Aplicación Web para la Gestión de Memorias y Apuntes

*W*eb Application for Reports and Notes Management

Killian Jiménez Lecroc

Escuela Técnica Superior de Ingeniería Informática

Trabajo de Fin de Grado

La Laguna, 6 de septiembre de 2014

Dra. Dña. **Coromoto León Hernández**, con N.I.F. 78.605.216-W, y Dr. D. **Eduardo Manuel Segredo González**, con N.I.F. 78.564.242-Z, adscritos al área de Lenguajes y Sistemas Informáticos de la Universidad de La Laguna

C E R T I F I C A N

Que la presente memoria titulada:

"*Aplicación Web para la gestión de Memorias y Apuntes*"

ha sido realizada bajo su dirección por D. **Killian Jiménez Lecroc**, con N.I.F. 43.833.473-G.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 6 de septiembre de 2014

Agradecimientos

Quisiera agradecer a varias personas la ayuda que me han prestado en la realización de este Proyecto de Fin de Grado. Entre ellas, y en primer lugar, a mis directores, Coromoto León Hernández y Eduardo Manuel Segredo González por toda la confianza y paciencia invertida en mi.

También agradecer a Rodrigo por toda su ayuda, sin la cual me hubiera sido imposible acabar el proyecto a tiempo.

A mis amigos, Jhony, Borja, Juan Miguel y, en especial, a Sara. Nunca olvidaré toda la ayuda y el apoyo prestado durante mi carrera universitaria, además de todos los buenos momentos que hemos pasado todos juntos.

A Victor, por estar siempre ahí cuando más le necesité. Por hacerme reir constantemente, y por todos esos pequeños momentos que compartimos y que sin duda me han dado fuerzas en los momentos de más flaqueza.

Y finalmente agradecer a quienes han hecho posible que pueda estar hoy aqui, mis padres. Gracias mamá y gracias papá por vuestro constante apoyo y esfuerzo. Siempre lo he dicho, y siempre lo diré: sois los mejores padres que un hijo puede tener. Os quiero.

Resumen

En el proyecto se ha desarrollo una aplicación web para la gestión de apuntes, documentos y memorias para la Titulación de Ingeniería Informática, de forma que sólo pueda ser usada por alumnos, profesores, y demás personal administrativo de esta. La aplicación consta de varios roles, según los cuales un usuario podrá tener ciertos permisos para realizar algunas de las acciones de gestión, consulta, subida de archivo, y otros.

Lo que diferencia esta aplicación web de otras del mismo estilo, es que los apuntes, documentos y memorias subidos a la aplicación, pasarán por un proceso de revisión previo. Con esto lo que se consigue es construir algo similar a una biblioteca virtual con información verificada y contrastada.

Debido a la gran cantidad de archivos que debe manipular la aplicación, es necesario hacer uso de un repositorio para almacenarlos. Por ello, se utilizó el repositorio de un Sistema Gestor de Contenido Empresarial (ECM).

Debido a mi familiarización con el lenguaje, su facilidad de uso, y la grán cantidad de documentación online, el lenguaje de programación escogido para el proyecto ha sido Ruby, utilizando el framework para desarrollo de aplicaciones web Ruby on Rails. También se ha hecho uso de HTML5 y CSS3 para la implementación de la aplicación.

Palabras clave: Ruby, Ruby on Rails, HTML5, CSS3, Sistema Gestor de Contenido Empresarial, Aplicación Web, Roles, Apuntes, Memorias, Documentos

Abstract

In this project, it has been developed a Web application for managing notes, documents and memories for the ETSII, so that it can only be used by students, teachers and other administrative staff of it. The application has several roles, according to which a user may have certain permissions to perform some of the actions of management, consulting, file upload, and others.

What differentiates this application from other web application in the same style, is that the notes, documents and reports uploaded to the application will go through a preliminary review process. By doing this, we can build something similar to a virtual library but with verified and validated information.

Due to the large number of files that will be uploaded to the application, we will need to use a repository to store them. Therefore, we will make use of a repository from a Enterprise Content Manager (ECM).

Because of the ease of use, large quantity of web documentation, and mainly, because I am already familiar with it, the programming language chosen for the project has been Ruby, using the web development framework Ruby on Rails. HTML5 and CSS3 has also been used for the implementation of the application.

Keywords: *Ruby, Ruby on Rails, HTML5, CSS3, Enterprise Content Management, Web Application, Roles, Notes, Reports, Documents*

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Objetivo Principal

La gestión de contenido empresarial o Enterprise Content Management (ECM) son las estrategias, métodos y herramientas utilizadas para capturar, gestionar, almacenar, preservar y entregar contenido y documentos relacionados con los procesos organizativos.

El objetivo de este proyecto es la creación de una aplicación web en la cual los usuarios de la Escuela Técnica Superior de Ingeniería Informática de la Universidad de La Laguna puedan subir memorias, apuntes, trabajos y demás información útil para otros estudiantes, haciendo uso de un Sistema Gestor de Contenido Empresarial adaptado de forma conveniente. De esta forma los usuarios de la escuela podrán acceder a estos recursos, que habrán sido previamente verificados y que estarán etiquetados correctamente y colocados en la categoría correspondiente.

1.2. Licencia

Cuando se habla de licencia de Propiedad Intelectual en general, se hace referencia a un contrato por escrito a través del cual el autor ejerce los derechos de explotación de su obra especificando qué derechos exactamente cede a la otra parte, las modalidades de explotación que se ceden y el tiempo y el ámbito territorial de cesión.

1.2.1. Licencia Creative Commons

En el caso de las licencias Creative Commons se consideran más bien "licencias de adhesión" que operan como algo parecido a un aviso legal por su carácter genérico ya que la otra parte del contrato no está determinada. Sus principales características

son:

- Las licencias Creative Commons sirven para expresar los usos que permite el autor sobre su obra a los demás usuarios, no los usos que se dispone a hacer de su obra.
- Estas licencias se centran exclusivamente en el ejercicio de los derechos de explotación.
- Solo el autor puede decidir el uso de estas licencias.
- El autor ha de concentrar y mantener la titularidad en exclusiva de los derechos de explotación.
- Antes de seleccionar la jurisdicción de la licencia debemos asegurarnos que efectivamente como autores se nos aplica la ley de Propiedad Intelectual de la jurisdicción en cuestión.
- El uso de estas licencias no supone ningún tipo de registro de la Propiedad Intelectual.
- Se trata de las licencias Copyleft más desarrolladas y sofisticadas. El Copyleft permite ofrecer la posibilidad de copiar, difundir, modificar o incluso hacer un uso comercial de nuestra obra a un público genérico sin renunciar a ningún tipo de protección legal por ello.

1.2.2. Tipos de Licencia

Las licencias Creative Commons están compuestas por un módulo fijo más tres módulos variables con 6 combinaciones posibles a través de su selector de licencia.

- Módulo fijo:

 Atribución (BY): El reconocimiento de la autoría es un derecho moral irrenunciable por parte del autor y todas las licencias deben respetarlo y aplicarlo siempre.

- Módulos variables:

 Compartir Igual (SA): permite obras derivadas bajo exactamente la misma licencia o una similar (una licencia CC más actualizada o de otra jurisdicción).

No uso Comercial (NC): prohíbe que a obra sea utilizada con fines comerciales directos o indirectos (ej: hilo musical en un negocio).

No Obras Derivadas (ND): no permite modificar de forma alguna la obra. (ej. traducción de una obra literaria).

1.2.3. Licencia para la aplicación web

Para esta aplicación web se hará uso de una licencia Creative Commons. Para ello, primeramente se ha de acceder a la página web oficial de Creative Commons (http://creativecommons.org/) y rellenar el formulario:

- ¿Quiere permitir que se compartan las adaptaciones de su obra?: Sí, mientras se comparta de la misma manera.
- ¿Quiere permitir usos comerciales de su obra?: No.

Una vez contestado a las preguntas del cuestionario, se genera la licencia, quedando de la siguiente forma:

En los capítulos siguientes se abarcará el Análisis funcional y la especificación de requisitos, el Diseño técnico del Sistema, y el Desarrollo e implementación de la aplicación web.

Capítulo 2

Análisis funcional y especificación de requisitos

La Especificación de Requisitos Software (ERS) es una descripción completa del comportamiento del Sistema Informático que se va a desarrollar. En él se incluye un conjunto de casos de uso, así como una serie de requisitos funcionales que imponen ciertas restricciones en el diseño e implementación.

Este capítulo está enfocado a realizar un análisis funcional y a establecer los requisitos mínimos que debe cumplir la aplicación web de gestión de apuntes.

2.1. Definición de los distintos roles

Tras distintas reuniones con los clientes del proyecto se ha decidido definir cinco roles de usuarios. Cabe tener en cuenta que un usuario puede tener dos o más roles.

Rol de Consultor: Un usuario que tenga rol de Consultor, deberá iniciar sesión con su identificador de la Escuela Técnica Superior de Ingeniería Informática de la Universidad de La Laguna para poder acceder a los diversos materiales administrados por el Sistema Gestor de Contenido. Este usuario sólo puede ver y descargar documentos de la aplicación web.

Rol de Contribuyente: Al igual que el Consultor, el usuario deberá identificarse también con su identificador y podrá acceder al mismo contenido que un usuario con rol de Consultor, con la única diferencia de además poder realizar subidas de archivos al Sistema Gestor de Contenido. Por defecto, todo usuario es Consultor y Contribuyente a la vez, pero un gestor podría quitarle este último privilegio a un usuario si intentase subir contenido no adecuado a la aplicación web.

Una vez subido los apuntes, estos no serán visibles a los demás usuarios hasta pasar por un proceso de revisión.

Rol de Revisor: Un usuario con rol de Revisor deberá autenticarse en la aplicación para poder revisar y validar los apuntes que se le ha sido asignado. Esta asignación será realizada por un usuario con rol de Gestor. Un usuario con rol de Revisor, tras la revisión del documento, puede aceptarlo o denegarlo. En el caso de ocurrir esto último, deberá argumentar sus razones en un campo de texto.

El usuario con rol de Contribuyente que ha subido el archivo deberá corregir lo expuesto por el revisor y volver a realizar una subida del nuevo archivo.

Rol de Gestor: Se encargarán de asignar los documentos subidos a revisores. Un documento puede tener varios revisores, y no será visible hasta que todos los revisores de dicho documento lo den por válido. El usuario con rol de Gestor será el encargado de publicar el documento una vez este haya pasado por el proceso de revisión.

Rol de Administrador: Tendrá los permisos correspondientes para la administración de la totalidad de la aplicación web. Entre sus principales tareas se encuentran las de administrar las categorías y los usuarios gestores.

Las tareas o actividades que pueden realizar cada uno de los roles definidos se relacionan en la (Tabla 2.1).

2.2. Mejoras que supondría la implantación de esta aplicación web

Entre las principales ventajas de la implementación de esta aplicación web, cabe destacar las siguientes:

- **Disponibilidad:** Se podrá acceder al material las 24 horas del día.
- Concesión de grandes ventajas para **personas de movilidad reducida**, ya que podrán realizar sus consultas y acceder al material a distancia.
- **Fiabilidad:** El material será revisado y contrastado por un encargado de realizar esta función (comúnmente un profesor de la escuela).
- **Flexibilidad:** Permite elegir entre los diferentes canales que se ofertan para la obtención del material que se busca.
- **Eco-responsabilidad:** El uso de las Tecnologías de la Información y la Comunicación permite ahorrar recursos energéticos, así como el consumo de papel.

Rol	Tareas
Consultor	▪ Debe iniciar sesión con su usuario correspondiente. ▪ Puede consultar los diferentes apuntes y documentos subidos a la aplicación web.
Contribuyente	▪ Debe iniciar sesión con su usuario correspondiente. ▪ Puede consultar los diferentes apuntes y documentos subidos a la aplicación web. ▪ Puede subir sus propios apuntes y otros documentos a la aplicación.
Revisor	▪ Debe iniciar sesión con su usuario correspondiente. ▪ Su función principal es corregir, contrastar y validar los distintos archivos subidos por los usuarios a la aplicación.
Gestor	▪ Debe iniciar sesión con su usuario correspondiente. ▪ Puede banear un usuario quitándole el rol de Contribuyente. ▪ Su función principal es asignar revisores a los distintos documentos subidos por otros usuarios.
Administrador	▪ Debe iniciar sesión con su usuario correspondiente. ▪ Su función principal es administrar la aplicación web, principalmente las categorías y documentos.

Tabla 2.1: Tareas de cada rol

2.3. Casos de Uso

En un primer lugar, se tratarán los casos de uso de los consultores y de los contribuyentes.

Un Consultor podrá iniciar sesión, consultar apuntes y descargarlos. Un Contribuyente podrá además, subir documentos a la aplicación web.

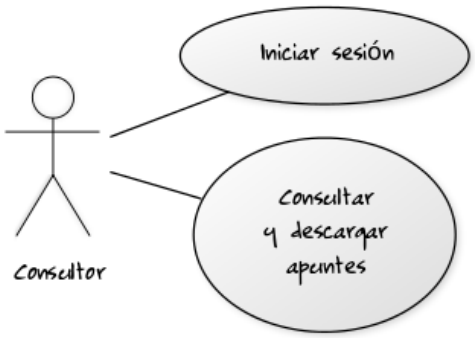

Figura 2.1: Caso de uso Rol de Consultor

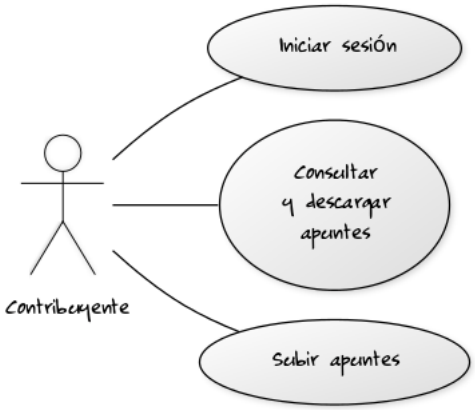

Figura 2.2: Caso de uso Rol de Contribuyente

Un revisor puede iniciar sesión, consultar apuntes y validarlos o denegarlos. En el caso de denegar un documento se deberá exponer las razones para que el usuario Contribuyente realice las modificaciones oportunas.

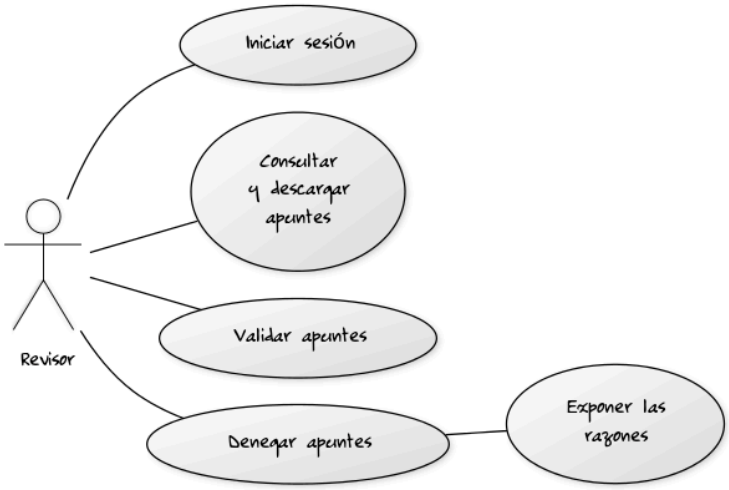

Figura 2.3: Caso de uso Rol de Revisor

El usuario gestor se encarga de asignar los revisores a una determinada tarea subida por un contribuyente. Una vez todos los revisores de una tarea hayan validado esta, el gestor deberá publicar

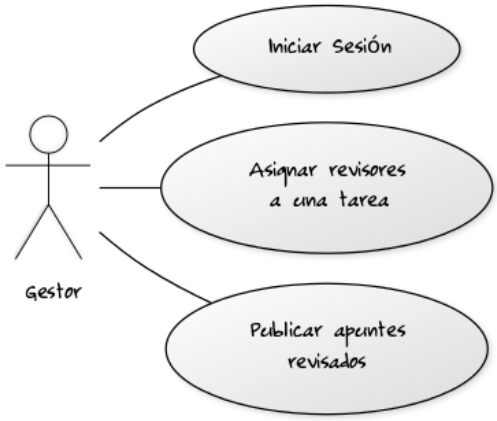

Figura 2.4: Caso de uso Rol de Gestor

El administrador se encargará de administrar la aplicación web, realizando acciones de administración a las categorías y los documentos.

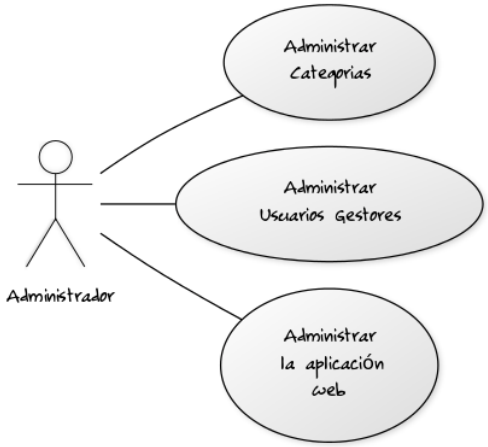

Figura 2.5: Caso de uso Rol de Administrador

2.4. Especificación de requisitos.

2.4.1. Requisitos Funcionales

- El usuario debe iniciar sesión para poder acceder a la aplicación.
- El acceso estará limitado a los usuarios de la Escuela Técnica Superior de Ingeniería Informática de la Universidad de La Laguna.
- Sólo los usuarios con derecho de administrador podrán acceder a las funciones administrativas.
- Los usuarios podrán examinar y descargar apuntes una vez hayan iniciado sesión en la aplicación web.
- Sólo los gestores y administradores podrán modificar y/o eliminar contenido del Sistema Gestor de Contenido.
- Los usuarios contribuyentes podrán subir documentos a la aplicación.
- Una tarea puede pertenecer a varias categorias.
- Los revisores contrastarán los documentos y los validarán si están correctos.
- Un usuario con rol de Gestor podrá asignar varios revisores a una tarea.

- Los consultores no podrán ver los documentos subidos hasta que un revisor los valide y el gestor los publique.
- El Sistema Gestor de Contenido debe soportar una gran cantidad de archivos.
- A la aplicación web se podrá subir documentos Word, txt, pdf, imágenes JPG y PNG, y presentaciones PowerPoint.
- Un usuario con rol de contribuyente podrá ser baneado por un usuario Gestor.

Capítulo 3

Diseño técnico del Sistema de Gestión de Apuntes

El diseño es el proceso de transformación del problema en una solución. La solución será la que satisface todos los requerimientos planteados en la especificación de requisitos del análisis funcional.

En este capítulo se definirá el diseño técnico del sistema de gestión de archivos, explicando la arquitectura de software y los objetivos que debe cumplir. Se definirá el diseño de la base de datos del sistema, se explicaran cuáles serán las tecnologías usadas para desarrollar e implementar la aplicación web y se propondrá un prototipo del posible diseño del sistema.

3.1. Arquitectura del Software

La Arquitectura del Software es el diseño de más alto nivel de la estructura de un sistema. El objetivo principal es realizar una aplicación web en la que los usuarios puedan acceder a un servidor a través de su propio navegador. Este servidor tendrá instalado un gestor documental.

El Navegador Web se considera como la primera capa de la estructura de una aplicación web. El Motor capaz de usar las tecnologías de web dinámica (como por ejemplo: Ruby on Rails, PHP, etc.) constituye la capa intermedia. Por último, la Base de Datos conforma la tercera y última capa.

El funcionamiento es como sigue: el Navegador Web manda peticiones a la capa intermedia, la cual ofrece servicios valiéndose de consultas y actualizaciones a la Base de Datos, y que a su vez proporciona una Interfaz de Usuario.

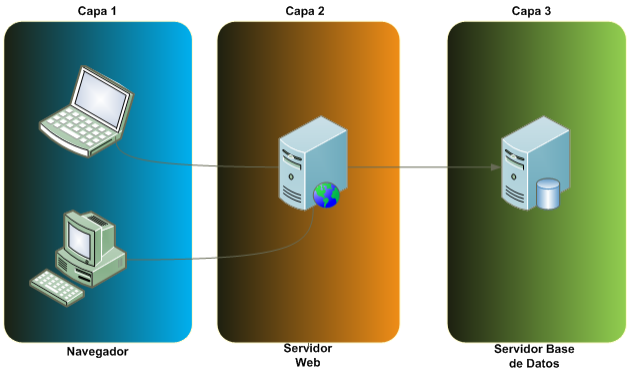

Figura 3.1: Estructura de las aplicaciones web

3.2. Características de la arquitectura

A continuación se enumeran las características más deseables para la arquitectura software diseñada.

- **Corrección**. El diseño se ajusta correctamente a los requerimientos dados.
- **Viabilidad**. Este diseño puede ser implementado y probado con las cantidades de tiempo y esfuerzo planeadas.
- **Comprensibilidad**. Los desarrolladores pueden entender este diseño e implementarlo correctamente.
- **Facilidad de acceso**. El código de la aplicación para acceder a los datos almacenados es sencillo.
- **Capacidad de Datos**. El sistema puede almacenar la cantidad de información necesaria.
- **Seguridad en los Datos**. Protección de los datos sensibles del usuario o de la empresa a los accesos no autorizados o a modificación.
- **Prevención de Intrusos**. Prevenir, por ejemplo, que "hackers" puedan abrir una terminal de comandos en nuestro servidor.
- **Soporte a Tareas y Eficiencia**. La interfaz del usuario encaja con las tareas que el usuario realizará y puede ser usada con un número razonable de clics.
- **Seguridad**. Los usuarios no podrán ser capaces de producir un resultado no deseado (por ejemplo, borrar información de la base de datos).

- **Consistencia y Familiaridad**. Los usuarios podrán aplicar sus conocimientos de interfaces similares o interfaces estándar a este sistema.

3.3. Modelo de Tareas

En el modelo de tareas se especifican cuáles serán los roles de usuarios capaces de manejar la aplicación a desarrollar así como el tipo de tareas que realizará cada rol de usuario.

3.3.1. Tipos de usuarios y sus responsabilidades.

El sistema será capaz de identificar cinco roles de usuarios.

- Consultor:
  - Inicia sesión.
  - Consulta apuntes.
  - Descarga ficheros.
- Contribuyente:
  - Inicia sesión.
  - Consulta apuntes.
  - Descarga ficheros.
  - Contribuye subiendo apuntes.
- Revisor:
  - Inicia sesión.
  - Consulta apuntes.
  - Descarga ficheros.
  - Valida documentos.
  - Deniega documentos.
  - Argumenta su denegación.
- Gestor:
  - Inicia sesión.
  - Consulta apuntes.

    - Descarga ficheros.
    - Asigna usuarios revisores a una tarea.
    - Publicar apuntes.

- Administrador:

    - Inicia sesión.
    - Consulta apuntes.
    - Descarga ficheros.
    - Puede subir apuntes.
    - Administra categorías.
    - Asigna usuarios con rol de Gestor.

3.4. Diseño de la Base de Datos

Una Base de Datos es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso. El primer paso para crear una Base de Datos, es planificar el tipo de información que se quiere almacenar en la misma, teniendo en cuenta dos aspectos: la información disponible y la información que necesitamos, asumiendo pues que el objetivo fundamental del diseño de bases de datos es obtener un conjunto de datos y un conjunto de operaciones sobre ellos, que permitan satisfacer las necesidades de la organización.

3.4.1. Descripción de las entidades

Los conceptos que reflejan los datos que le interesan a la aplicación, se representan mediante entidades. Una entidad es un objeto real o abstracto del que se quiere obtener una información.

3.4.1.1. Nombres de las entidades

- Usuario.

    - Consultor.
    - Contribuyente.
    - Revisor.
    - Gestor.
    - Administrador.

- Archivo.
    - Imágenes.
    - Texto Plano.
    - Documento pdf.
    - Enlaces.
- Categoría.
    - Niveles
- Solicitud.

3.4.1.2. Breve descripción

- La entidad **Usuario** se encarga de almacenar todos los usuarios que pueden hacer uso de la aplicación. Los usuarios pueden cumplir varios tipos de roles:
    - Rol de administrador.
    - Rol de contribuyente.
    - Rol de consultor.
    - Rol de gestor.
    - Rol de revisor.
- La entidad **Archivo** se encarga de guardar los datos referentes a los apuntes que se quieran subir al gestor documental, ya sea imágenes, texto plano, un enlace o un pdf.
- La entidad **Categoría** será utilizada para la organización de los documentos en diferentes niveles de dificultad.
- La entidad **Solicitud** almacenará los datos correspondientes a las peticiones de subidas de archivo.

3.4.1.3. Atributos

Cada entidad tiene asociados unos atributos que son las características o propiedades de aquello que representa la entidad. Dando valores a estos atributos, se obtienen las diferentes ocurrencias de una entidad. Existen dos tipos de atributos:

- **Identificador de entidad:** son atributos que identifican de manera unívoca cada ocurrencia de una entidad. Siempre debe existir, al menos, un atributo identificador. En este caso los atributos identificadores se distinguen por estar subrayados.
- **Descriptores de entidad:** son atributos que muestran una característica de la entidad.

Para cada una de las tablas de la base de datos, se necesitarán estos identificadores y descriptores:

USUARIO {ID_USUARIO, NOMBRE, APELLIDOS, CORREO_ELECTRÓNICO, CONTRASEÑA, ROLES}

Puesto que todo alumno, profesor, y personal administrativo de la Escuela Técnica Superior de Ingeniería Informática tiene un identificador de usuario (ya sea el alu para los alumnos, u otro identificador para el profesorado y demás personal administrativo), éste será el identificador de entidad (o clave primaria) para cada uno de los usuarios. Además también constarán de varios descriptores de entidad tales como nombre, apellidos, correo electrónico, y los roles a los que pertenece este usuario.

ARCHIVO {ID_ARCHIVO, ID_USUARIO, ID_CATEGORÍA, ID_REVISOR, TITULO, DESCRIPCIÓN, RUTA, VISIBLE}

La entidad Archivo se identificará por su ID, por lo que esta será la clave primaria. El ID_USUARIO se corresponderá al identificador del usuario que haya subido el archivo. También existirá un atributo *visible* que controlará que un archivo sea visible a los demás usuarios o no. El ID_CATEGORÍA se corresponderá al identificador de las categorías a las que pertenece el archivo. Por último, también se encuentra ID_REVISOR, que almacenará el ID de los usuarios revisores.

CATEGORÍA {ID_CATEGORIA, NOMBRE, DESCRIPCION }

La entidad Categoría se identificará, al igual que los anteriores, por su ID y almacenará datos tales como nombre de la categoría y descripcion de la misma.

SOLICITUD {ID_SOLICITUD, ID_ARCHIVO, ESTADO }

La clave primaria de una solicitud será el ID_SOLICITUD. Puesto que una solicitud está asociada directamente a un archivo, será necesario incluír el ID de este último y, además, el ESTADO de la solicitud.

3.4.2. Descripción de las Relaciones

Mediante las relaciones, se representan las asociaciones que se establecen entre los elementos del mundo real. En el Modelo de Datos, se traducen en relaciones entre entidades. Con ellas se pretende completar la representación que se tiene de la realidad.

3.4.2.1. Nombres de las relaciones

- Tiene
- Realiza
- Sube
- Publica
- Pertenece

3.4.2.2. Breve descripción

- Un archivo puede pertenecer a n categorías, y una categoría puede pertenecer a n archivos.
- Una solicitud pertenece a un archivo, y un archivo pertenece a una solicitud.
- Un revisor puede revisar n archivos, y un archivo puede ser revisado por n revisores.
- Un usuario con rol de gestor puede publicar n archivos, y un archivo es publicado por un usuario gestor.
- Un contribuyente puede realizar n solicitudes, y una solicitud es realizada por un contribuyente.
- Un archivo tiene una solicitud, y una solicitud pertenece a un archivo.
- Un contribuyente puede subir n archivos, y un archivo es subido por un contribuyente.

3.4.2.3. Grado de la relación

El grado de una relación es el número de entidades que participan en la relación. (Ver Figura 3.2)

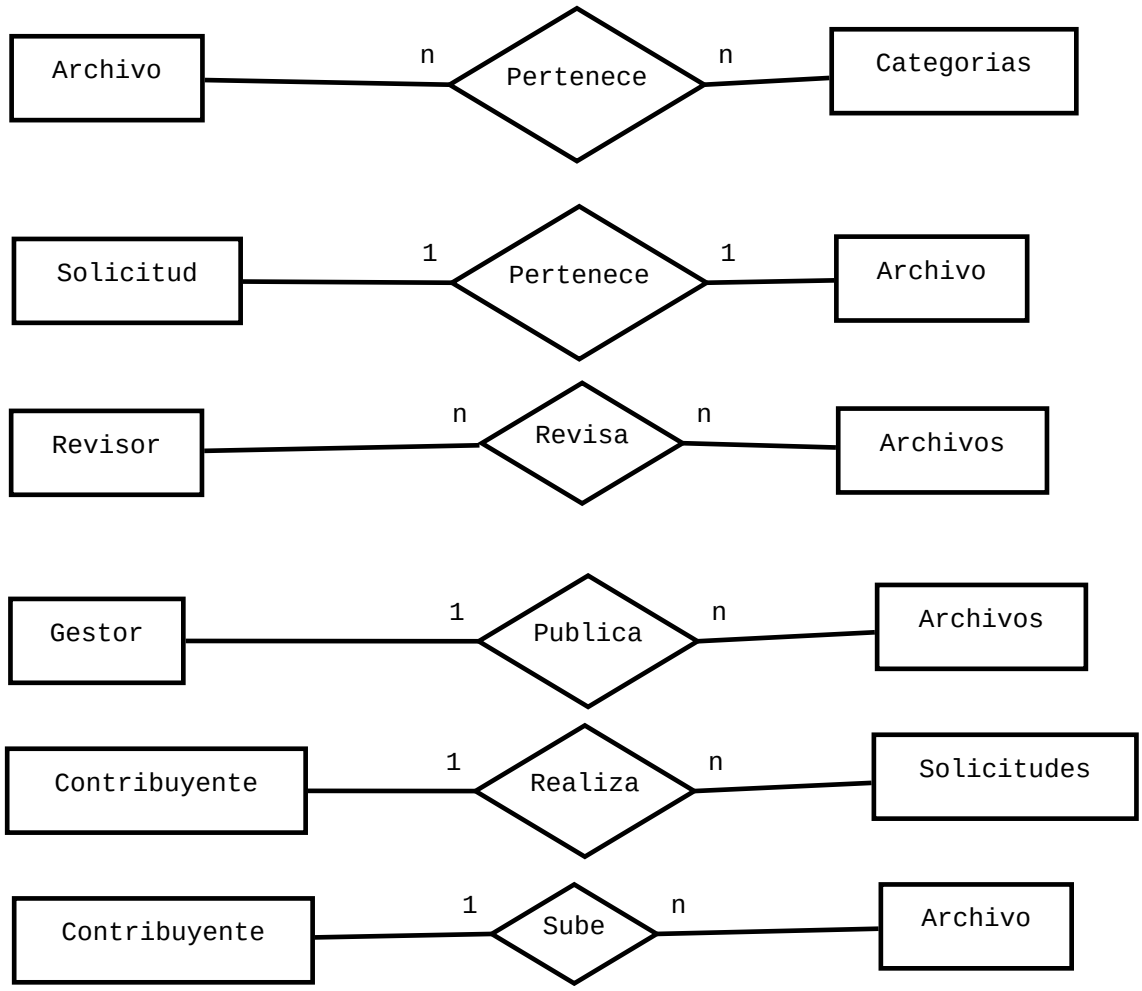

Figura 3.2: Grados del Diagrama Entidad Relación

3.4.3. Diagrama Entidad Relación

El Diagrama Entidad/Relación es la técnica de análisis y especificación de datos más ampliamente utilizada. Se ubica en el plano conceptual, obteniendo una representación de la realidad que sólo dependa de las características del problema.
Utiliza una serie de símbolos y reglas para representar los elementos (información) que forman parte del problema y las relaciones entre ellos. Esta representación gráfi-

ca facilita la comunicación con el usuario, por un lado y con el diseñador, por otro. (Ver Figura 3.3)

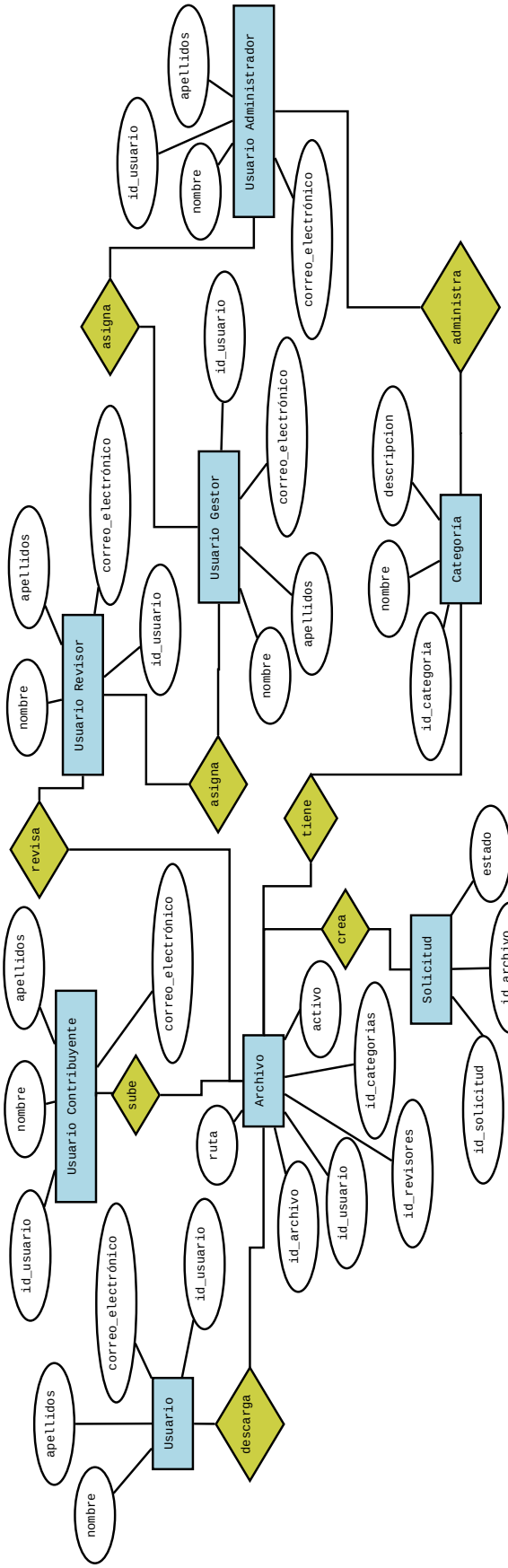

Figura 3.3: Diagrama Entidad Relación

3.4.4. Modelo Relacional

El Modelo Relacional consiste en un conjunto de una o más tablas estructuradas en registros (líneas) y campos (columnas), que se vinculan entre sí por un campo en común, en ambos casos posee las mismas características y generalmente se le denomina ID, identificador o clave. (Ver Figura 3.4)

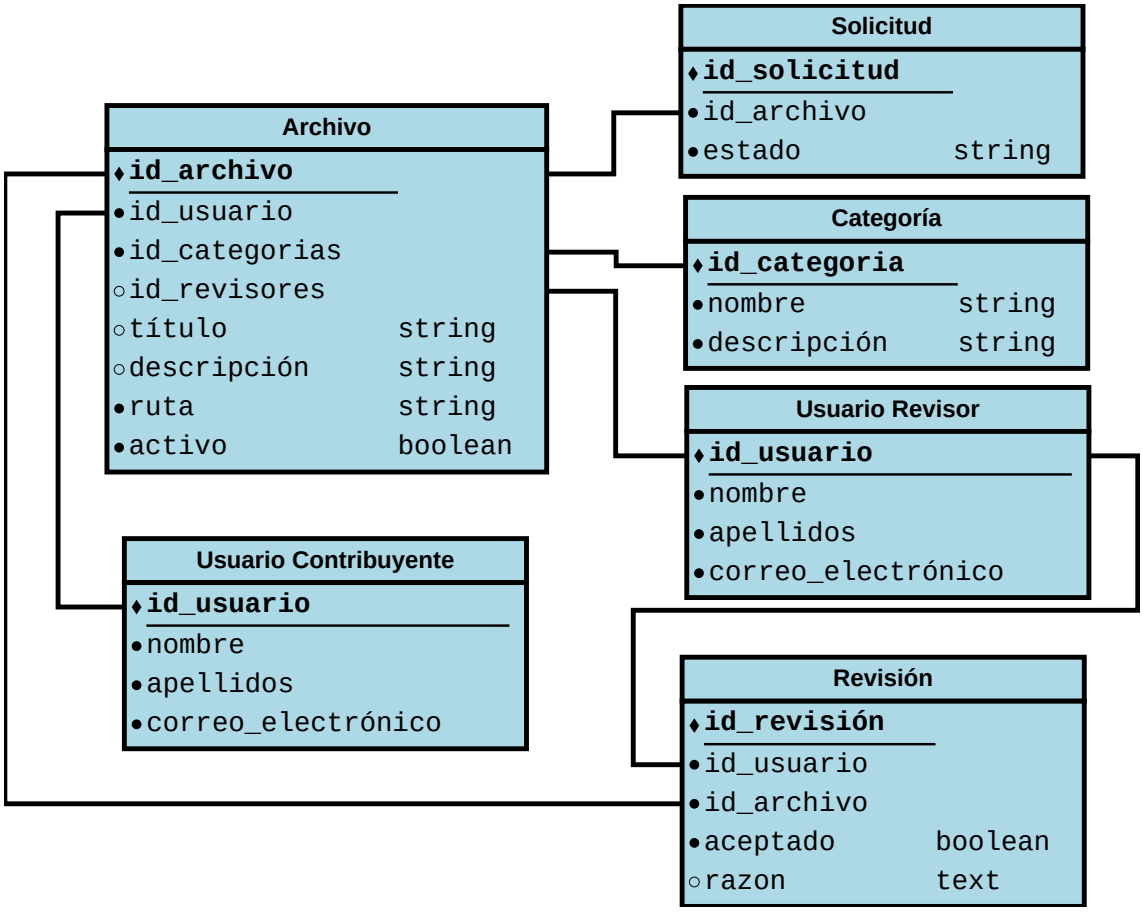

Figura 3.4: Modelo Relacional

3.5. Tecnología usada para el desarrollo de la aplicación web

Para el desarrollo del proyecto se han utilizado los siguientes lenguajes:

- Framework Ruby On Rails

- HTML5
- CSS3 (SASS)

Además se ha utilizado una base de datos SQL (SQLite3) y Alfresco como repositorio.

3.5.1. Ruby on Rails

Para el desarrollo de la aplicación se ha utilizado Ruby 2.1.1 y Rails 4.0

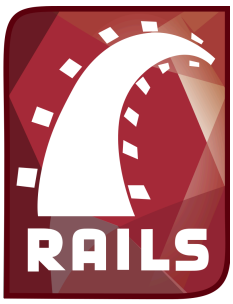

Figura 3.5: Ruby on Rails

Ruby on Rails, también conocido como RoR o directamente Rails es un framework de aplicaciones web de código abierto escrito en el lenguaje de programación Ruby, sigue el paradigma de la arquitectura Modelo Vista Controlador (MVC). Trata de combinar la simplicidad con la posibilidad de desarrollar aplicaciones del mundo real escribiendo menos código que con otros frameworks y con un mínimo de configuración. El lenguaje de programación Ruby permite la metaprogramación, de la cual Rails hace uso, lo que resulta en una sintaxis que muchos de sus usuarios encuentran muy legible. Rails se distribuye a través de RubyGems, que es el formato oficial de paquete y canal de distribución de bibliotecas y aplicaciones Ruby.

Los principios fundamentales de Ruby on Rails incluyen No te repitas (del inglés Don't repeat yourself, DRY) y Convención sobre configuración. No te repitas significa que las definiciones deberían hacerse una sola vez. Dado que Ruby on Rails es un framework de pila completa, los componentes están integrados de manera que no hace falta establecer puentes entre ellos. Por ejemplo, en ActiveRecord, las definiciones de las clases no necesitan especificar los nombres de las columnas, Ruby puede averiguarlos a partir de la propia base de datos, de forma que definirlos tanto en el código como en el programa sería redundante.

Convención sobre configuración significa que el programador sólo necesita definir aquella configuración que no es convencional. Por ejemplo, si hay una clase Historia en el modelo, la tabla correspondiente de la base de datos es historias, pero si la tabla no sigue la convención (por ejemplo blogposts) debe ser especificada manualmente (set_table_name "blogposts"). Así, cuando se diseña una aplicación partiendo de cero sin una base de datos preexistente, el seguir las convenciones de Rails significa usar menos código (aunque el comportamiento puede ser configurado si el sistema debe ser compatible con un sistema heredado anterior).

En las aplicaciones web orientadas a objetos sobre bases de datos, el Modelo consiste en las clases que representan a las tablas de la base de datos.

En Ruby on Rails, las clases del Modelo son gestionadas por ActiveRecord. Por lo general, lo único que tiene que hacer el programador es heredar de la clase ActiveRecord::Base, y el programa averiguará automáticamente qué tabla usar y qué columnas tiene.

En MVC, Vista es la lógica de visualización, o cómo se muestran los datos de las clases del Controlador. Con frecuencia en las aplicaciones web la vista consiste en una cantidad mínima de código incluido en HTML.

Existen en la actualidad muchas maneras de gestionar las vistas. El método que se emplea en Rails por defecto es usar Ruby Empotrado (archivos.rhtml, desde la versión 2.x en adelante de RoR archivos.html.erb), que son básicamente fragmentos de código HTML con algo de código en Ruby, siguiendo una sintaxis similar a JSP. También pueden construirse vistas en HTML y XML con Builder o usando el sistema de plantillas Liquid.

Es necesario escribir un pequeño fragmento de código en HTML para cada método del controlador que necesita mostrar información al usuario. El "maquetado" o distribución de los elementos de la página se describe separadamente de la acción del controlador y los fragmentos pueden invocarse unos a otros.

En MVC, las clases del controlador responden a la interacción del usuario e invocan a la lógica de la aplicación, que a su vez manipula los datos de las clases del Modelo y muestra los resultados usando las Vistas. En las aplicaciones web basadas en MVC, los métodos del controlador son invocados por el usuario usando el navegador web.

La implementación del Controlador es manejada por el ActionPack de Rails, que contiene la clase ApplicationController. Una aplicación Rails simplemente hereda de esta clase y define las acciones necesarias como métodos, que pueden ser invocados desde la web, por lo general en la forma http:aplicacionejemplometodo, que invoca a EjemploController#método, y presenta los datos usando el archivo de plantilla appviewsejemplométodo.html.erb, a no ser que el método redirija a algún otro lugar.

3.5.2. HTML5

HTML, siglas de HyperText Markup Language (Lenguaje de Marcado de Hipertexto), es el lenguaje de marcado predominante para la elaboración de páginas web. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes.

Figura 3.6: HTML5

HTML se escribe en forma de <etiquetas>, rodeadas por corchetes angulares (<,>). HTML también puede describir, hasta un cierto punto, la apariencia de un documento, y puede incluir un script (por ejemplo Javascript), el cual puede afectar el comportamiento de navegadores web y otros procesadores de HTML.

HTML5 no es simplemente una nueva versión del lenguaje de marcación HTML, sino una agrupación de diversas especificaciones concernientes al desarrollo web. Es decir, HTML5 no se limita sólo a crear nuevas etiquetas, atributos y eliminar aquellas marcas que están en desuso o se utilizan inadecuadamente, sino que va mucho más allá.

- **Estructura del cuerpo:** La mayoría de las webs tienen un formato común, formado por elementos como cabecera, pie, navegadores, etc. HTML 5 permite agrupar todas estas partes de una web en nuevas etiquetas que representarán cada uno de las partes típicas de una página.
- **Etiquetas para contenido específico:** Hasta ahora se utilizaba una única etiqueta para incorporar diversos tipos de contenido enriquecido, como animaciones Flash o vídeo. Ahora se utilizarán etiquetas específicas para cada tipo de contenido en particular, como audio, vídeo, etc.
- **Canvas:** es un nuevo componente que permitirá dibujar, por medio de las funciones de un API, en la página todo tipo de formas, que podrán estar animadas y responder a interacción del usuario. Es algo así como las posibilidades que nos ofrece Flash, pero dentro de la especificación del HTML y sin la necesidad de tener instalado ningún plugin.
- **Bases de datos locales:** el navegador permitirá el uso de una base de datos local, con la que se podrá trabajar en una página web por medio del cliente y a través de un API. Es algo así como las Cookies, pero pensadas para almacenar grandes cantidades de información, lo que permitirá la creación de aplicaciones web que funcionen sin necesidad de estar conectados a Internet.
- **Web Workers:** son procesos que requieren bastante tiempo de procesamiento por parte del navegador, pero que se podrán realizar en un segundo plano, para que el usuario no tenga que esperar que se terminen para empezar a usar la página. Para ello se dispondrá también de un API para el trabajo con los Web Workers.
- **Aplicaciones web Offline:** Existirá otro API para el trabajo con aplicaciones web, que se podrán desarrollar de modo que funcionen también en local y sin estar conectados a Internet.
- **Geolocalización:** Las páginas web se podrán localizar geográficamente por medio de un API que permita la Geolocalización.

- **Nuevas APIs para interfaz de usuario:** temas tan utilizados como el "drag & drop" (arrastrar y soltar) en las interfaces de usuario de los programas convencionales, serán incorporadas al HTML 5 por medio de un API.

- **Fin de las etiquetas de presentación:** todas las etiquetas que tienen que ver con la presentación del documento, es decir, que modifican estilos de la página, serán eliminadas. La responsabilidad de definir el aspecto de una web correrá a cargo únicamente de CSS.

3.5.3. CSS3

Figura 3.7: CSS3

Las hojas de estilo en cascada (en inglés Cascading Style Sheets), CSS es un lenguaje usado para definir la presentación de un documento estructurado escrito en HTML o XML (y por extensión en XHTML). El W3C (World Wide Web Consortium) es el encargado de formular la especificación de las hojas de estilo que servirán de estándar para los agentes de usuario o navegadores.

La idea que se encuentra detrás del desarrollo de CSS es separar la estructura de un documento de su presentación. Por ejemplo, el elemento de HTML <h1> indica que un bloque de texto es un encabezamiento y que es más importante que un bloque etiquetado como <H2>. Versiones más antiguas de HTML permitían atributos extra dentro de la etiqueta abierta para darle formato (como el color o el tamaño de fuente). No obstante, cada etiqueta <H1> debía disponer de la información si se deseaba un diseño consistente para una página y, además, una persona que leía esa página con un navegador perdía totalmente el control sobre la visualización del texto.

Cuando se utiliza CSS, la etiqueta <H1> no debería proporcionar información sobre cómo será visualizado, solamente marca la estructura del documento. La información de estilo, separada en una hoja de estilo, específica cómo se ha de mostrar <H1>: color, fuente, alineación del texto, tamaño y otras características no visuales, como definir el volumen de un sintetizador de voz, por ejemplo.

La información de estilo puede ser adjuntada como un documento separado o en el mismo documento HTML. En este último caso podrían definirse estilos generales en la cabecera del documento o en cada etiqueta particular mediante el atributo "style". La especificación de CSS3 viene con interesantes novedades que permitirán hacer webs más elaboradas y más dinámicas, con mayor separación entre estilos y contenidos. Dará soporte a muchas necesidades de las webs actuales, sin tener que

recurrir a trucos de diseñadores o lenguajes de programación como JavaScript.

3.5.4. SASS

Figura 3.8: SASS

SASS (Syntactically Awesome Stylesheets) es un metalenguaje de Hojas de Estilo en Cascada (CSS). Es un lenguaje de script que es interpretado a CSS. Sass consiste en dos sintaxis. La sintaxis original, usa la indentación para separar bloques de código y el carácter nueva línea para separar reglas. La sintaxis más nueva, SCSS usa el formato de bloques como CSS. Este usa llaves para denotar bloques de código y punto y coma (;) para separar las líneas dentro de un bloque. La sintaxis indentada y los ficheros SCSS tienen las extensiones .sass y .scss respectivamente. La implementación oficial de Sass es software libre y está escrita en Ruby, sin embargo existen otras implementaciones. Es un metalenguaje anidado, lo que es válido en CSS es válido en SCSS con la misma semántica.

3.5.5. Alfresco

Figura 3.9: Alfresco

Alfresco es un sistema de administración de contenidos de código fuente libre, desarrollado en Java, basado en estándares abiertos y de escala empresarial para sistemas operativos tipo Windows, Unix Solaris y algunas versiones de Linux.

Una de las principales razones por las que se ha elegido Alfresco, es que está diseñado para usuarios que requieren un alto grado de modularidad y rendimiento escalable. Incluye un repositorio de contenidos, un framework de portal web para administrar y usar contenido estándar en portales, una interfaz CIFS que provee compatibilidad de sistemas de archivos en Windows y sistemas operativos tipo Unix, un sistema de administración de contenido web, capacidad de virtualizar aplicaciones web y sitios estáticos vía Apache Tomcat, búsquedas vía el motor Apache Solr-Lucene y flujo de trabajo en jBPM.

3.5.6. SQLite

Figura 3.10: SQLite

SQLite3 es el sistema de gestión de Bases de Datos relacional que Rails soporta por defecto. El acceso a esta es totalmente abstracto desde el punto de vista del programador y Rails gestiona los accesos a la Base de Datos automáticamente (aunque, si se necesita, se pueden hacer consultas directas en SQL).

A diferencia de los sistemas de gestión de Bases de Datos cliente-servidor, el motor de SQLite no es un proceso independiente con el que el programa principal se comunica. En lugar de eso, la biblioteca SQLite se enlaza con el programa pasando a ser parte integral del mismo. El programa utiliza la funcionalidad de SQLite a través de llamadas simples a subrutinas y funciones. Esto reduce la latencia en el acceso a la base de datos, debido a que las llamadas a funciones son más eficientes que la comunicación entre procesos. El conjunto de la Base de Datos (definiciones, tablas, índices, y los propios datos), son guardados como un sólo fichero estándar en la máquina host.

La biblioteca de mapeo objeto-relacional es llamada Active Record. Cuando se utiliza Active Record, la comunicación con la Base de Datos es en Ruby, y él internamente realiza la transformación a un lenguaje que la Base de Datos pueda entender (SQL).

Algunas de las operaciones que se pueden realizar usando Active Record son:

- `all`: para extraer todos los elementos de la tabla.
- `find`(numero): se buscan los elementos por su número de id. Ejemplo:

  ```
  Example.find(3)
  ```

- `first`: muestra el primer elemento de la tabla.
- `last`: selecciona el último elemento de la tabla.
- `delete`(id): borra la fila con dicho id.
- `order`: ordenar según lo que se pida. Ejemplo: Ordenar por nombre

  ```
  Example.order('nombre')
  ```

Rails intenta mantener la neutralidad con respecto a la Base de Datos, la portabilidad de la aplicación a diferentes sistemas de Base de Datos y la reutilización de Bases de Datos preexistentes.

3.6. Modelo REpresentational State Transfer (REST)

REST (Representational State Transfer) es una técnica de arquitectura software para sistemas hipermedia distribuidos como la World Wide Web. Rails fue escrito inicialmente para apoyar el estilo CRUD- Create, Read, Update and Delete (crear, leer, actualizar y eliminar), pero al implementarse el modelo REST Rails lo fue adoptando poco a poco. Algunas de las diferencias que existen entre el modelo REST y CRUD son:

REST request	**CRUD request**	**Acción**
POST /posts	/posts/create	Crear
GET /posts/1	/posts/show/1	Leer
PUT /posts/1	/posts/update/1	Actualizar
DELETE /posts/1	/posts/destroy/1	Eliminar

Tabla 3.1: REST vs CRUD

El conjunto de operaciones más importantes de REST son: POST, GET, PUT y DELETE.

- GET: sirve para leer elementos.
- POST: sirve para crear elementos.
- PUT: sirve para modificar elementos.
- DELETE: sirve para eliminar elementos.

3.7. Prototipo

Para facilitar una visión general de la aplicación web, se ha diseñado un simple prototipo de esta.

Nada más acceder a la aplicación, se solicitará el identificador y la contraseña del usuario (Véase Figura 3.11). Esta vista será igual para cualquier usuario de cualquier rol.

Figura 3.11: Página principal

Una vez el usuario haya iniciado sesión como miembro de la ETSII, aparecerán distintas opciones según los roles que tenga asociado el usuario. Un usuario puede tener dos o más roles, por lo que las opciones se suman. Véase la Tabla 3.2.

En la opción "Consultar Apuntes", se verá un listado de los apuntes que han sido subidos, revisados, y publicado por un usuario gestor.

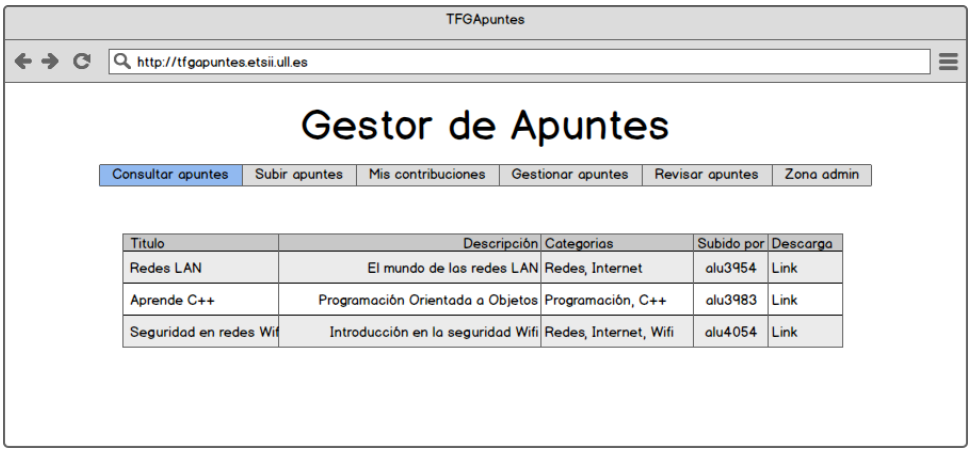

Figura 3.12: Consultar Apuntes

En la opción "Subir Apunte", un usuario podrá realizar una petición para subir un archivo. Cuando el usuario sube un archivo, automaticamente se crea una petición para ese archivo. Una petición puede tener los siguientes estados:

Rol	Opciones
Consultor	▪ Consultar Apuntes
Contribuyente	▪ Consultar Apuntes ▪ Subir Apuntes ▪ Mis Contribuciones
Gestor	▪ Consultar Apuntes ▪ Gestionar Apuntes
Revisor	▪ Consultar Apuntes ▪ Revisar Apuntes
Administrador	▪ Consultar Apuntes ▪ Subir Apuntes ▪ Mis Contribuciones ▪ Gestionar Apuntes ▪ Revisar Apuntes ▪ Zona Admin

Tabla 3.2: Opciones según el rol

- **Nuevo:** Al archivo subido aún no se le ha asignado ningún Revisor.
- **En Proceso:** Al archivo está en proceso de revisión.

- **Aceptado:** Todos los revisores han revisado y aceptado el archivo.

- **Denegado:** Alguno de los revisores ha denegado el archivo.

- **Publicado:** El archivo está publicado y, por tanto, visible para todos los demás usuarios.

Figura 3.13: Subir Apuntes

En la opción "Mis Contribuciones", se verá un listado de los archivos que ha subido el usuario al gestor documental. Además, este podrá eliminar el archivo del repositorio de Alfresco si así lo desea.

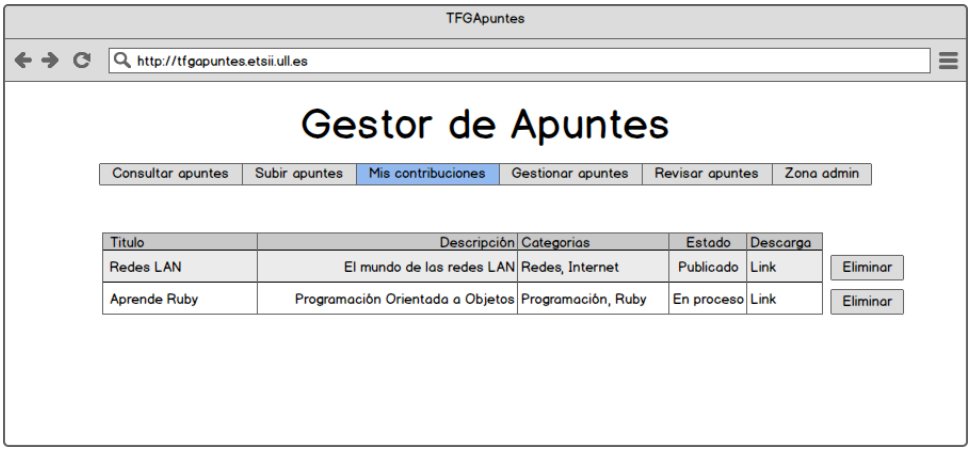

Figura 3.14: Mis Contribuciones

En la opción "Gestionar Apuntes", los usuarios que tengan rol de gestor, podrán asignar revisores a un archivo siempre y cuando aún no se le haya asignado. Cuando un archivo ha sido aceptado por todos sus revisores, al usuario gestor le aparecerá la opción de publicar dicho archivo.

Figura 3.15: Gestionar Apuntes

En la opción de "Revisar Apuntes", aparecerán los archivos que un usuario gestor le ha asignado. Un revisor descargaría el archivo, y tras revisarlo, lo aceptaría o lo denegaría. En el caso de denegarlo, tendrá que exponer sus razones en un cuadro de texto que se le enviará al usuario contribuyente que ha subido el archivo.

Figura 3.16: Revisar Apuntes

En la "Zona Admin", se podrá crear nuevas categorías y otorgar rol de gestor a los usuarios.

Figura 3.17: Zona Admin

Capítulo 4

Desarrollo e implementación de la aplicación Web

Una aplicación web es una herramienta que los usuarios pueden utilizar accediendo a un servidor web a través de Internet mediante un navegador. Las aplicaciones web son populares debido a lo práctico del navegador web como cliente ligero, a la independencia del sistema operativo, así como a la facilidad para actualizar y mantener aplicaciones web sin distribuir e instalar software a miles de usuarios potenciales.

En este capítulo se tratará de explicar todo lo referente al desarrollo e implementación de la aplicación web, así como las tecnologías usadas, definiendo paso a paso todo lo realizado para llevar a cabo el desarrollo de este sistema.

4.1. Instalación de las herramientas necesarias

1. Instalación de Ruby

   ```
   $ sudo apt-get install ruby irb rdoc
   ```

2. Instalación de RVM
   RVM es una herramienta de línea de comandos que permite instalar y gestionar fácilmente diferentes versiones de Ruby dentro de un mismo entorno.

   ```
   $ sudo -s
   # apt-get install curl
   # curl -L get.rvm.io | bash -s stable
   ```

   a) Comenzar a usar RVM

```
# source /etc/profile.d/rvm.sh
```

b) Instalación de Dependencias

```
# rvm autolibs enable
# rvm requirements
```

c) Instalación de Ruby desde RVM

```
# rvm install 2.1.1
```

d) Para usar una versión por defecto

```
# rvm use 2.1.1 --default
```

3. Creación y uso de Gemset
   Gemset son un conjunto de gemas que se definen para una versión de Ruby, aislados unos de otros gracias a RVM.

   Una buena práctica es definir un gemset para cada proyecto.

   a) Creación de un Gemset

   ```
   $ rvm gemset create <proyecto>
   ```

   b) Uso

   ```
   $ rvm gemset use <proyecto>
   ```

   c) Evitar el cambio de gemset para cada proyecto

   1) Crear el fichero `.rvmrc` en la raíz del proyecto.
   2) Añadir el siguiente contenido y guardar los cambios:
      `rvm --create use default@<proyecto> > /dev/null`

   d) Ventajas:

   - Carga automática del gemset de un proyecto
   - Si no existe el gemset, se crea.

4. Instalación Rails

   ```
   $ gem install rails
   ```

4.2. Creación de un nuevo proyecto RoR

Para poder crear un nuevo proyecto debemos ejecutar en la consola el siguiente comando:

```
$ rails new NombreProyecto
```

- Instalación de todo lo necesario para que la aplicación funcione correctamente

  ```
  bundle install
  ```

- Creación de la base de datos:

  ```
  rake db:create
  ```

- Acceso al Sistema de gestión de base de datos desde Rails:

  ```
  rails dbconsole
  ```

- Ejecutar el servidor

  ```
  rails s
  ```

4.3. Gemas Utilizadas

Para este proyecto, se han utilizado las siguientes gemas:

- Devise
- CanCanCan
- Rolify
- Twitter-bootstrap-rails
- HttMultiParty
- Carrierwave

4.3.1. Devise

Devise gestiona la autenticación a todos los niveles, cubre además de los modelos, también vistas y controladores. La arquitectura de Devise es modular y consta de once módulos cada uno de los cuales cubre un aspecto diferente de la autenticación. Por ejemplo el módulo *Rememberable* recuerda la autenticación del usuario en una cookie mientras que otro módulo, *Recoverable*, se ocupa de reiniciar la clave del usuario enviando instrucciones por correo. Este enfoque hace que sea muy fácil escoger exactamente las funcionalidades de autenticación que queramos utilizar.

Sus principales características son:

- Se basa Rack.
- Es una solución completa basada en el Modelo Vista Controlador (MVC) de Rails.
- Permite tener múltiples roles.
- Se basa en un concepto modular ya que utiliza sólo lo que realmente necesita. Se compone de 11 módulos:
  - Database Authenticatable
  - Token Authenticatable
  - Omniauthable
  - Confirmable
  - Recoverable
  - Registerable
  - Rememberable
  - Trackable
  - Timeoutable
  - Validatable
  - Lockable

Para poder usar la gema 'devise' se deben seguir las siguientes instrucciones:

1. Instalación de la gema
   - Introducir en el Gemfile:

     ```
     gem 'devise'
     ```

- Ejecutar en la consola:
  - `bundle install`
  - `rails g devise:install`

2. Generación del modelo.

```
$ rails g devise User
```

4.3.2. CanCanCan

Es la versión para Rails 4.0 de CanCan. Con esta gema gestionaremos la autorización que tienen los usuarios para realizar acciones, una vez autenticados en el sistema. CanCanCan es una biblioteca de autorización para Ruby on Rails, restringe lo que los recursos de un usuario dado está autorizado a acceder. Todos los permisos se definen en un solo lugar, la clase de capacidad (*Ability*) y no se duplican entre controladores, vistas y consultas de bases de datos.

1. Instalación de la gema
   - Introducir en el Gemfile:

     `gem 'cancancan'`
   - Ejecutar en la consola:
     - `bundle install`
2. Generación de la clase Ability. Este archivo define, si el usuario es considerado como administrador entonces tiene derecho de manipular las acciones protegidas por CanCan, de otra manera se considera no administrador.

```
$ rails generate cancan:ability
```

4.3.3. Rolify

Rolify es una función de gestión de bibliotecas con recursos de alcance. Esta biblioteca es lo suficientemente genérica para ser utilizada por cualquier solución de autenticación. Normalmente se usa con CanCan (o CanCanCan) y Devise.

1. Instalación de la gema
   - Introducir en el Gemfile:

     `gem 'rolify'`

- Ejecutar en la consola:
  - `bundle install`

2. Generación de los roles de usuario. Esto genera el archivo `role.rb` que se encuentra en `app/models`

```
$ rails generate rolify:role
```

4.3.4. Twitter-bootstrap-rails

Bootstrap es una herramienta creada por Twitter y diseñada para acelerar el desarrollo de aplicaciones y sitios web. Incluye CSS básicos y HTML para tipografía, formularios, botones, tablas, navegación, y más.

1. Instalación de la gema
   - Introducir en el Gemfile:

     `gem 'Twitter-bootstrap-rails'`
   - Ejecutar en la consola:
     - `bundle install`
2. Ejecutar el generador bootstrap para añadir Bootstrap al proyecto:

```
$ rails generate bootstrap:install
```

4.3.5. Httmultiparty

HTTMultiParty es una gema que proporciona subidas multipart. En el proyecto, se usará para las subidas de ficheros al repositorio Alfresco.

1. Instalación de la gema
   - Introducir en el Gemfile:

     `gem 'Httpmultiparty'`
   - Ejecutar en la consola:
     - `bundle install`
2. Incluír HTTMultiparty en nuestra clase cliente.

```
$ rails generate rolify:role
```

4.3.6. Carrierwave

Esta gema proporciona una forma simple pero extremadamente flexible de subir archivos desde aplicaciones en Ruby. Funciona bien en aplicaciones web basadas en Rack, tal que Ruby on Rails.

1. Instalación de la gema
   - Introducir en el Gemfile:

     `gem 'Carrierwave'`
   - Ejecutar en la consola:
     - `bundle install`
2. Crear un "uploader". Esto crea el archivo `fichero_uploader.rb` que se encuentra en `app/uploaders`

   ```
   $ rails generate uploader Fichero
   ```
3. Ahora se podrá hacer uso de la clase Fichero para guardar y recuperar archivos de esta forma:

   ```
   uploader = FicheroUploader.new

   uploader.store!(mi_fichero)

   uploader.retrieve_from_store!('mi_fichero.png')
   ```

4.4. Creación de Modelos

A continuación se entrará en detalle en la creación de todos los modelos de los cuales se ha hecho uso en esta aplicación.

4.4.1. Creación del modelo User

El *scaffolding* es un método para construir aplicaciones basadas en bases de datos. Sirve para implantar de manera inmediata un entorno de administración temporal sobre el que trabajar. El *scaffold*, generalmente es sustituido por métodos propios en el controlador.

Para la creación del modelo *User*, se utilizó el *scaffolding*:

```
$ rails generate scaffold User email:string password:string
```

Seguidamente, se ha hecho uso de la gema *Devise* ya mencionada anteriormente. Tras instalarla, ya tendríamos nuestro modelo *User* creado.

Para cargar las migraciones pendientes a la base de datos, se debe hacer:

```
$ rake db:migrate
```

4.4.2. Creación del modelo Post

Para crear el modelo *Post* también se ha usaro el *scaffolding*:

```
$ rails generate scaffold Post title:string description:string
file:string visible:boolean
```

Y posteriormente se carga la base de datos que se acaba de crear con:

```
$ rake db:migrate
```

4.4.3. Creación del modelo Petition

El modelo *Petition* se usa para las peticiones de subida de archivo. Una vez más, se hace uso del *scaffolding*:

```
$ rails generate scaffold Petition status:string
```

Recordar que hay que cargar las migraciones a la base de datos de nuevo:

```
$ rake db:migrate
```

4.4.4. Creación del modelo Category

El modelo *Category* se usará para administrar las categorías de los archivos subidos:

```
$ rails generate scaffold Category name:string description:string
```

y una vez más:

```
$ rake db:migrate
```

4.4.5. Creación del modelo Acceptment

Acceptment (Aceptación) es un modelo creado para almacenar si un usuario con rol de revisor ha aceptado un *Post*, y en caso de que no, la razón por la que se deniega:

Para las *Acceptments* no fue necesario usar el *scaffolding*:

```
$ rails generate model Acceptment accepted:boolean reason:text
```

Puesto que al crear un modelo nuevo, también se crea una migración nueva, esta se ha de cargar en la base de datos:

```
$ rake db:migrate
```

4.5. Relaciones entre los modelos

Cada modelo representa una tabla en una base de datos, por lo tanto, así como las tablas tienen relaciones entre sí también los modelos.

Existen tres tipos de relaciones:

- Relaciones de 1 a 1
- Relaciones de 1 a n
- Relaciones de n a n

4.5.1. Relaciones de 1 a 1

Estas se dan cuando un registro está ligado a otro. En la aplicacion web, sólo existe una relacion 1 a 1: el modelo *Post* y el modelo *Petition*. Ambas tienen una relacion 1 a 1, puesto que una petición (*Petition*) corresponderá únicamente con una publicación (*Post*) y viceversa.

En rails estas relaciones se crean con "has_one" y "belongs_to" en los modelos correspondientes.

4.5.2. Relaciones de 1 a n

Son las relaciones más comunes. En este caso un registro puede tener relación con otros, pero cada uno de esos otros le pertenecerán sólo a ese registro. Para este proyecto se han usado 3 relaciones 1 a n:

- Un *Post* (Publicación) pertenecerá a un usuario (*User*), mientras que ese mismo usuario podrá tener varias publicaciones (*Posts*)

- Un *Post* tiene muchas *Acceptments* (Aceptaciones), mientras que una Aceptación pertenecerá a un *Post*
- Un usuario (*User*) tiene muchas Aceptaciones (*Acceptments*), mientras que una Aceptación pertenercerá a un usuario.

4.5.3. Relaciones de n a n

Estas relaciones se dan cuando un registro puede tener relaciones con otros registros, pero a su vez estos registros además de tener relación con el primero pueden estar vinculados con otros. En esta aplicación web, tenemos las siguientes:

- Un *Post* tiene muchas categorias (Categories), y una Categoría tiene muchos *Posts*.
- Un *Post* tiene muchos revisores (Users), y un Revisor tiene muchos *Posts*.

4.6. Modelos resultantes

Los modelos resultantes trás aplicar las relaciones son los siguientes:

4.6.1. Modelo User

```
class User < ActiveRecord::Base
  has_and_belongs_to_many :posts
  has_many :acceptments
end
```

4.6.2. Modelo Post

```
class Post < ActiveRecord::Base
  belongs_to :user
  has_many :acceptments
  has_and_belongs_to_many :categories
  has_one :petition, :dependent => :destroy
  has_and_belongs_to_many :revisors, :class_name => 'User'
end
```

4.6.3. Modelo Petition

```
class Petition < ActiveRecord::Base
```

```
  belongs_to :post
end
```

4.6.4. Modelo Category

```
class Category < ActiveRecord::Base
  has_and_belongs_to_many :posts, :dependent => :destroy
end
```

4.6.5. Modelo Acceptment

```
class Acceptment < ActiveRecord::Base
  belongs_to :post
  belongs_to :revisor, :class_name => "User", :foreign_key => "user_id"
end
```

4.7. Creación de los roles

Es posible asignar multiples roles a un usuario y guardarlos en una única columna integer usando un *bitmask* o Máscara.

Primero que nada, se ha de añadir la columna "roles_mask" a la tabla *users*. Para ello, se ha de crear una migración y volver a cargar la base de datos:

```
$ rails generate migration add_roles_mask_to_users roles_mask:integer
$ rake db:migrate
```

Seguidamente, será necesario establecer la lista de los posibles roles, que será un array de strings, denominado *ROLES* y definido en el modelo *user*.

```
  ROLES = %w[admin consultor contribuyente gestor revisor]
```

También será necesario agreagar el siguiente código:

```
# in models/user.rb
def roles=(roles)
  self.roles_mask = (roles & ROLES).map { |r|
2**ROLES.index(r) }.inject(0, :+)
end

def roles
  ROLES.reject do |r|
```

```
    ((roles_mask.to_i || 0) & 2**ROLES.index(r)).zero?
  end
end
```

Este código se usa para obtener y establecer la lista de roles a los que un usuario pertenece.

Además, puesto que se está usando Devise con Rails 4, no hay que olvidarse de agregar los roles a la lista permitida en el controlador:

```
class ApplicationController < ActionController::Base
  before_filter :configure_permitted_parameters, if: :devise_controller?

  protected

  def configure_permitted_parameters
    devise_parameter_sanitizer.for(:sign_up) {|u| u.permit(:email, :password,
:password_confirmation, roles: [])}
  end
end
```

Finalmente, se agrega el siguiente código al modelo *user* para averiguar si un usuario es de un determinado rol:

```
# in models/user.rb
def is?(role)
  roles.include?(role.to_s)
end
```

Capítulo 5

Conclusiones y trabajos futuros

La principal función de la aplicación web desarrollada para la Escuela Técnica Superior de Ingeniería Informática de la Universidad de La Laguna, es la centralización de documentación oficial de las asignaturas en una biblioteca virtual. Esto ayudaría a miles de estudiantes de la ETSII en sus estudios al acceder de una forma fácil, via internet, a información verificada y contrastada.

Debido a la gran cantidad de apuntes, documentos y memorias que la aplicación deberá manejar, se ha usando el repositorio de un Gestor de Contenido Empresarial (EMC) como almacenamiento. Para la elección de este Gestor de Contenido Empresarial, se ha realizado un análisis de varios de estos, teniendo en cuenta que debía ser de código abierto. Los Gestores propuestos analizados fueron Athento, Nuxeo, OpenKM y Alfresco, pero debido principalmente a la gran cantidad de documentación oficial encontrada en internet, se acabó eligiendo este último.

Además de Alfresco, para este proyecto también se ha hecho uso del framework Ruby on Rails (RoR), HTML5 y CSS3 para la creación de la aplicación. La razón de usar Ruby on Rails (RoR) ha sido por su facilidad de uso, por su lenguaje de programación (Ruby), y por la gran cantidad de documentación disponible online.

La aplicación consta de varios roles, según los cuales un usuario podrá tener ciertos permisos para realizar algunas de las acciones de gestión, consulta, subida de archivo, y otros.

A nivel personal he podido familiarizarme con las diferencias entre Ruby on Rails 3 y 4, ya que este es el primer proyecto que realizo usando esta nueva versión, y los cambios de una a otra han sido, aunque no numerosos, algo complicados de entender.

Como posibles líneas de trabajo futuro, por un lado se podría ampliar las funcionalidades de la aplicación web, agregar más roles a esta y, puesto que en este proyecto se ha centrado más la atención en la funcionalidad que en el aspecto de la aplicación, sería conveniente mejorar su estética para hacerla más orientada a usuario.

También cabría la posibilidad de utilizar algunas funciones que ofrece Alfresco

tales como la integración de Microsoft Office, puesto que Alfresco en este punto del trabajo sólo se usa como repositorio, pudiendo ofrecer mucho más.

Capítulo 6

Conclusions and future lines of work

The main function of the web application developed for the ETSII, is the centralization of official documentation of the courses in a virtual library. This would help thousands of students from the school in their studies by easily getting access, via internet, to checked and verified information.

Due to the large number of notes, documents and reports that the application must handle, the use of a Enterprise Content Manager's repository has been needed in order to store all those documents. For the choice of the Enterprise Content Manager, an analysis of several of these has been made, considering that it should be open source. The proposed Content Managers analyzed were Athento, Nuxeo, OpenKM and Alfresco, but mainly because of the large number of official documentation found on the internet, Alfresco was the chosen one.

In addition to Alfresco, it has also been used the framework Ruby on Rails (RoR), HTML5 and CSS3 to create the application. The reason why Ruby on Rails (RoR) has been chosen it's because of its ease of use, its programming language (which is Ruby), and the large amount of documentation available online.

The application has several roles, according to which a user may have certain permissions to perform some of the actions of management, consulting, file upload, and others.

At a personal level i have been able to familiarize with the main differences between Ruby on Rails 3 and 4, since this is the first project i have done using this new version. Even if the changes between versions haven't been numerous, it has been quite difficult to understand.

As possible future of lines of work, on one side we could expand the web application functionality, add more roles to it and, since the project has been more focused in the functionality rather than the aspect itself, it would be convenient to improve the aspect so it is more user friendly.

Another approach would be to use some of the features that Alfresco offers, such as Microsoft Office integration, since Alfresco at this work level only uses its Repository, when it could bring so much more.

Capítulo 7

Presupuesto

Para la creación del presupuesto, se han establecido una serie de tareas que se han realizado a lo largo de la ejecución del proyecto a las cuales se le asignará una cantidad estimada de dinero en base a las horas trabajadas en cada tarea. Debido a esto, el presupuesto se ha dividido en dos partes fundamentales: el presupuesto de las tareas realizadas (estimando el precio por hora) y el presupuesto en base al desarrollo e implementación de la aplicación Web.

7.1. Presupuestos por Tarea

Sabiendo que el proyecto se divide en tres tareas fundamentales: Análisis y Especificación de Requisitos, Diseño Técnico del Sistema y Desarrollo e Implementación del Sistema, se han podido valorar el precio por hora de cada tarea.

Posteriormente, mediante la herramienta GanttProject, se ha creado el diagrama de Gantt correspondiente.

Y finalmente se ha calculado el precio total por horas trabajadas de cada tarea.

Tareas	**Euros/Hora**	**Horas Trabajadas**	**Total**
Análisis funcional y Especificación de Requisitos	80 €/h	25 horas	2000 €
Diseño Técnico del Sistema	100 €/h	25 horas	2500 €
Desarrollo e Implementación del Sistema	*Ver Tabla 7.2*		
TOTAL	**4500 €**		

Tabla 7.1: Tabla de Presupuestos por Tarea

7.2. Presupuestos por Módulo del Diseño e Implementación del Sistema

Para la tarea de Desarrollo e Implementación del Sistema, se ha optado por buscar el precio por módulo integrado en la aplicación web y luego hallar el total.

Desarrollo e Implementación del Sistema	
Módulo	**Precio**
Roles de Usuarios	500 €
Módulo formularios	90 €
Módulo integración con Alfresco	200 €
Módulo Archivos	240 €
Módulo Base de Datos	250 €
Módulo Menú de Aplicación Web	45 €
TOTAL	**1325 €**

Tabla 7.2: Tabla de Presupuestos por Módulo del Diseño e Implementación del Sistema

7.3. Presupuesto Total

Para concluir se han sumado los dos subtotales, dando como precio final del presupuesto del proyecto tres mil setecientos veinticinco euros.

PRESUPUESTO TOTAL	
Tabla	**Precio**
Presupuesto Tabla 7.1	4500 €
Presupuesto Tabla 7.2	1325 €
TOTAL	**5825 €**

Tabla 7.3: Tabla con el Presupuesto Total del Proyecto

7.4. Diagrama de Gantt

Usando la herramienta GanttProject se ha realizado un diagrama de Gantt: Figura 7.4.

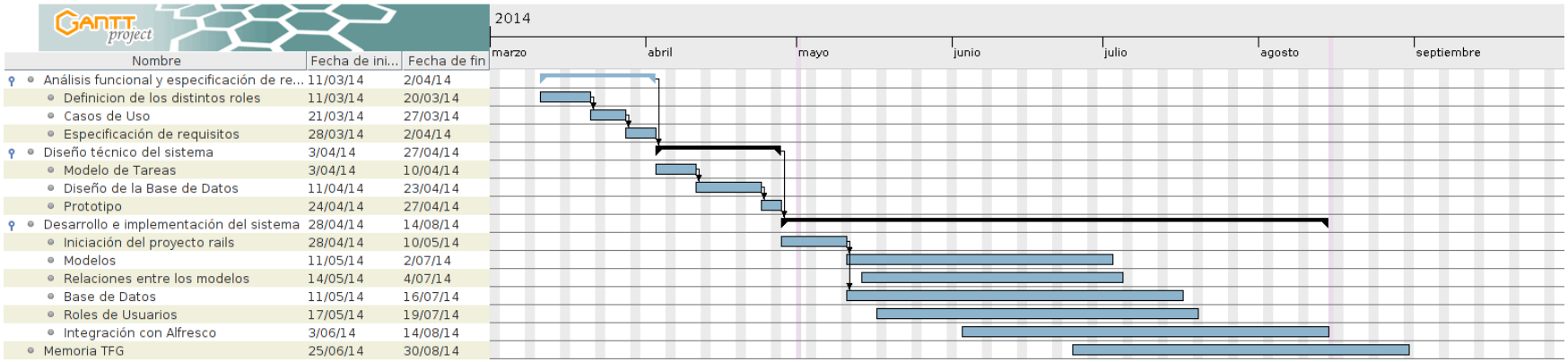

Figura 7.1: Diagrama de Gantt del Proyecto de Fin de Grado

Capítulo 8

Acrónimos

- ECM: Enterprise Content Management (Gestor de Contenido Empresarial)
- ULL: Universidad de La Laguna.
- PDF: Portable Document Format (formato de documento portable)
- RoR: Ruby on Rails.
- ERS: Especificación de Requisitos Software.
- REST: Representational State Transfer (Transferencia de Estado Representacional)
- CRUD: Create, Read, Update and Delete (crear, leer, actualizar y eliminar)
- TFG: Trabajo de Fin de Grado

Bibliografía

[1] Sara María García Blanco and Enrique Morales Ramos. *Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*. Thomson Paraninfo, 2003.

[2] Caridad Racero Borrell. Importancia de la ingeniería de requerimientos dentro del ciclo de desarrollo de software (spanish). importance of requirements engineering in the software development life cycle (english). *Revista Técnica de la Empresa de Telecomunicaciones de Cuba, S.A*, pages 52–56, 2006.

[3] ReadySet. Plantillas para hacer un análisis funcional. `http://readyset.tigris.org/nonav/es/templates/frameset.html`.

[4] Ian Sommerville. *Ingeniería de Software*. Pearson Educación, 2002.

[5] Mario G. Piattini Velthuis, JosÃ© A. CalvoManzano Villalón, Joaquin Cervera Bravo, and Luis Fernández Sanz. *Análisis y Diseño de Aplicaciones Informáticas de Gestión*. RA-MA Editorial, mayo de 2007.