes, se implementó en el sprint 6 siendo el último servicio:

- **Añadir un nuevo apunte**.

POST http://localhost:8080/Colaborativa/rest/apuntes

- **Modificar un apunte**, donde {id} es el id del apunte a modificar.

PUT http://localhost:8080/Colaborativa/rest/apuntes/{id}

- **Eliminar un apunte**, donde {id} es el id del apunte a eliminar.

DELETE http://localhost:8080/Colaborativa/rest/apuntes/{id}

- **Obtener un apunte**, donde {id} es el id del apunte a obtener.

GET http://localhost:8080/Colaborativa/rest/apuntes/{id}

- **Obtener una lista** de los apuntes de una {carrera}, {curso} y {asignatura}.

GET http://localhost:8080/Colaborativa/rest/apuntes?carrera={carrera}
&curso={curso}&asignatura={asignatura}

- **Obtener una lista** de las carreras disponibles.

GET http://localhost:8080/Colaborativa/rest/apuntes/propiedades/carreras

- **Obtener una lista** de los cursos disponibles de una {carrera}.

GET http://localhost:8080/Colaborativa/rest/apuntes/propiedades/{carrera}/cursos

- **Obtener una lista** de las asignaturas disponibles del {curso} de una {carrera}.

GET http://localhost:8080/Colaborativa/rest/apuntes/propiedades/{carrera}/{curso}
/asignaturas

- **Obtener una lista** de los apuntes de un, donde {usuario} es el usuario seleccionado.

GET http://localhost:8080/Colaborativa/rest/apuntes/propietario?usuario={usuario}

Una vez definidos los servicios, pasamos a mostrar (figura 26) cómo hemos implementado uno de estos servicios anteriores, el servicio noticias. Utilizando en este caso anotaciones para definir los servicios. El resto de servicios serán implementados de forma similar, sin suponer ninguna novedad al lector.

```
@Path("noticias")
public class NoticiaResource {

    private AportesDAO aportesDAO;
    public static String TIPO = "Noticia";

    public NoticiaResource() {...3 lines }

    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.APPLICATION_JSON)
    public Response add(Aporte aporte, @Context UriInfo uriInfo) {...9 lines }

    @PUT
    @Path("{id}")
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.APPLICATION_JSON)
    public Aporte update(Aporte aporte, @PathParam("id") long id) {...12 lines }

    @DELETE
    @Path("{id}")
    @Produces(MediaType.APPLICATION_JSON)
    public Response remove(@PathParam("id") long id) {...8 lines }

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public List<Aporte> getAll(@QueryParam("inicio") int inicio, @QueryParam("tamPagina"

    @GET
    @Path("tam")
    @Produces(MediaType.APPLICATION_JSON)
    public long getTam() {...5 lines }

    @GET
    @Path("{id}")
    @Produces(MediaType.APPLICATION_JSON)
    public Aporte getId(@PathParam("id") long id) {...8 lines }

    @GET
    @Path("propietario")
    @Produces(MediaType.APPLICATION_JSON)
    public List<Aporte> getAllUser(@QueryParam("usuario") String usuario) {...5 lines }

}
```

Fig. 26: Implementación de “Noticias” usando anotaciones.

En la primera iteración hay que destacar que tuvimos que definir el servlet[10] de *Jersey* en nuestro archivo de configuración web.xml[11], sin él no podríamos hacer uso de las capacidades que nos ofrece el framework.

[10] Servlet: Clase de Java utilizada para ampliar las capacidades de un servidor.
[11] web.xml: Es un descriptor de despliegue más conocido como fichero de configuración donde se describe cómo se debe desplegar el servicio o la aplicación web.

```
<servlet>
    <description>JAX-RS Tools Generated - Do not modify</description>
    <servlet-name>jaxrs-servlet</servlet-name>
    <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</se
    <init-param>
        <param-name>com.sun.jersey.config.property.packages</param-name>
        <param-value>es.tfg.controlador.services</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>jaxrs-servlet</servlet-name>
    <url-pattern>/rest/*</url-pattern>
</servlet-mapping>
```

Fig. 27: Descripción del servlet de *Jersey* en web.xml.

Como vemos en la descripción del servlet, con "param-value" definimos dónde situaremos los servicios creados como el de la figura 26, y con "url-pattern" decimos la URL con la que se accede a los servicios.

- Seguridad -

Ya comentamos que REST es *stateless*, lo que significa que no mantiene estados. Al no haber estado, no hay necesidad de que los servicios guarden las sesiones de los usuarios y por lo tanto cada petición al servicio es independiente de las demás. Existe un mecanismo estándar de autenticación denominado *HTTP Basic*, teniendo que enviar los credenciales en cada petición. Consiste en enviar login y password en Base64[12] dentro de la cabecera *Authorization*.

```
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

Fig. 28: HTTP Basic.

Pero tener que enviar usuario y contraseña sin cifrar en cada llamada, es algo muy inseguro. Por lo tanto decidimos hacer una identificación por token. Un token de autenticación es un valor que nos identifica frente al servidor, este valor se obtiene normalmente tras hacer login con usuario y contraseña tradicionales. A diferencia de una contraseña, el token se puede anular o hacer caducar sin causar excesivas molestias al usuario.

En resumen los pasos serían: hacer login como se haría normalmente, recibimos el token el cual hay que enviar en cada petición. Los token lo almacenaremos en la base de datos para su comprobación en las diferentes peticiones, esta gestión de los tokens la llevaremos manualmente ya que no supone mucho esfuerzo.

[12] Base64 no cifra los datos, se usa para el envío correcto de los caracteres por HTTP.

Ahora nos toca ver como se ha implementado todo esto, la seguridad completa se ha llevado a cabo en dos sprint. Casualmente el primero y el último, en el primero definimos la gestión de los tokens y finalmente se ha implementado el filtro que mantiene la seguridad. No ha supuesto ningún problema ésta división tan distante en la seguridad pues, durante todo el desarrollo hemos trabajado con los tokens como si la seguridad existiera. Y en la última fase, hemos creado el filtro que controla el acceso a los servicios.

Por lo tanto primero vemos cómo gestionamos los token, y luego, cómo hemos implementado la seguridad. Para comprobar los tokens debemos almacenarlos en la base de datos, creando una clase token y su DAO correspondiente (el cómo es implementado, se explica en la persistencia). Un token contiene un identificador único, una fecha de caducidad y el id del usuario al que pertenece. Las operaciones para su gestión son login, logout y verificar el token, devolviendo si es correcto, el usuario al que pertenece.

Antes de nada, hemos creado un filtro al acceder a ciertas URLs que son los recursos a proteger, el filtro está definido en *RestAuthenticationFilter.java* cuya descripción se muestra a continuación.

```
<filter>
    <filter-name>AuthenticationFilter</filter-name>
    <filter-class>es.tfg.controlador.security.RestAuthenticationFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>AuthenticationFilter</filter-name>
    <url-pattern>/rest/personas/*</url-pattern>
    <url-pattern>/rest/noticias/*</url-pattern>
    <url-pattern>/rest/apuntes/*</url-pattern>
</filter-mapping>
```

Fig. 29: Descripción del filtro en web.xml.

Hay que mencionar que al añadir un nuevo servicio, el nuevo servicio debe ser definido con “url-pattern” en el fichero de configuración para ser filtrado. Como parámetro de “url-pattern” debemos darle la url con la que accedemos a sus servicios.

RestAuthenticationFilter.java es una clase encargada de la seguridad del servicio, donde obtiene el token de la cabecera y con él comprueba si el usuario está autenticado. Luego con el usuario al que le pertenece el token comprueba si tiene autorización para acceder a la petición demandada, si es así deja pasar el filtro, en caso contrario devuelve que no está autorizado. Para realizar esto *RestAuthenticationFilter.java* se ayuda de la clase *AuthenticationService.java* que contiene dos métodos básicos que son: authenticate, para comprobar que el token sea correcto y authoritate, que comprueba que cumpla los requisitos de seguridad para el acceso demandado.

En definitiva la estructura y distribución de los archivos necesarios para la creación de nuestro servicio web REST la mostramos en la siguiente figura.

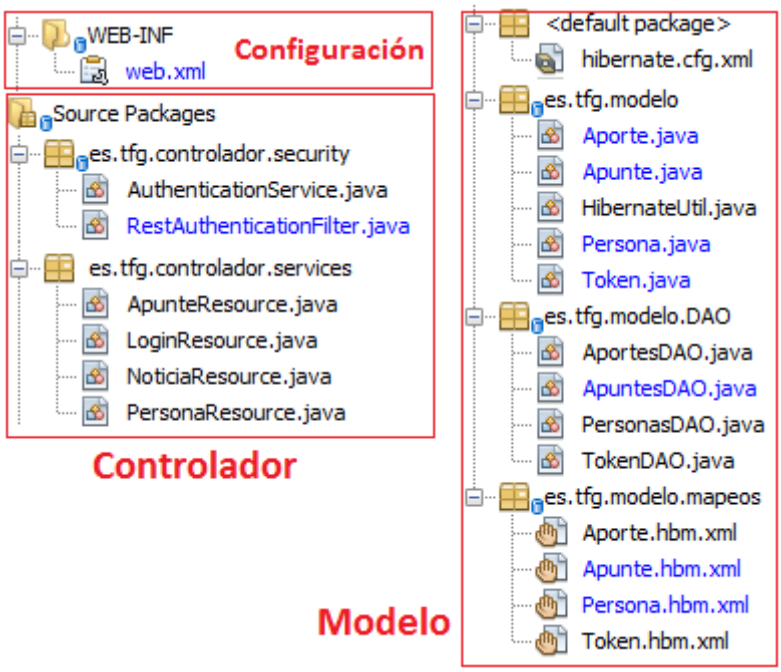


Fig.30: Estructura y ficheros del proyecto.

4.2.1. Implementación Cliente

Sobre la implementación de la aplicación cliente comentaremos aspectos importantes de la aplicación web en general, como también aspectos de *AngularJS*. Sobre *AngularJS* no vamos a entrar en detalles muy específicos del frameworks, sólo lo que necesitamos saber para entender la implementación. También comentaremos cómo hemos desarrollado los elementos del patrón MVC y el cambio de diseño que ha supuesto para su creación.

Antes de empezar con el desarrollo del primer sprint, se nos planteó una decisión. Por experiencia sabíamos que nuestro servidor *Apache Tomcat* [24] por tema de seguridad, no nos iba a dejar acceder a un puerto diferente. Y si nuestra aplicación web del lado del cliente no puede acceder al puerto del servicio web, no dispondría de la lógica de negocio necesaria. Para solucionar este inconveniente existen multitud de soluciones, nosotros pudimos optar por la más fácil gracias a que a la hora de crear nuestro servicio web, creamos un proyecto como aplicación web. Por lo tanto, la solución es crear nuestra aplicación en el mismo proyecto creado para el servicio, haciendo esto compartimos el puerto y evitamos el problema de acceso.

Antes de entrar en la implementación de *AngularJS,* tuvimos una primera historia de usuario que nos dejó con la entrega del primer sprint inacabada. Pues que tuvimos que realizar en paralelo las historias de usuario sobre la interfaz del segundo sprint. Es decir acabamos correctamente las historias de usuario relacionadas con el servicio pero no las relacionadas con la aplicación web al realizar seguidas las historias de usuario INT1AW, INT2AW e INT3AW, donde las dos últimas son pertenecientes al segundo sprint.

En ellas, entre otras cosas, estudiamos la posibilidad de uso de *las plantillas* (o *Templates*), en el contexto web, pueden entenderse para su comprensión como una plantilla *Word* que proporciona la estructura del documento donde sólo se necesita introducir los datos en los campos necesarios. De forma similar es en el contexto web, utilizan unos sistemas de plantilla para separar el estilo proporcionado por el CSS y la lógica de la visualización (JavaScript). Lo que agiliza bastante el trabajo proporcionando un diseño completo, lo que ha marcado un modo de negocio disponiendo de plantillas básicas que son gratuitas o plantillas más complejas por las que debe pagar. Las plantillas de pago además, suelen dispone de una muy buena documentación para un correcto uso y aprendizaje.

Investigando por internet aprendiendo más sobre *AngularJS*, descubrimos una plantilla que se adapta a la perfección a lo que buscábamos. Si usarla o no, no supuso ninguna duda pues su descarga era gratuita [25] y con pocas modificaciones quedaría acoplada en nuestro proyecto.

Ya en el segundo sprint, como se comentó en el punto 3.2. cuando veíamos los diagramas de clases de diseño, se tuvo que plantear un cambio en el diseño. Dado que el modo de trabajar del framework era MVVM y no MVC. En resumen, no existe una definición del ViewModel como tal, sino que cada Controlador tiene asociado un objeto $scope sobre el que construirlo (actuando en perspectiva como controlador pero son funciones controladoras).

Para obtener el Modelo-Vista-Controlador, ya no se trabajaría con un controlador por entidad, sino que dispondremos de un controlador que inicializa las funciones controladoras necesaria para la vista a tratar, su diseño se puede ver en la figura 12*.

Antes de ver el código de su implementación, comentamos que durante el desarrollo del proyecto en los diferentes sprints hemos ido introduciendo gracias al gestor de inyecciones del framework diferentes módulos. Estos módulos nos han ayudado en diferentes puntos de la implementación de nuestro proyecto. En la siguiente figura mostramos los módulos utilizados y hablaremos de ellos cuando sea necesario.

```
var app = angular.module('ClienteApp',
    'ClienteApp.services',
    'ClienteApp.controllers',
    'ui.router',
    'ngCookies',
    'ngMaterial',
    'ngAnimate',
    'ngMdIcons',
    "brantwills.paging",
    "flow"
]);
```

Fig. 31: Inyecciones de los módulos necesarios.

En el segundo sprint donde desarrollamos el registro y el logeo necesitamos:

- “ClienteApp.services” donde esta nuestro modelo.DAO que se encarga de realizar las peticiones a nuestro servicio web.

- “ClienteApp.controllers” que contiene las funciones controladora de las diferentes vistas que es llamado por nuestro controlador.
- “ui.router” usado por nuestro controlador para el enrutamiento de la aplicación, donde a cada vista se le otorga una url y sus funciones controladoras que harán uso del modelo si fuese necesario.
- “ngCookies” módulo que nos ayudará a guardar el token en una cookie en nuestro navegador.
- “ngMaterial”, “ngAnimate” y “ngMdIcons” módulos que nos ayudan a seguir los principios de material design en el diseño de la interfaz de la aplicación web. Aunque éste últimos, no nos servirá hasta el sprint 5 cuando mostremos las operaciones posibles con el uso de los iconos.

Como ya hemos visto, el módulo “ui.router” nos ayudará a implementar el controlador, donde con el comando $stateProvider vamos definiendo el control de nuestro sistema. Por ejemplo para el segundo sprint:

```
$stateProvider
        .state('base', {
            abstract: true,
            url: '',
            templateUrl: 'views/base.html',
            controller: 'baseController'
        })
        .state('login', {
            url: '/login',
            parent: 'base',
            templateUrl: 'views/inicioLogin.html',
            controller: 'loginController'
        })
        .state('registro', {
            url: '/registro',
            parent: 'base',
            templateUrl: 'views/inicioRegistro.html',
            controller: 'registroController'
        })
```

Fig. 32: Implementación del controlador en el sprint 2.

Tenemos un estado “base”, es una vista abstracta que estará presente en toda la aplicación para la comprobación del estado del token en caso de caducidad, su única función es saber si dispone de un token válido al inicio de la aplicación para llevarte al login o a la página de inicio. Recordemos que la comprobación del token se lleva a cabo en cada petición. También tenemos el control para login y registro, definiendo su ruta, donde está su vista y las funciones controladoras para la vista.

Ahora veamos un ejemplo de las funciones controladoras, presentes en el fichero controllers.js siendo un módulo añadido como hemos visto anteriormente.

```
angular.module('ClienteApp.controllers', []).
        controller('baseController', function (auth) {
        controller('colaborativaController', function
        controller('loginController', function ($scope
        controller('registroController', function ($sc
        controller('noticiasTodasController', function
        controller('noticiasNuevaController', function
        controller('noticiaIdController', function ($s
        controller('noticiasMiasController', function
        controller('noticiasEditarController', functio
        controller('ApuntesTodosController', function
        controller('ApuntesSubirController', function
        controller('apunteIdController', function ($sc
        controller('apuntesMiosController', function (
```

Fig. 33: Todas las funciones controladoras usadas.

```
controller('loginController', function ($scope, personasAPIservice, auth) {
    $scope.progressVisibility = false;
    $scope.errorVisibility = false;
    $scope.login = function ()
    {
        $scope.progressVisibility = true;
        var response = personasAPIservice.login($scope);
        response.success(json = function (data, status, headers, config) {
            $scope.data = data;
            if ($scope.data != null && status == "201") {
                auth.login(data);
            }
        });
        response.error(function (data, status, headers, config) {...13 lines });
    }
}).
```

Fig. 34: Ejemplo de función controladora para el login.

El módulo de las funciones controladoras así como el de los servicios (modelo) que se comunica con el servidor web, se han definido en un archivo usando el comando `angular.module('nombre del módulo', ['dependencias'])`. Dependiendo de si es controlador irán añadiendo los diferentes `.controler` donde definimos las funciones controladoras o añadiendo `.factory` donde definimos las funciones del modelo que se comunicará con el servidor figura 35.

En la figura 34 tenemos un ejemplo de la definición de las funciones e inicialización de la vista login. Donde inicializamos las variables como booleanos de la visibilidad de ciertos mensajes o elementos y definimos la función login, que será llamada al pulsar el botón correspondiente. Lo único que hacemos en la función login es mostrar la barra de progreso, como que se está procesando la petición y usar el modelo para realizar el login con los datos del formulario que está definidos en el $scope.

Por último para terminar la definición del modelo MVC, es describir la implementación del modelo que incluye la interfaz de comunicación con el servicio web, razón por la que se ha denotado como APIservice.

```
angular.module('ClienteApp.services', [])
        .factory('personasAPIservice', function ($http) {...45 lines })
        .factory('noticiasAPIservice', function ($http, $cookies) {...101 lines })
        .factory('apuntesAPIservice', function ($http, $cookies) {...114 lines });
```

Fig. 35: Definición de los servicios del modelo.

```
.factory('personasAPIservice', function ($http) {

    var personasAPI = {};

    personasAPI.login = function ($scope) {
        var config = {
            method: "POST",
            headers: {authorization: "Basic "
                        + btoa($scope.username + ":"
                                + $scope.password)},
            url: "http://localhost:8080/ServiceRestRegiter/rest/acceso/login"
        }


        return $http(config);

    }
```

Fig. 36: Implementación de la función login del modelo.

Las peticiones al servidor, como se muestra en la figura 36, gracias a *AngularJS* se realiza de un modo realmente simple donde solo definimos el método, la url y si fuese necesario los elementos a añadir a la cabecera.

Con esto dejamos explicado todo lo que debemos entender de *AngularJS*, pero ahora hablaremos de otros aspectos importantes donde pueden surgir dudas sobre su implementación y uso. Como son el uso de las cookies, la paginación y subida de fotos y subidas apuntes.

Para cualquier tema relacionado con las cookies, creamos una factoría como se puede ver en el diagrama de clases, finalmente la llamamos “auth” por facilidad al llamarla en vez de “autorizacion”. La factoría “auth” es incluida en las diferentes funciones controladoras y en el caso de no estar autorizado en alguna petición significa que la cookie ha caducado y procedería a hacer logout. Para las APIs del servicio del modelo no necesita usar esta factoría con llamar $cookie sería suficiente.

En segundo lugar, podemos tener duda con la paginación, los elementos a tener en cuenta en toda paginación son: el total de elementos del listado (`$scope.total`), el número de elementos por página (`$scope.pageSize`) y la página en la que nos encontramos (`$scope.currentPage`). Por lo tanto, durante el sprint 5 desarrollamos la paginación de la siguiente manera.

```
<div class="text-center">
    <div ng-click="cambioPagina()" paging
         page="currentPage"
         page-size="pageSize"
         total="total">
    </div>
</div>
```

Fig. 37: Definición de la paginación en el HTML.

```
controller('noticiasTodasController', function ($scope, noticiasAPIservice, auth) {
    $scope.currentPage = 1;
    $scope.pageSize = 2;
    $scope.total = 0;
    var response = noticiasAPIservice.getNoticias(1, $scope.pageSize);
    response.success(json = function (data, status, headers, config) {...19 lines });
    response.error(function (data, status, headers, config) {...17 lines });

    $scope.cambioPagina = function () {
        var response = noticiasAPIservice.getNoticias($scope.currentPage, $scope.pageSize);
        response.success(json = function (data, status, headers, config) {...10 lines });
        response.error(function (data, status, headers, config) {...17 lines });
    }
}).
```

Fig. 38: Función controladora de listado de noticias, con función cambioPagina().

Así de sencillo ha sido llevar la paginación gracias al módulo “brantwills.paging”, definiendo en el HTML el `<div paging …>` en la que definimos también las variables del $scope que son necesarias las cuales ya hemos comentado. El módulo se encarga de gestionar la paginación, tan solo nos queda definir la función del cambio de página donde como podemos ver en la figura 38, se llama al servicio para obtener las noticias de la página que es gestionada por el módulo.

Para acabar, vemos el modo de subir fotos o archivos al servidor. Se han tomado decisiones diferentes en ambos casos, siendo la más rápida y sencilla de explicar la subida de documentos como es el caso de apuntes. Dado que nuestra aplicación no se centrará en una funcionalidad concreta completamente, buscamos las soluciones más sencilla y rápida para ayudar al alumno. Por lo que no crearemos una funcionalidad completa de almacenamiento de archivos como es el caso de la aplicación de Google Drive. Podemos almacenarla en aplicaciones similares donde te ofrecen un enlace público del archivo en la que se le incluye la funcionalidad completa como (visualizar, descargar…). Nuestro trabajo al subir un apunte, será almacenar éste enlace y no el archivo, liberando también de éste modo de la sobrecarga de almacenamiento a nuestro servidor.

En el caso de las imágenes como es el caso de noticia, es diferente, ya si es lógico que una noticia disponga de su propia imagen y que ésta imagen no esté en otra aplicación y/o en otro servidor. Dicho esto podemos deducir que tuvimos que decidir en el sprint 5 a la hora de subir una noticia, ¿Cómo subir la imagen? Decidimos que se subiera y almacenará en el servidor en un tipo *string* (cuyo tamaño es tanto como disponga en memoria), el modo de hacer esto posible es cifrando la imagen en base64.

4.3.- Pruebas

A lo largo de la implementación se han ido realizando diferentes pruebas para obtener un software de calidad, pero no ha sido creando e implementando diferentes casos de pruebas. Al realizar un prototipo la calidad no necesariamente tiene que ser perfecta, pero tampoco por dicho motivo no realizar un control de calidad mínimo.

Nuestro modo de actuar ha sido aprovechar las historias de usuario de la planificación ágil, ya que definen la funcionalidad de los casos de uso, definiendo los criterios de aceptación podemos obtener lo más parecido a unos objetivos de prueba.

Es decir, conforme iba implementando un sprint, al terminar una historia de usuario, comprobaba que se cumplían los criterios de aceptación. Pasemos a un ejemplo práctico para entenderlo mejor:

ID	Historia de usuario	Criterio de aceptación	Puntos
AC6AW	**Como** usuario **Quiero** logearme **Para poder** identificarse y acceder al sistema	a) Todos los campos obligatorios rellenos antes de enviar la comprobación. b) Si es correcta accede al sistema. c) Si no es correcta se mostrará un mensaje de error.	5.0
AC2SW	**Como** desarrollador **Quiero** disponer de un servicio personas **Para poder** hacer un CRUD sobre la información de persona	a) Respetar la arquitectura REST.	3.5

En la primera historia de usuario, la historia AC6AW, es una funcionalidad de la aplicación cliente concretamente la funcionalidad de logeo. Los criterios de aceptación a cumplir son tres y pasemos a comprobar que se cumple:

a. Todos los campos obligatorios rellenos antes de enviar la comprobación: Si observamos la figura 39, el botón login se encuentra desactivado pues el campo obligatorio “Password” está sin rellenar.
b. Si es correcta accede al sistema: Al introducir el usuario y contraseña válidos, nos lleva a la ventana inicio de usuario identificados.
c. Si no es correcta se mostrará un mensaje de error: Siendo el mensaje de fondo rojo de la figura 39.

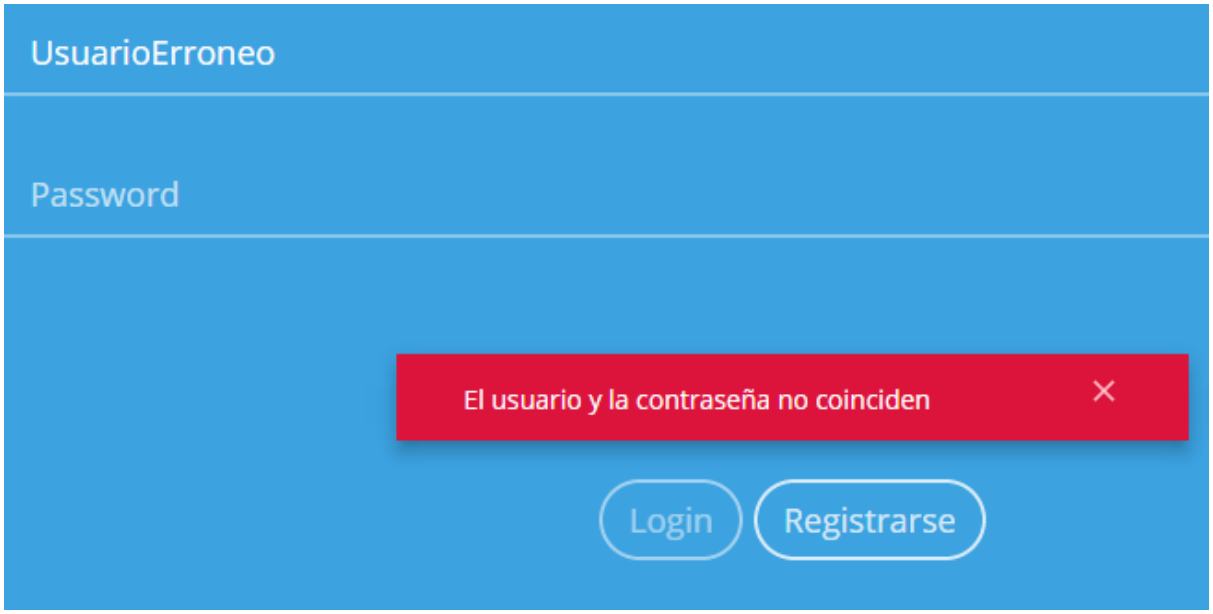


Fig. 39: Comprobación de criterios de aceptación de AC6AW.

Aun así, puede que algunos criterios de aceptación no abarquen del todo un control de calidad necesario, con lo que también hemos ido haciendo comprobaciones basada en nuestra experiencia. Un ejemplo sería, el incluir el aspa para cerrar la ventana del mensaje de error puesto que puede resultar molesto y tras su notificación ya no es necesario.

En historia de usuario siguiente (AC2SW) debemos respetar la arquitectura REST, además de las comprobaciones adicionales que hayamos considerado oportunas. Para las historias de usuario correspondiente al servicio web no disponemos de una interfaz para éstas comprobaciones y usamos una extensión de Google, Avanced Rest Client ya mencionada anteriormente.

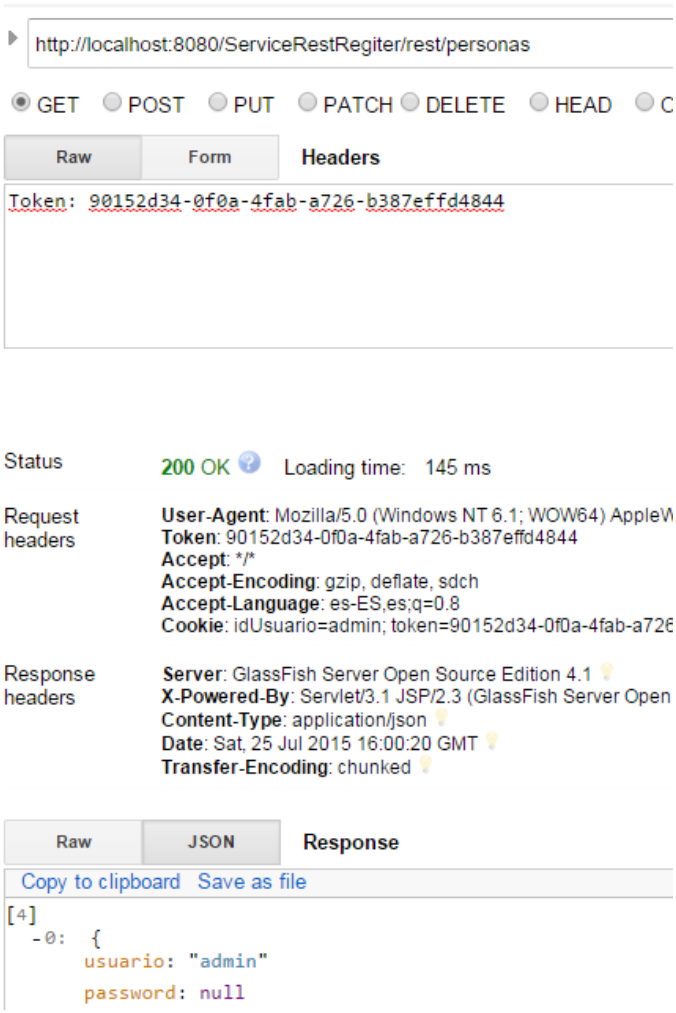


Fig. 40: Comprobación petición listado de personas.

Se comprueba que el código HTTP sea correcto, que el listado está completo, que no se envía la contraseña de los usuarios (sería una vulnerabilidad enorme), etc.

Como vemos, podemos realizar un control de calidad de forma iterativa gracias a las historias de usuario que supone además como ventaja, un consumo ínfimo de recurso.

5.- Conclusiones

En la siguiente memoria se ha realizado un estudio sobre las necesidades de los alumnos dentro del Campus Universitario, planteando que tipo de plataforma puede suplir estas necesidades. Se ha propuesto un diseño de una aplicación para establecer el contacto e intercambio de información entre alumnos, siempre pensando en ellos para obtener el diseño que mejor se adapta al problema planteado. Y por último se ha implementado un prototipo utilizando diferentes frameworks que nos han ayudado durante la implementación, planteándonos las posibilidades de las que disponemos y la necesidad de su uso. Con los resultados alcanzados se puede concluir que se han cubierto los objetivos inicialmente propuestos

La tecnología, siempre que se use con control o en el contexto adecuado puede suponer una herramienta para el estudiante universitario. Tras el desarrollo del prototipo y como propio estudiante, he podido observar que: como estudiante muchas veces no es fácil obtener lo que buscas y como desarrollador tampoco es fácil proporcionar una solución. Hablamos en el análisis preliminar, donde realizaba un estudio de las necesidades, que una solución horizontal donde acotamos el contexto en el campus universitario era una buena solución. En éste punto del trabajo, sigo opinando lo mismo y la base ha quedado realizada, solo hay que dedicarle más tiempo puesto que, el contexto de campus universitario sigue siendo un contexto bastante amplio.

Durante la memoria podemos observar las ventajas de haber utilizado una metodología ágil, aun decidiendo realizar un modelado completo entrando en la burocracia del análisis y el diseño tradicional. La ventaja se ha podido observar tanto en la ayuda de obtener los requisitos que el cliente quiere, incorporándolo en el proyecto, como también durante la planificación y el desarrollo del proyecto, gracias a las historias de usuario y el *product backlog*.

El enfoque tomado para la arquitectura general del proyecto, de crear un servicio REST y una aplicación cliente web, ha resultado ser la correcta. Cierto es que el uso de unas tecnologías ha resultado ser más cómodas que otras, no más correctas sino más cómodas como opinión personal. Es decir, los avances en el servicio no ha supuesto ningún inconveniente, ni complicación, sin embargo, en la aplicación usando *AngularJS* ha supuesto algunas horas más de reflexión antes de pasar a la fase de implementación.

Con ello no quiero decir que la elección del framework no sea el correcto tampoco, sino que me ha llevado un tiempo de aprendizaje, lo que a nivel personal es satisfactorio. Todo ingeniero de software hoy en día debe estar en un proceso continuo de formación, los meses que he estado trabajando en una empresa y los meses que he estado trabajando en un proyecto como es el TFG me han supuesto aprender nuevas tecnologías. Y creo que la formación universitaria recibida ha ayudado a obtener esta habilidad de continua evolución y la habilidad de ser autodidacta.

No es la primera vez que he utilizado una metodología ágil en un proyecto con cierta envergadura, ya realicé un proyecto junto a otros tres compañeros en las prácticas conjuntas de dos asignaturas “Desarrollo de Aplicaciones Web” y “Desarrollo Ágil”. Ambos desarrollos han aportados cosas diferentes, el primero era en un equipo de 4 personas

pudiendo aplicar otras técnicas de esta metodologías. Sin embargo, en el desarrollo de este prototipo he podido observar la verdadera utilidad del desarrollo ágil, la posibilidad del cambio de diseño.

Como comenté, al aprender *AngularJS* surgieron varias complicaciones y en unas de ellas produjo un cambio de diseño, que si hubiera seguido una metodología tradicional, tal vez hubiera supuesto el fracaso del proyecto.

Por último, antes de pasar al siguiente apartado, he de reconocer que la realización del TFG ha sido un trabajo donde no solo he aplicado los conocimientos adquiridos durante la formación universitaria. Sino, un trabajo donde he tenido que adquirir información para realizar un proceso completo de ingeniería ayudándome de los conocimientos adquiridos. No ha sido un simple guion de práctica donde los puntos ya estaban fijados, todo lo contrario, los puntos han sido decisiones que he tenido que estudiar y tomar.

5.1.- Mejoras y trabajos futuros

El sistema a desarrollar recordemos es un prototipo. Aunque es funcional y se han controlado bastantes aspectos de calidad, hay que resaltar lo que es, un prototipo, por lo tanto debemos continuar trabajando para convertirlo en un software realmente funcional.

La base ya está, la cual cumple todo lo requisito del mejor modo posible utilizando la tecnología analizadas. Pero es posible con el tiempo encontrar otras tecnologías, por ejemplo si seguimos en la tendencia de servicio y cliente utilizando *AngularJS*, tal vez nos deberíamos plantear analizar y estudiar el uso del stack *MEAN.JS* [26]. *MEAN.JS* nos ayuda a construir aplicaciones web usando *MongoDB*, *Express*, *AngularJS* y *Node.js* eliminando suciedad en el código y los errores comunes. La ventaja que nos proporciona es que podríamos crear fácilmente la siguiente arquitectura:

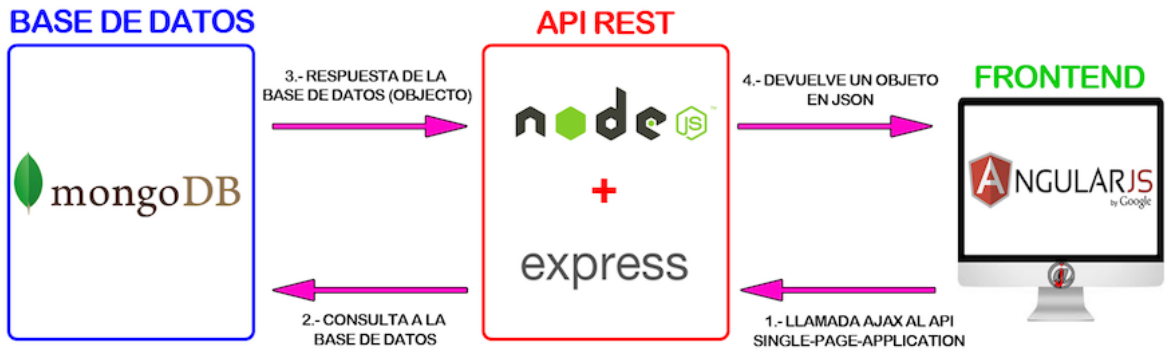


Fig. 41: Arquitectura MEAN (Fuente: http://goo.gl/iRtuLi)

Sin embargo, resulta mucho más cómodo y fácil seguir la base del prototipo creado para los futuros trabajos. Los posibles trabajos a desarrollar ahora son fácilmente reconocible, como ya comentamos existen infinidad de funcionalidad que se deben abordar del modo más sencillo posible. Podríamos seguir con las siguientes funcionalidades cuya idea básica de actuar sería:

- **Compartir piso:** Tendríamos que añadirle además de la información de una aportación, información adicional como es el número de habitaciones, habitaciones disponibles, precio de alquiler, precio adicional (como ADSL), … El modo de actuar para la comunicación es que cada usuario como máximo sólo podrá ofertar-publicar un anuncio. Los interesados se suscriben al anuncio y el propietario podrá ver los datos necesarios para crear el modo de contacto entre ellos como por ejemplo el correo electrónico.

- **Grupo de estudio:** Donde tendrán objetivos comunes como: estudiar para el próximo examen de POO, comprobar los resultados de la relación de ejercicios no corregida en clase de BBDD I, etc. Teniendo información similar a compartir apuntes como es carrera, asignatura,... para poder filtrar lo que buscas. Se fijará al crear el evento la fecha, hora y lugar en la que tendrá lugar, los interesados se pueden dar de alta en el grupo y así se actualizará el evento con las personas que participarán.

Como vemos, intentamos ofrecer una solución fácil y cómoda tanto para el estudiante, como para el desarrollador de la funcionalidad. Se busca ofrecer una solución fácil a una nueva demanda surgida de un modo sencillo y rápido.

6.- Bibliografía

[1] Tomás González. (2014). *Busco en la inmensidad de la Red… y encuentro demasiado*. Computer Hoy. [Online]. Disponible en: http://goo.gl/1kYK5r Acceso: Recuperado en Marzo, 2015

[2] *Los 4 Framework Web Java más usados*. (2014). *JavaHispano*. [Online]. Disponible en: http://goo.gl/h6Qwuh Acceso: Recuperado en Abril, 2015

[3] *Cómo usan Internet en sus estudios los alumnos hispanohablantes*. (3 febrero, 2015). Blog toyoutome. [Online] Disponible en: URL http://goo.gl/S3WcKs Acceso: Recuperado en Marzo, 2015

[4] *Advanced REST client*. (3 febrero, 2015). Chrome web store. [Online] Disponible en: URL https://goo.gl/DqfKM7 Acceso: Recuperado en Enero, 2015

[5] *Jersey - Service Web REST en Java*. Jersey. [Online] Disponible en: URL https://jersey.java.net Acceso: Recuperado en Enero, 2015

[6] *Los 5 mejores frameworks MVC de Javascript*. (2013). CODEJOBS Aprende a programar. Acceso: Disponible en: URL http://goo.gl/ht6NO9 Acceso: Recuperado en Febrero, 2015

[7] *E. Freeman y E. Freeman. O'Reilly*. (2014). "Chapter 12. Patterns of Patterns". Head First Design Patterns. Acceso: Disponible online UJA http://avalos.ujaen.es/record=b1571392

[8] *Data Access Objetc*. CodeFutures. Acceso: Disponible en URL http://codefutures.com/data-access-object/ Acceso: Recuperado en Abril, 2015

[9] *E.M. Jiménez*. Bubok publishing S.L. (2011). Ingeniería del software ágil. Acceso: Disponible biblioteca UJA.

[10] *Resolución de 8 de febrero de 2013, de la Dirección General de Empleo, por la que se registra y publica el IX Convenio colectivo de Telefónica Telecomunicaciones Públicas, SA*. (26 febrero, 2014). Agencia Estatal Boletín Oficial del Estado. [Online] Disponible en: URL https://www.boe.es/buscar/doc.php?id=BOE-A-2013-2153 Acceso: Recuperado en Febrero, 2015

[11] *Unified Modeling Language, Resource Page*. Unified Modeling Language. [Online] Disponible en: URL Acceso: http://www.uml.org/ Recuperado en Enero, 2015

[12] *Leffingwell, Dean(2007). Scaling Software Agility: Best Practices for Large Enterprises*. Acceso: Disponible online UJA http://goo.gl/u2XH2y

[13] *Frameworks javascript para el Patrón MVVM (2013).* Wordpress de Rodrigo Pérez Burgues. [Online] Disponible en: URL https://goo.gl/xbp9Z4 Recuperado en Abril, 2015

[14] *Introducción a la interacción hombre-máquina.* Pegueros. [Online] Disponible en: URL http://www.pegueros.net/wp/wp-content/uploads/2015/03/comunicacion1.pdf Acceso: Recuperado en Marzo, 2015

[15] *Material design - Introduction.* Google Design. [Online] Disponible en: URL https://www.google.com/design/spec/material-design/introduction.html#introduction-goals Acceso: Recuperado en Marzo, 2015

[16] *Teoría del color y su aplicación en el diseño web.* Ipixel. [Online] Disponible en: URL http://www.ipixelestudio.com/teoria-color-diseno-web.html Acceso: Recuperado en Abril, 2015

[17] *Material Design Example.* Material Palette. [Online] Disponible en: URL http://www.materialpalette.com/blue/teal Acceso: Recuperado en Abril, 2015

[18] *XAMPP*. Wikipedia. [Online] Disponible en: URL https://es.wikipedia.org/wiki/XAMPP Acceso: Recuperado en Enero, 2015

[19] *Introducción a Hibernate*. (14 agosto, 2011). Mi granito de Java. [Online] Disponible en: URL http://goo.gl/nmbmld Acceso: Recuperado en Enero, 2015

[20] *Alberto Basalo y Miguel Ángel Álvarez - Qué es AngularJS*. (28 agosto, 2014). Desarrollo Web. [Online] Disponible en: URL http://goo.gl/unWYjN Acceso: Recuperado en Febrero, 2015

[21] *Curso de Hibernate con Spring* (Noviembre 2014). Curso Hibernate. [Online] Disponible en: URL http://cursohibernate.es/doku.php Acceso: Recuperado en Febrero, 2015

[22] *Códigos de estado HTTP* (última mod. Febrero 2015). Wikipedia. [Online] Disponible en: URL https://es.wikipedia.org/wiki/Anexo:C%C3%B3digos_de_estado_HTTP Acceso: Recuperado en Febrero, 2015

[23] *Servicio REST con Jersey JAX-RS* (Octubre 2014). Youtube. [Online] Disponible en: URL https://www.youtube.com/watch?v=Jk8fjj_q1Ew Acceso: Recuperado en Marzo, 2015

[24] *Tomcat* (Abril 2015). Wikipedia. [Online] Disponible en: URL https://es.wikipedia.org/wiki/Tomcat Acceso: Recuperado en Mayo, 2015

[25] *Free Angular JS Theme and Templates*. Start Angular. [Online] Disponible en: URL http://startangular.com/ Acceso: Recuperado en Febrero, 2015

[26] *Open-Source Full-Stack Solution For MEAN Applications*. MEAN.JS. [Online] Disponible en: URL http://meanjs.org/ Acceso: Recuperado en Junio, 2015

Apéndice I. Manual de Instalación del sistema

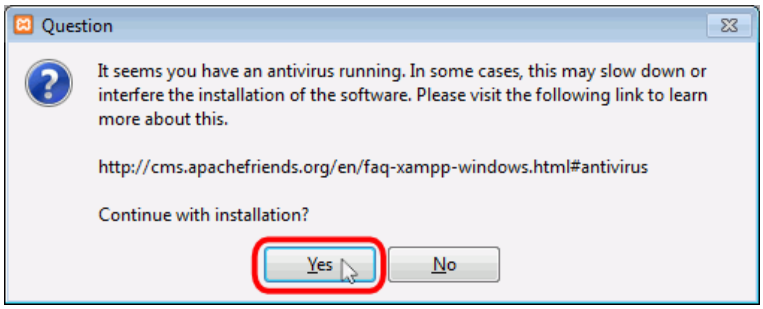
Para la construcción del proyecto, hemos usado el entorno de desarrollo *NetBeans* en su versión 8.0.2. Y para la creación de la base de datos MySQL, hemos utilizado el gestor de base de datos XAMPP.

A continuación mostraremos cómo instalar los programas necesarios para Windows y por últimos, los pasos a seguir hasta su ejecución.

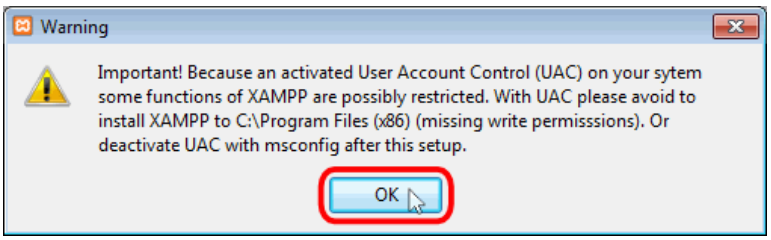
1. Instalación gestor de base de datos XAMPP

Antes de empezar con la instalación, tenemos que descargar la última versión de XAMPP en el siguiente enlace: https://www.apachefriends.org/es/download.html, una vez descargado el ejecutable, los pasos son muy sencillos.

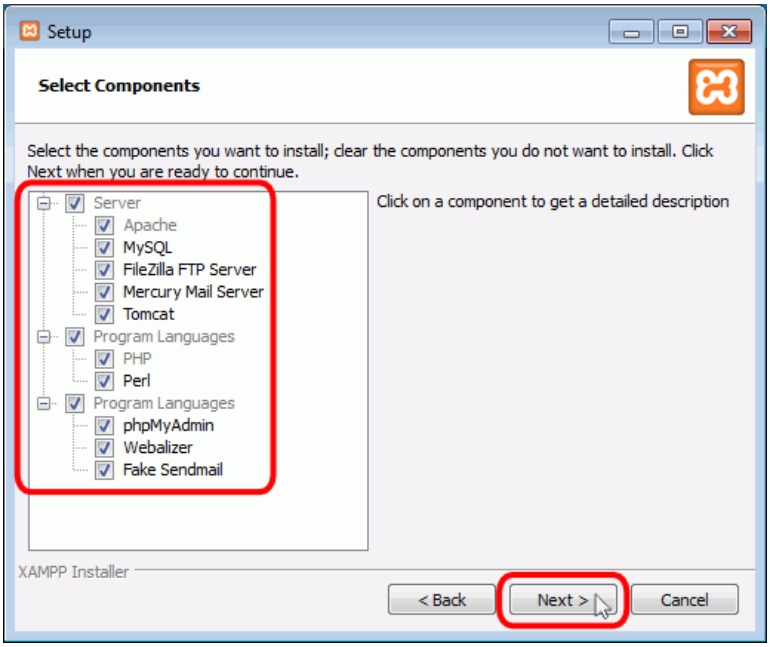
Al arrancar el ejecutable, lo primero que nos aparece es una ventana que nos pregunta si en el ordenador hay instalado un antivirus:



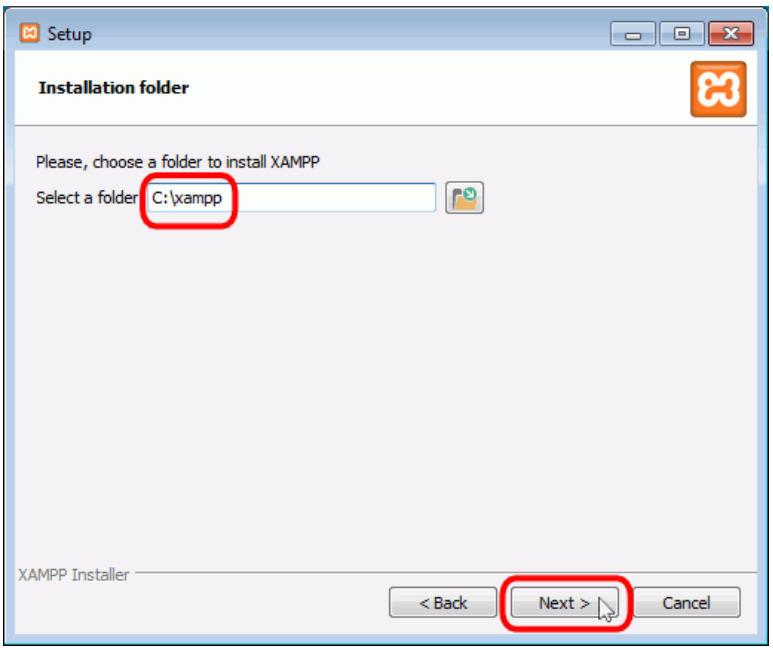
En segundo lugar, nos recuerda si está activado el Control de Cuentas de Usuario que algunos directivos tienen permisos restringidos, simplemente es informativo:



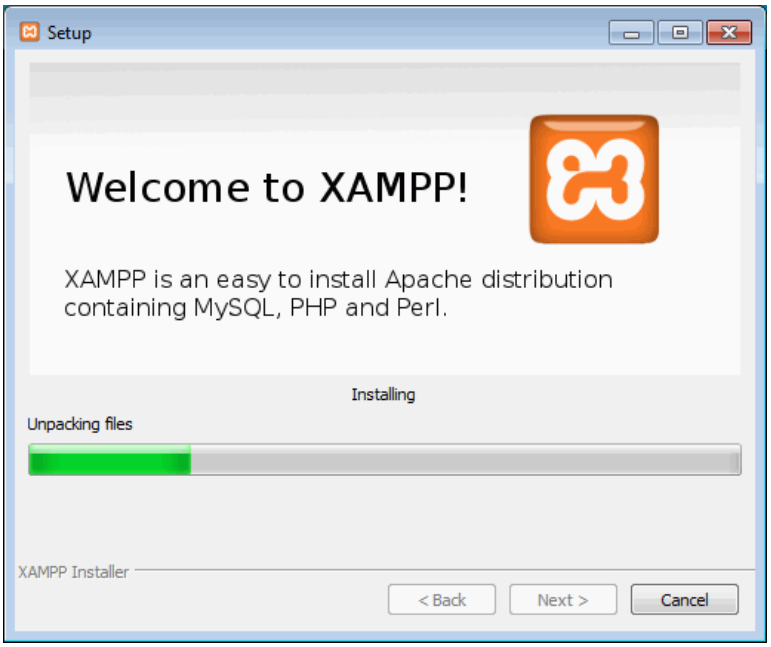
Lo siguiente es la inicialización del asistente de instalación. Para continuar, hay que hacer clic en el botón “Next”. Los componentes mínimos que instala XAMPP son el servidor Apache y el lenguaje PHP pero XAMPP también instala otros elementos, los cuales también nos harán falta como “phpMyAdmin” que nos ayudará a observar el estado y gestionar visualmente nuestra base de datos. Por defecto viene marcado todos los elementos y nosotros dejaremos marcados todos, tal y como viene (nunca se sabe cuándo podrás necesitarlo):



En la siguiente pantalla, se puede elegir la carpeta de instalación de XAMPP, de forma predeterminada es C:\xampp. Nosotros no lo cambiaremos a no ser que lo consideremos oportuno, una vez seleccionada le damos a siguiente:



Antes de terminar, nos aparece una pantalla con información sobre los instaladores creados por Bitnami, desmarcamos la casilla “Learn more” y hacemos clic en “Next”. A continuación para empezar la instalación, hay que hacer otra vez clic en “Next”. Ahora, deberemos esperar a que termine el proceso de copia de archivos:

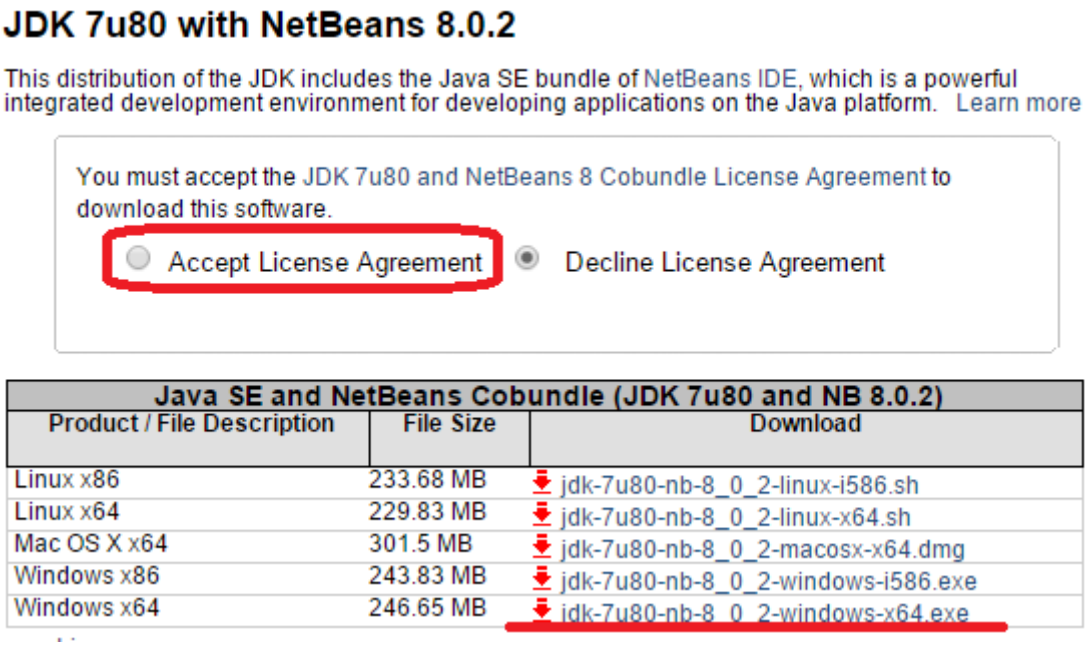


Para terminar, hay que hacer clic en el botón “Finish”, dejaremos marcado la casilla para abrir el panel de control de XAMPP.

2. Instalación NetBeans

Antes de empezar con la instalación tenemos que descargar el JDK junto al NetBeans 8.0.2. disponible en el siguiente enlace: http://www.oracle.com/technetwork/java/javase/downloads/jdk-7-netbeans-download-432126.html

Nosotros haremos el proceso para Windows(x64) por lo que descargamos el ejecutable correspondiente.



JDK 7u80 with NetBeans 8.0.2

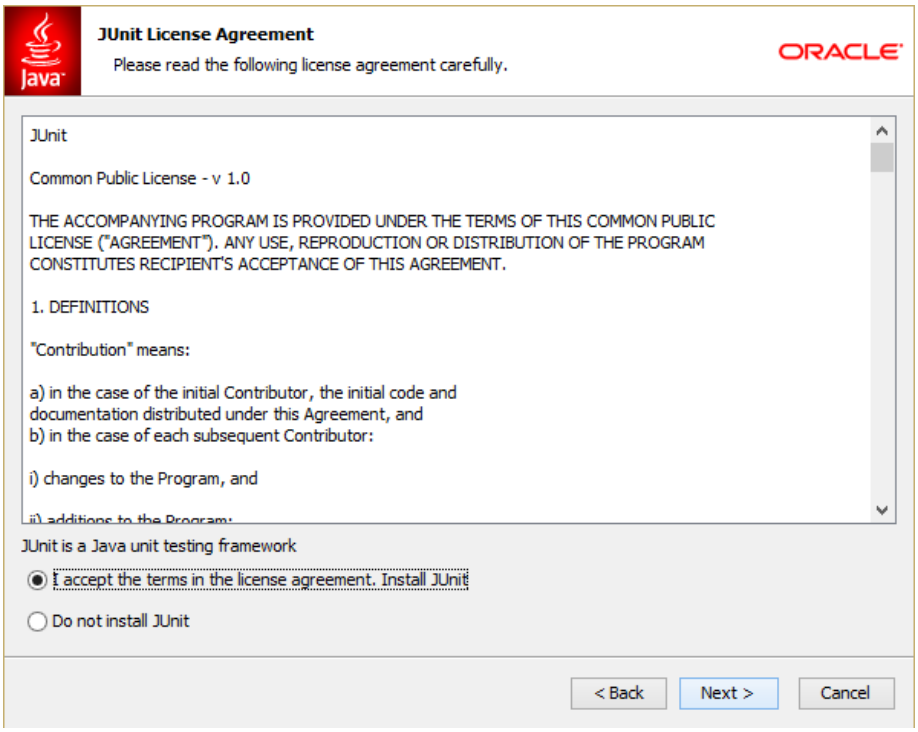
This distribution of the JDK includes the Java SE bundle of NetBeans IDE, which is a powerful integrated development environment for developing applications on the Java platform. Learn more

You must accept the JDK 7u80 and NetBeans 8 Cobundle License Agreement to download this software.

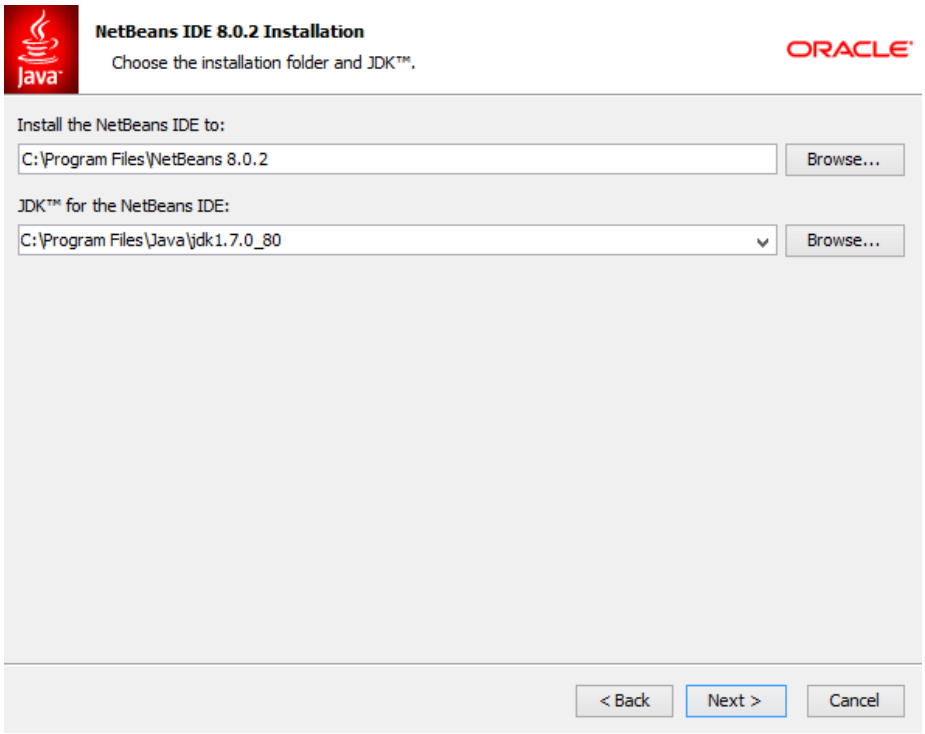
Accept License Agreement Decline License Agreement

Java SE and NetBeans Cobundle (JDK 7u80 and NB 8.0.2)		
Product / File Description	File Size	Download
Linux x86	233.68 MB	jdk-7u80-nb-8_0_2-linux-i586.sh
Linux x64	229.83 MB	jdk-7u80-nb-8_0_2-linux-x64.sh
Mac OS X x64	301.5 MB	jdk-7u80-nb-8_0_2-macosx-x64.dmg
Windows x86	243.83 MB	jdk-7u80-nb-8_0_2-windows-i586.exe
Windows x64	246.65 MB	jdk-7u80-nb-8_0_2-windows-x64.exe

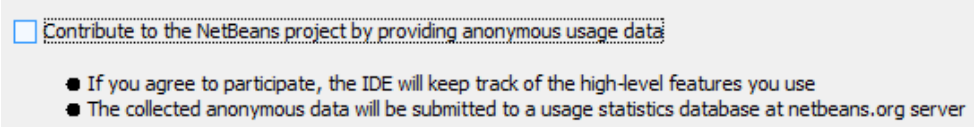
Una vez descargado, arrancamos el ejecutable y seguimos los pasos, destacando en la instalación los elementos descritos a continuación.
Aceptamos la licencia y pasamos al siguiente paso:



Seleccionamos la dirección en la que queremos que se instale tanto el JDK, como la dirección de la instalación del NetBeans y pasamos a la instalación.



Para terminar desmarcamos la casilla de contribución con NetBeans y finalizamos.



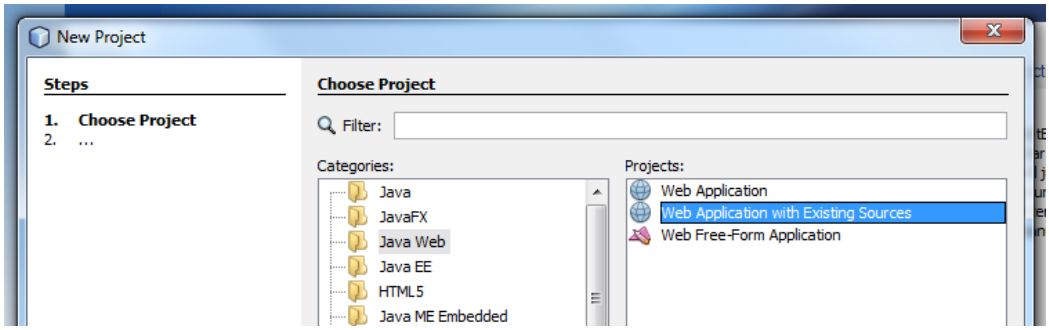
3. Pasos a seguir en la instalación.

Una vez instalado el software necesario para la ejecución del proyecto, queda crear la base de datos MySQL con XAMPP, crear un proyecto con el software proporcionado en NetBeans y por último, si se desea, importar los datos de prueba proporcionados en la BBDD creada.

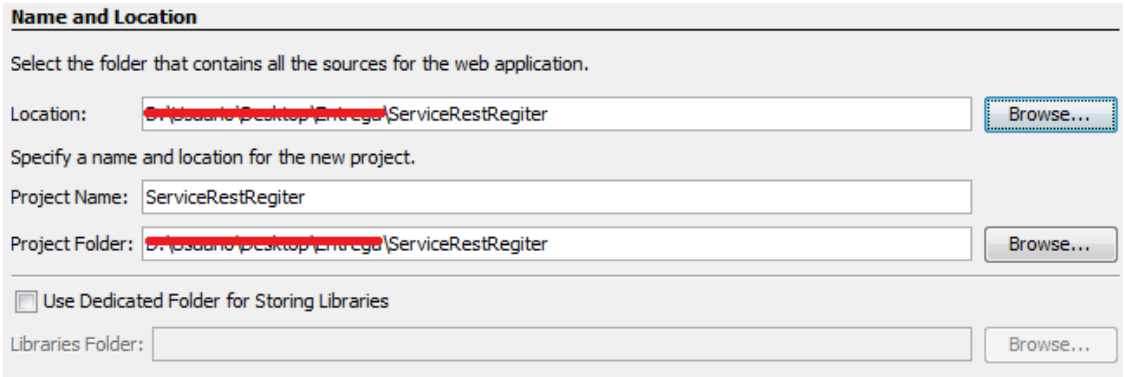
Por lo tanto, primero creamos la base de datos con el nombre “colaborativa”, para ello primero tenemos que arrancar en el panel de control de XAMPP el servidor “Apache” y “MySQL”

Ahora nos dirigimos al siguiente enlace http://localhost/phpmyadmin/server_databases.php donde crearemos la base de datos con la siguiente configuración:

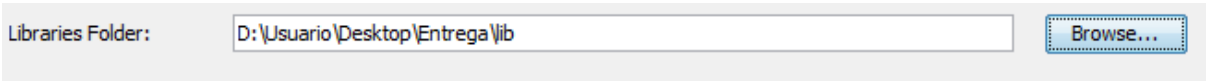
Ya podemos pasar a crear el proyecto en NetBeans, le damos a nuevo proyecto y seleccionamos en “Java Web” la opción de crear un proyecto con software existente (“Web Application with Existing Source”). Como se visualiza a continuación y le damos a siguiente.



A continuación seleccionamos la carpeta donde se encuentra el proyecto, aconsejamos mantener el mismo nombre dado que al acceder a las peticiones REST lo hará incluyendo éste nombre en la URL. Tal como mostramos en la imagen de abajo:



Le damos a siguientes hasta que lleguemos a la siguiente ventana, donde incluimos las librerías necesarias, proporcionada en la raíz junto al código, tras seleccionar la carpeta “lib” finalizamos:



Una vez creado ya podemos ejecutar el código y todo funciona correctamente pero, para poder observar correctamente todas las funcionalidades, se aconseja importar el archivo import.sql a la base de datos. Para ello nos dirigimos al siguiente enlace http://localhost/phpmyadmin/server_databases.php#PMAURL-8:db_import.php?db=colaborativa seleccionamos el archivo ya nombrado y le damos a continuar.

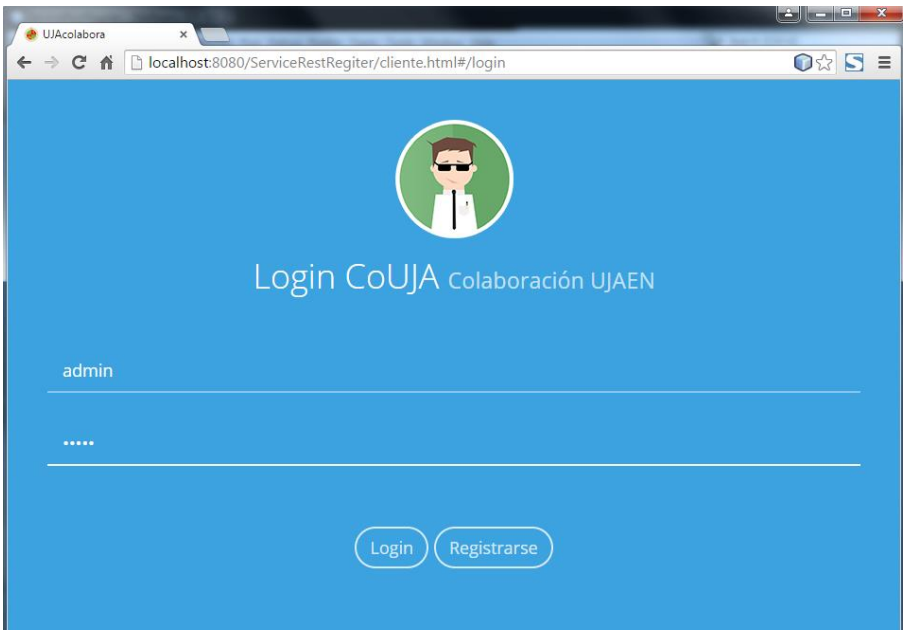
Apéndice II. Manual de Usuario

En este apartado comentaremos cómo realizar las funcionalidades disponibles del prototipo desarrollado. Dividiéndolo en diferentes apartados para una mejor organización del documento.

1. Acceso a la plataforma

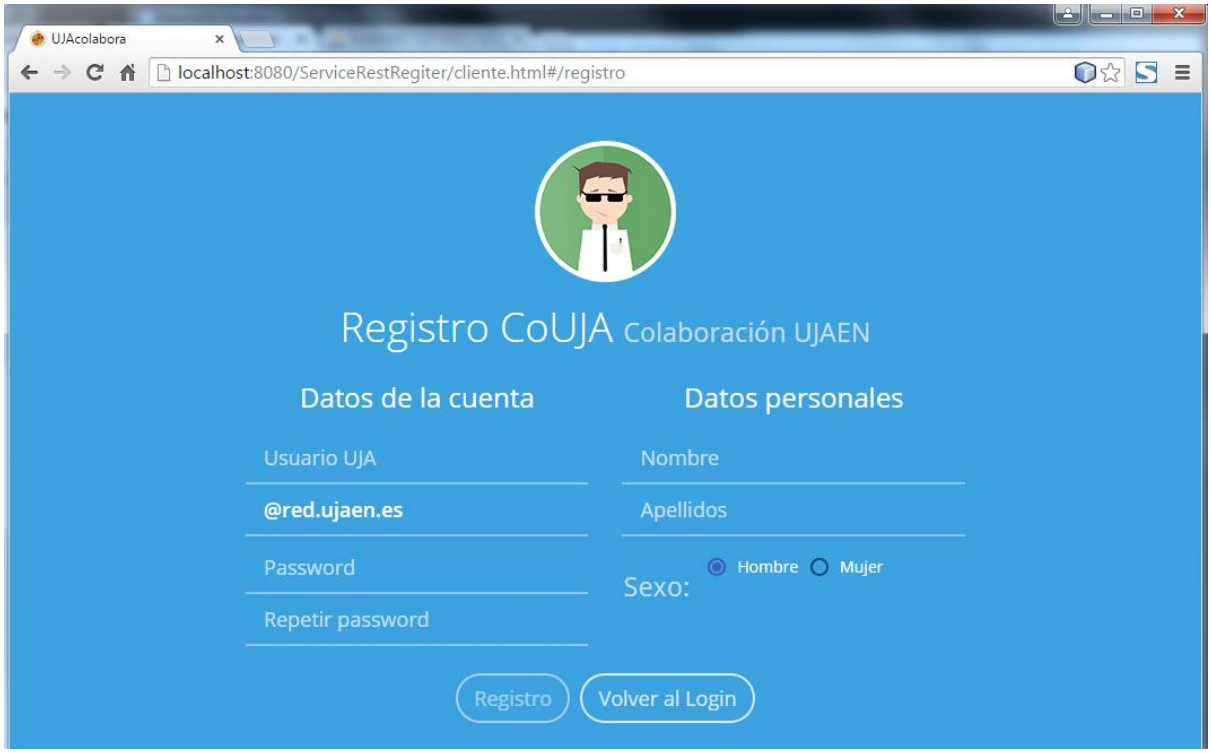
Nada más ejecutar el proyecto, nos aparece este simple hipertexto, donde nos muestra que el servicio está arrancado (para probarlo con otra aplicación cliente si se desea). Para acceder a la aplicación web cliente, hay que pulsar “Iniciar” al final del punto 1: “Probar con cliente AngularJS”.

La pantalla de acceso consiste simplemente en un formulario donde sólo tenemos que introducir el ***Usuario*** y la ***Contraseña***. Para enviar el formulario pulsamos el botón “Login”. La pantalla se muestra a continuación:



2. Registrar un nuevo usuario

A la pantalla de registro se llega pulsando el botón “Registro”, situada en la pantalla de acceso.



Como podemos observar también consiste en un formulario divididos en dos apartados: Datos de la cuenta, donde definiremos el usuario y la contraseña. Y datos personales, que como su nombre indica introducimos los datos personales básicos necesarios. Para enviar el formulario, tendremos que pulsar el botón “Registro”.

3. Navegar por la aplicación

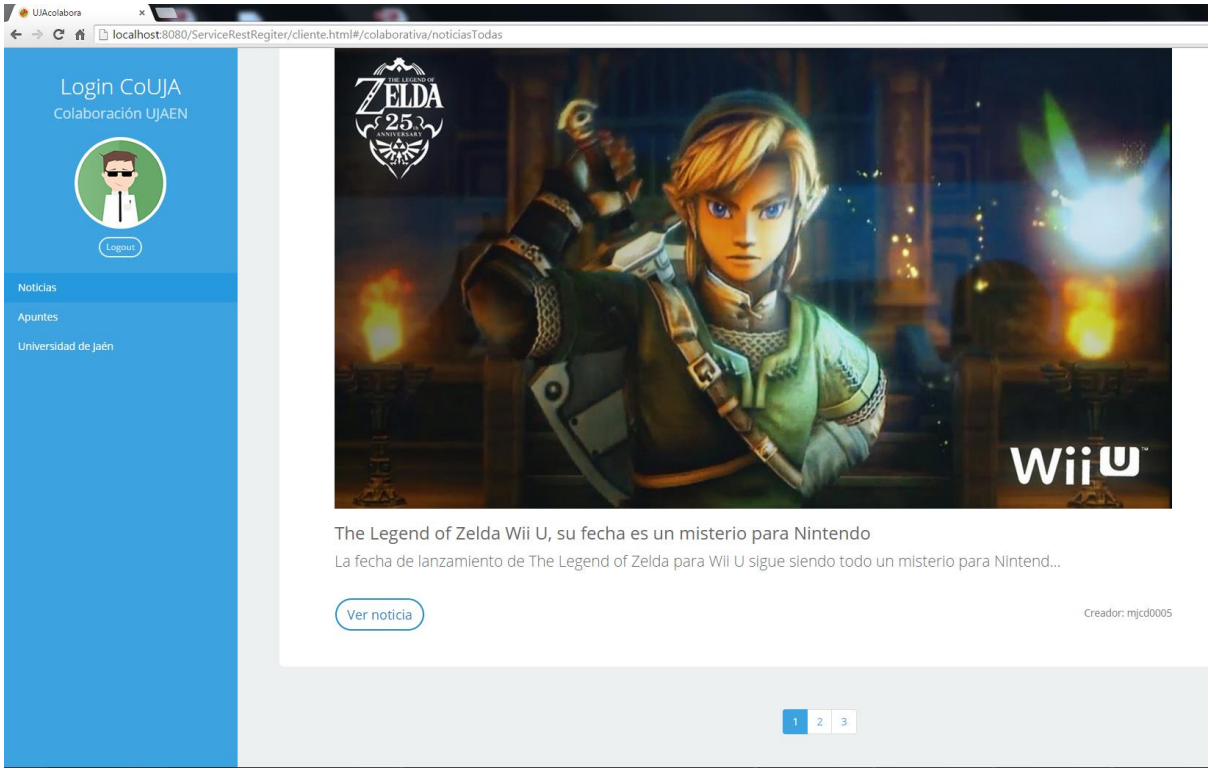
Una vez accede a la aplicación, se te muestra un menú vertical a la izquierda donde podrás salir de la aplicación pulsando el botón “Logout” o navegar por las diferentes funcionalidades. Las funcionalidades que están disponibles en el prototipo como podemos ver son: “Noticias” y “Apuntes”.

4. Funcionalidad de noticias

Es la que te aparece por defectos y, las operaciones disponibles te aparecen en la zona central, tras una breve descripción del tipo de colaboraciones que se hacen en éste apartado. Las operaciones disponibles son ver todas las noticias, crear una noticia o ver para editar y borrar tus noticias ya creadas.

TODAS LAS NOTICIAS

Consiste en un listado paginado donde mostramos todas las noticias, para abrir la noticia que nos interese simplemente pulsamos “Ver noticia”. Para movernos por el listado usaremos la paginación situada al final de la página.

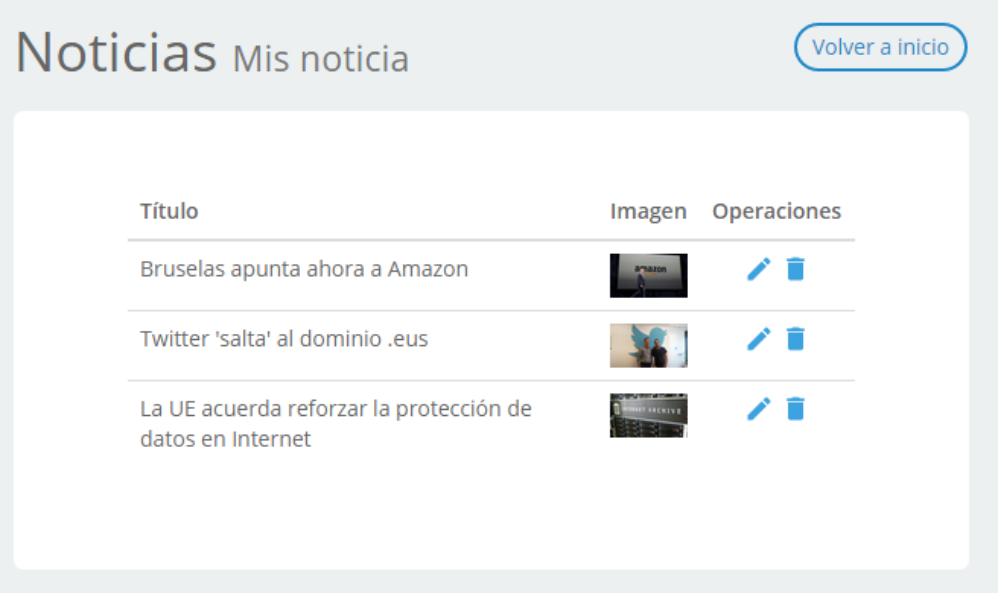


CREAR NOTICIA

Para crear una noticia tendremos que rellenar el formulario, donde introducimos el título, imagen de la noticia y la descripción. Tras completar todos los campos pulsamos el botón “Crear”.

MODIFICAR O ELIMINAR MIS NOTICIAS

En “Mis noticias” podremos ver una tabla con las operaciones disponible de editar (icono del lápiz) o eliminar (icono de la papelera).



Si queremos editar, nos llevará a un formulario donde podremos editar la información deseada. O bien si queremos eliminar, nos aparecerá un modal donde deberemos confirmar su borrado.

1. *Funcionalidad de apuntes*

Las operaciones disponibles te aparecen en la zona central, tras una breve descripción del tipo de colaboraciones que se hacen en éste apartado. Las operaciones disponibles son buscar apuntes, subir un nuevo apunte y ver tus apuntes subidos, pudiendo en éste último caso eliminarlos si se desea.

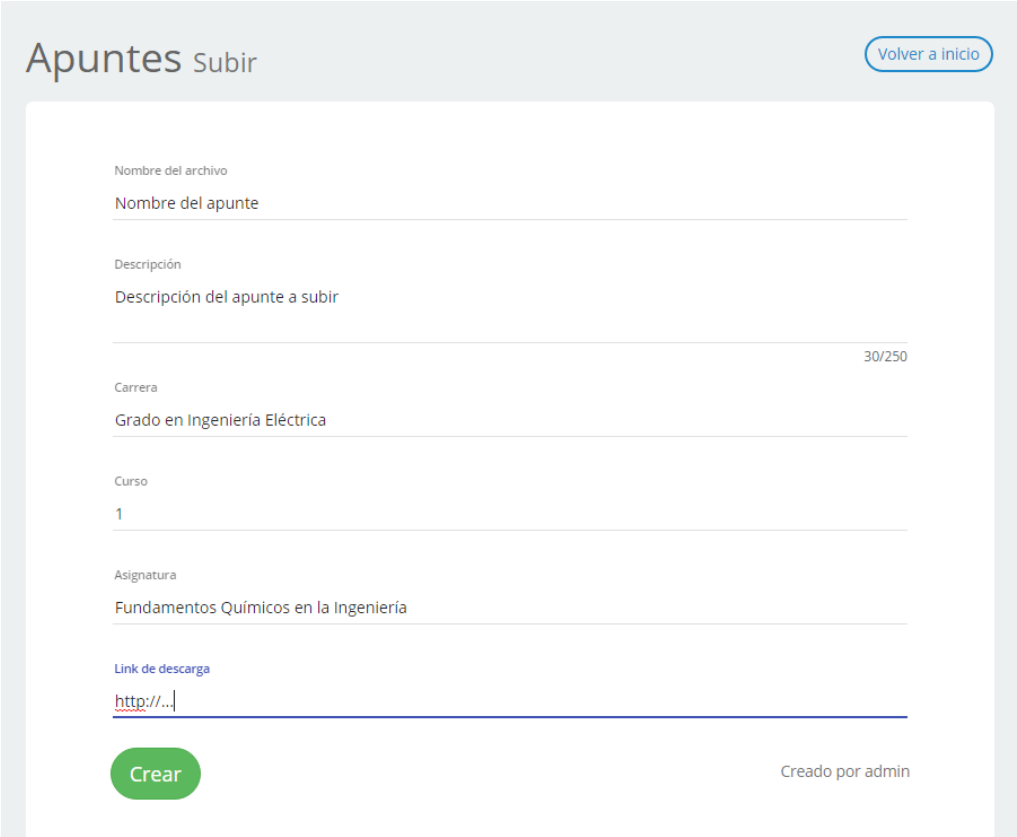
BUSCAR APUNTES

Tendremos que navegar entre las carreras disponibles, los cursos y por últimos las asignaturas para llegar a la lista de apuntes que corresponde. Como mostramos en la siguiente imagen:

En el ejemplo podemos ver dos apuntes para la asignatura de “Desarrollo de aplicaciones empresariales” del “Grado en Ingeniería Informática”. Las operaciones posibles con los apuntes son descargarlo directamente o ver más detalles sobre ellos como la fecha de subida.

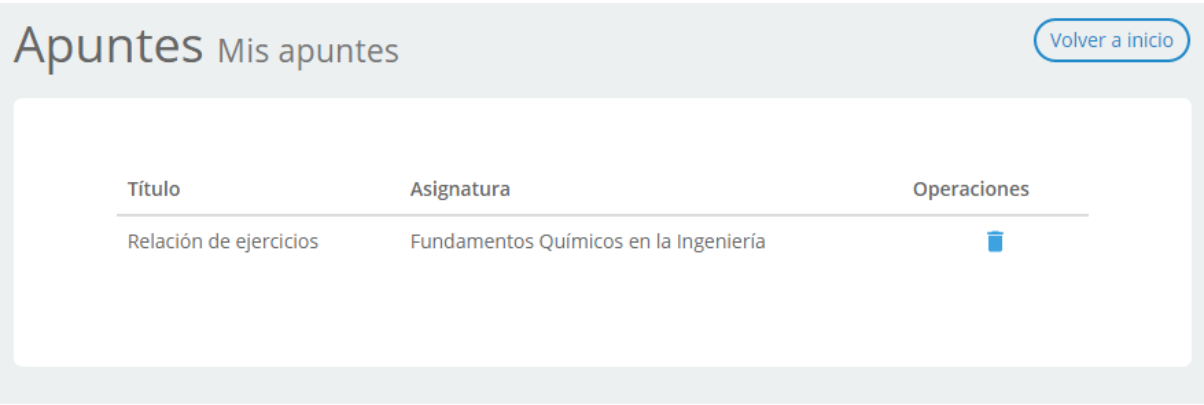
SUBIR UN APUNTE

De igual modo que al colaborar con una noticia, subir un apuntes consiste en rellenar un formulario, cuyos campos son: Nombre del archivo, Descripción (del archivo a subir), Carrera, Curso, Asignatura y por último Link de descarga (donde se podrá descargar el archivo).



TUS APUNTES

Encontramos una lista de los apuntes que hemos subido para colaborar, en esta lista ofrecemos la opción de poder eliminar y dejar de compartir un archivo dado. Mostramos la forma de listar esta información a continuación:



Apéndice III. Descripción de contenidos suministrados

En el cd suministrado se encuentran los siguientes contenidos:

1. ***Memoria.pdf:*** Memoria del proyecto.
2. ***ServiceRestRegiter:*** Carpeta donde se encuentra el código fuente para la construcción del proyecto.
3. ***lib:*** Carpeta con las librerías necesarias de nuestro proyecto.
4. ***import.sql:*** Archivo con los datos de pruebas a importar en la base de datos.
5. ***VideoFuncionalidad.mp4:*** Video donde se muestra las funcionalidades añadidas al prototipo, un ejemplo del software en ejecución.

NOTA: Para una correcta instalación del sistema leer apéndice I.