Universidad
Carlos III de Madrid

PROYECTO FIN DE CARRERA

DESARROLLO DE UNA APLICACIÓN WEB PARA LA CREACIÓN DE EXÁMENES DE OPCIÓN MÚLTIPLE

Autor: Elena Martín García
Tutor: Jesús Arias Fisteus

Leganés, 23 de Octubre de 2013

Agradecimientos

Quiero agradecer a Jesús la paciencia que ha tenido conmigo en la realización de este proyecto. Muchas gracias por todo lo que me has enseñado tanto en las asignaturas, como en los trabajos dirigidos y en este proyecto. Me has recordado que aún existen buenos profesores que se interesan por sus alumnos.

También quiero agradecer a Tomás su apoyo para que todo siguiera adelante en esos momentos de recaída.

Por último, agradecer a mi hermana Nerea su ayuda en determinados momentos de este proyecto.

Resumen

Partimos de un sistema de corrección automática de exámenes llamado Eyegrade, desarrollado en Python y que permite la creación de exámenes con distintos modelos (reorganizando preguntas y respuestas) y su posterior corrección a través de una cámara Web. Eyegrade permite crear estadísticas a partir de los resultados obtenidos en las distintas correcciones de los exámenes.

Se ha desarrollado una aplicación Web integrándose con dicho sistema, en la cual se permite la creación de exámenes y sus respectivas preguntas tanto en formato LaTeX (utilizado por Eyegrade), como PDF. También permite editar tanto preguntas como exámenes y descargar sus ficheros.

Todo esto facilita enormemente la utilización de Eyegrade de forma más sencilla y sin necesidad de descargar ningún programa para la realización e impresión de exámenes. También se alberga la posibilidad de extender este proyecto para integrar totalmente Eyegrade en la aplicación Web en el resto de funciones de éste: corrección de exámenes y estadísticas de corrección.

Índice general

1. INTRODUCCIÓN

1.1. Motivación del proyecto

La corrección automática de exámenes puede evitar una pérdida de tiempo para los profesores, de modo que éstos puedan dedicarlo a otros menesteres como tutorías. Una corrección automática de exámenes puede facilitar enormemente la tarea del profesor y simplificar el proceso, pudiéndolo reducir incluso a un par de horas únicamente, dependiendo de la complejidad del examen y la cantidad de exámenes a corregir.

Hoy en día existen herramientas que permiten puntuar texto libre. Pero lo que mejores resultados obtiene es la puntuación de exámenes basados en preguntas de respuesta múltiple.

Existen sistemas que permiten el reconocimiento de respuestas para formularios y exámenes. Estos sistemas son de alto coste, y las empresas que los utilizan no disponen de la cantidad de maquinaria necesaria para utilizarlos a gran escala, como para una corrección de exámenes realizada por varios profesores.

Para el ejemplo de la Universidad Carlos III de Madrid, se dispone de una única máquina que realiza este proceso situada en Getafe. Para acceder a dicha máquina los profesores deben pedir una cita, disponer de un becario que supervise el trabajo realizado y tiempo para realizar los traslados a Getafe desde otros campus. Además, estos sistemas necesitan un papel específico y unas características muy concretas que deben cumplirse.

Por todo ello, varios profesores de la Universidad Carlos III colaboraron en la realización de un sistema de corrección de exámenes basados en preguntas de respuesta múltiple mediante cámara Web llamado Eyegrade, del que hablaremos en profundidad en el apartado 2.1 de esta memoria. Eyegrade se encuentra todavía en fase alpha, aunque es totalmente funcional ya que ha sido utilizado previamente. Eyegrade permite la corrección de exámenes tipo test mediante una cámara Web, y también su creación en diferentes modelos (reorganizando preguntas y respuestas).

En Eyegrade, los exámenes se crean manualmente editando un fichero XML que sigue una serie de normas respecto a las etiquetas y un fichero LaTeX que contiene el formato del examen. En este último pueden incrustarse marcas de LaTeX para controlar el formato, incluir notación matemática, etc. Todo esto implica un control de herramientas difícilmente alcanzable para el usuario medio.

Se pretende extender el uso de esta aplicación creando herramientas que faciliten la tarea de realizar exámenes y compartir preguntas con otros profesores; y en un futuro, el resto de funciones que implementa Eyegrade. Es deseable que estas herramientas estén disponibles en una interfaz gráfica amigable y que no requieran instalación en el equipo del usuario. Por ello, se desea que estas herramientas estén disponibles vía Web, donde cualquier usuario pueda acceder a ellas desde cualquier hardware y software que permita conexiones Web y disponga de navegador.

1.2. Objetivos

El objetivo de este proyecto es desarrollar una aplicación Web que facilite la creación de exámenes para Eyegrade. En concreto, la aplicación debe:

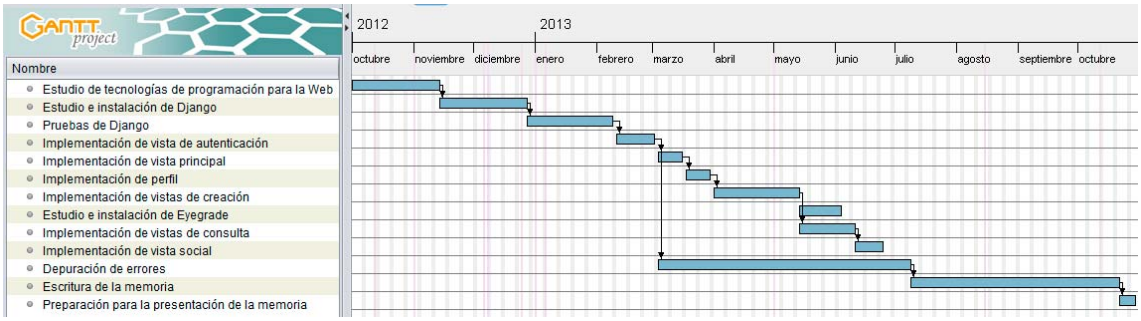
- Permitir la creación y modificación de preguntas, respuestas y exámenes.
- Permitir la eliminación de exámenes.
- Incorporar un sistema de gestión de usuarios.
- Implementar mecanismos de control de acceso a la información, de tal forma que cada usuario pueda mantener sus preguntas y exámenes privados, o compartirlos con determinados usuarios.
- Incorporar un sistema de gestión de asignaturas.
- Permitir exportar los exámenes creados en formatos LaTeX, XML y PDF.

1.3. Plan de trabajo

Este proyecto tiene una duración aproximada de 12 meses. Se emplearán 4 horas diarias durante los días laborables. La lista de tareas es la siguiente:

Nombre	**Fecha de inicio**	**Fecha de fin**	**Duración (días laborables)**
Estudio de tecnologías de programación para la Web	1/10/12	13/11/12	30
Estudio e instalación de Django	14/11/12	27/12/12	30
Pruebas de Django	28/12/12	8/02/13	30
Implementación de vista de autenticación	11/02/13	1/03/13	15
Implementación de vista principal	4/03/13	15/03/13	10
Implementación de perfil	18/03/13	29/03/13	10

Implementación de vistas de creación	1/04/13	13/05/13	30
Estudio e instalación de Eyegrade	14/05/13	3/06/13	15
Implementación de vistas de consulta	14/05/13	10/06/13	20
Implementación de vista social	11/06/13	24/06/13	10
Depuración de errores	4/03/13	8/07/13	90
Escritura de la memoria	9/07/13	21/10/13	74
Preparación para la presentación de la memoria	22/10/13	29/10/13	6



1.4. Estructura de la memoria

Para comprender mejor esta memoria, debemos explicar su estructura. Esta memoria consta de 9 apartados.

El apartado en el que nos encontramos, el primero, pone en contexto el por qué de la realización de este proyecto y cuáles son los objetivos a perseguir. Además muestra un diagrama con el plan de trabajo que se ha seguido y obviamente, este subapartado que explica cómo está estructurada esta memoria.
Expliquemos brevemente de qué tratan el resto de apartados:

- **ESTADO DEL ARTE:** En este apartado se explican las funciones de Eyegrade y su funcionamiento más cercano a la finalidad del proyecto. Además, analizamos varias tecnologías de programación para la realización de la aplicación Web. Tras mostrar sus características y desventajas, pasamos a realizar una comparativa que nos permita decantarnos por una de ellas para la realización del proyecto. Una vez que hemos explicado los motivos por los que se ha elegido una tecnología, realizamos un análisis en

profundidad que muestra los aspectos más relevantes de ésta para este proyecto.

- **REQUISITOS:** Aquí pasamos a explicar los requisitos que el proyecto debe cumplir para cubrir la motivación de éste.

- **DISEÑO DE LA APLICACIÓN:** Este apartado cubre cómo se ha diseñado la aplicación, qué detalles se han tenido en cuenta a la hora de desarrollarla, etc.

- **IMPLEMENTACIÓN:** Detalles más técnicos sobre la implementación de la aplicación. Posibilidades para abordar las soluciones y por qué se han elegido unas respecto a otras.

- **PRUEBAS:** Como toda aplicación, se han realizado una serie de pruebas para depurar fallos y otras para ver el alcance de las posibilidades que disponemos para abordar las soluciones respecto a algo en concreto de implementación. También se explica cómo poner en marcha la aplicación Web.

- **CONCLUSIONES Y TRABAJOS FUTUROS:** Una vez finalizada la realización del proyecto se han extraído unas conclusiones sobre su realización. Además se proponen herramientas y alternativas para ser realizadas en un futuro para la mejora de la aplicación Web.

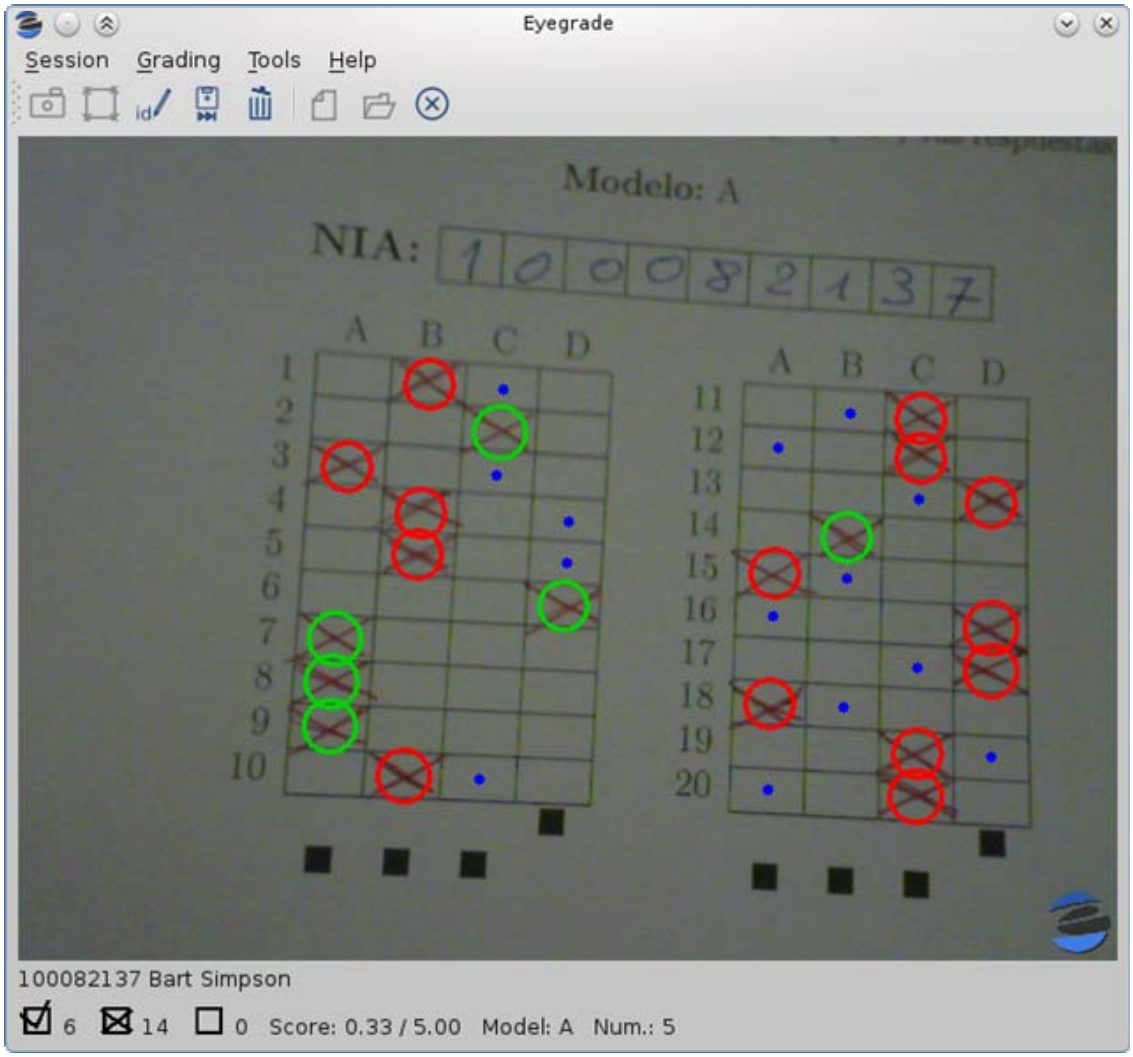
- **PRESUPUESTO:** Todo proyecto debe estar presupuestado, pues nos indica si es factible realizarlo y así analizar el coste de realizarlo frente a los ingresos que puede generar o el ahorro de coste/tiempo por sustituir otras herramientas.

- **REFERENCIAS BIBLIOGRÁFICAS:** Aquí se exponen y detallan todas las referencias bibliográficas utilizadas para la realización de este proyecto, de modo que el lector pueda profundizar en detalles expuestos en esta memoria.

2. ESTADO DEL ARTE

2.1. Eyegrade

La función principal de Eyegrade [1, 2, 3] es puntuar exámenes basados en preguntas de respuesta múltiple (Multiple Choice Question, MCQ). Para ello utiliza una cámara Web, convirtiéndolo en una solución portable y de bajo coste. El programa es de software libre.



Las principales funciones de Eyegrade son:

- **Creación de exámenes:** Aunque se puede crear exámenes con otras utilidades, Eyegrade integra una utilidad para crear exámenes basados en preguntas de respuesta múltiple. Puede crear los exámenes en formato PDF. Además puede crear

automáticamente varias versiones del examen reordenando las preguntas y respuestas de cada una.

- **Corrección de exámenes:** Usando una cámara Web, la interfaz gráfica de usuario de Eyegrade permite corregir un examen. Eyegrade es capaz de reconocer no solo las respuestas a las preguntas, si no también la identidad del estudiante usando su módulo de reconocimiento de dígitos escritos a mano. El proceso entero es supervisado por el usuario para poder detectar y solucionar errores potenciales del sistema.

- **Extracción de estadísticas:** se pueden ver las estadísticas pregunta a pregunta para analizar qué temas están claros para la mayoría de los estudiantes, y qué temas necesitan un trabajo más exhaustivo.

- **Exportación de puntuaciones:** las puntuaciones pueden ser exportadas a formato CSV, compatible con otros programas del estilo hojas de cálculo.

Exámenes

Para mantener la compatibilidad con Eyegrade, es conveniente que en este proyecto se utilice el mismo formato para describir las preguntas de un examen. Dada la importancia que tendrá este formato, por tanto, en aspectos tan relevantes como el diseño del modelo de datos de este proyecto, se describe éste a continuación.

Como ya hemos mencionado en el apartado *1.1. Motivación del proyecto*, Eyegrade usa un modelo XML para describir los exámenes y una plantilla en formato LaTeX que contiene la estructura del examen (colocación de los elementos, indentación, etc.). El primer fichero, en formato XML, contiene dos bloques. El primer bloque contiene información general del examen, gracias a nodos XML.

El nodo raíz se llama <*exam*> y posee los siguientes atributos:

```
<exam xmlns="http://www.it.uc3m.es/jaf/eyegrade/ns/"
    xmlns:eye="http://www.it.uc3m.es/jaf/eyegrade/ns/">
```

Como nodos hijo de <*exam*>, tenemos los siguientes nodos de información general:

- <*subject*>: Este nodo representa el nombre de la asignatura. El texto se escribe entre las etiquetas de inicio y fin.

- <*degree*>*:* Representa la titulación a la que pertenece la asignatura. El texto se escribe entre las etiquetas de inicio y fin.

- *<title>:* Aquí se incluye el título que se desea dar al examen en concreto. El texto se escribe entre las etiquetas de inicio y fin.

- *<date>:* Representa la fecha en la que el examen será realizado. Puede ser escrita en cualquier formato conocido. El texto se escribe entre las etiquetas de inicio y fin.

- *<duration>:* Aquí se incluye la duración de dicho examen. Como en el caso de la fecha, no tiene restricciones de formato. El texto se escribe entre las etiquetas de inicio y fin.

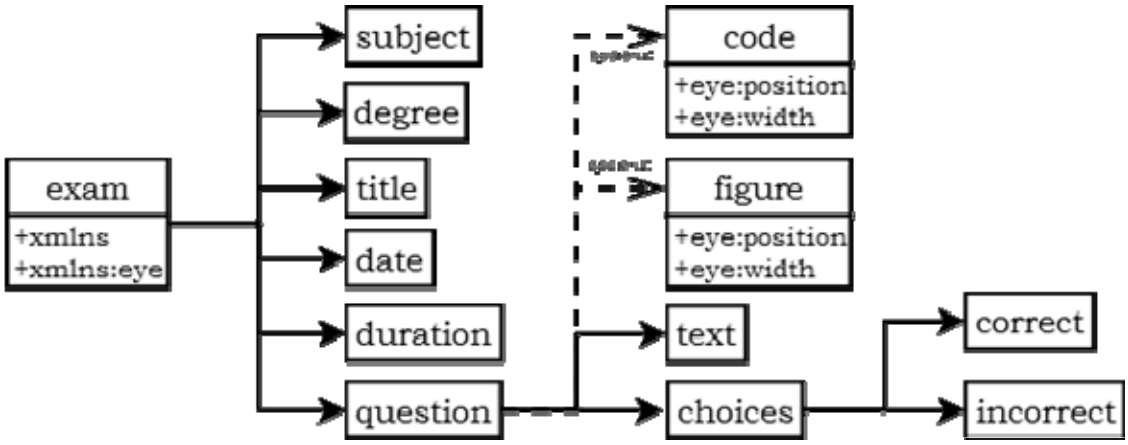
Como acabamos de ver, toda la información del examen se escribe dentro del nodo raíz *<exam>* y el texto se escribe entre las etiquetas de inicio y fin de cada uno, sin atributos.

El segundo bloque del fichero XML contiene las preguntas, que se representan mediante el nodo *<question>* y que también está dentro del nodo <exam>, a continuación de los nodos que describen la información general del examen. Analicemos sus características:

- Posee dos nodos hijo obligatorios, llamados *<text>* y *<choices>*. *<text>* representa el texto de la pregunta, que se escribe entre las etiquetas de inicio y fin. *<choices>* abarca las respuestas de la pregunta. Cada respuesta se representa entre los nodos *<correct>* o *<incorrect>*, según sea la respuesta correcta o incorrecta respectivamente. Solo puede haber una respuesta correcta por cada pregunta.

- Tiene dos nodos opcionales llamados *<code>* y *<figure>*. *<code>* se utiliza para incluir código a la pregunta. Este código se escribe entre las etiquetas de inicio y fin. *<figure>* se utiliza para incluir una imagen en la pregunta. Esta imagen, en Eyegrade, para que sea procesada, debe estar en formato *.eps.* Se escribe la ruta de la imagen entre las etiquetas de inicio y fin. Puede escribirse simplemente su nombre si está incluida en la misma carpeta que el fichero XML. Ambos nodos poseen dos atributos, *eye:position* y *eye:width*. *eye:position* puede tener uno de los siguientes valores: "*center*" o "*right*". Si no es especificado, toma por defecto el valor "*center*". *eye:width* puede tomar cualquier valor float positivo. Este atributo es obligatorio especificarlo para *<figure>*, siendo también obligatorio para *<code>* en el caso de que se haya especificado que el atributo *eye:position* tiene el valor de "*right*".

El árbol de nodos quedaría de la siguiente manera:



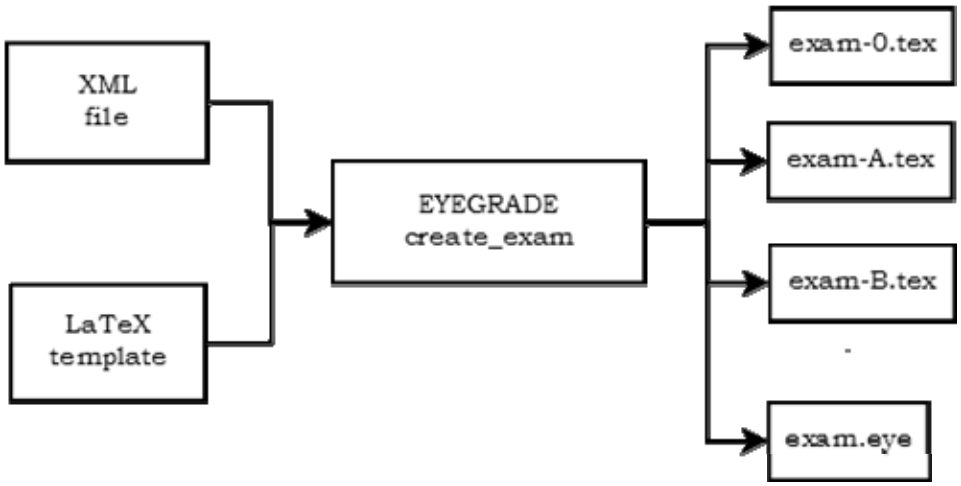
El fichero LaTeX que utiliza Eyegrade se trata de una plantilla con la que forma la estructura del examen. Dicha plantilla obtiene datos del fichero XML para formar el examen final. Estos datos pueden ser incluidos gracias a un sistema de marcas entre { } para que posteriormente Eyegrade incluya la información del examen en esos puntos. Las marcas utilizadas son:

- {{declarations}}: El script pondrá aquí las declaraciones necesitadas para generar el fichero LaTeX.
- {{subject}}, {{degree}}: nombre de la asignatura y titulación a la que pertenece. Cogido del fichero XML con las preguntas.
- {{title}}: el título del examen. Cogido del fichero XML con las preguntas.
- {{duration}}: duración del examen. Cogido del fichero XML con las preguntas.
- {{model}}: una letra representando el modelo del examen. Cada modelo tiene un orden diferente para preguntas y respuestas de cada pregunta.
- {{id-box(9,ID}}: reemplazado por un cuadro para que los estudiantes rellenen su ID. El número de dígitos y el texto que será puesto a la izquierda del cuadro, están especificados entre paréntesis.
- {{answer-table}}: reemplazado por la tabla en la que los estudiantes marcarán sus respuestas.
- {{questions}}: reemplazado por las preguntas del examen.

Esta plantilla es altamente reutilizable para diferentes exámenes y asignaturas.

Una vez que se dispone de ambos ficheros editados a mano, Eyegrade genera el examen en los diferentes modelos especificados. Los modelos de Eyegrade son reordenaciones concretas de preguntas y opciones para dificultar que los alumnos copien. Eyegrade creará una serie de ficheros con los nombres *exam-0.tex, exam-A.tex y exam-B.tex;* es decir, un fichero *.tex* por cada modelo indicado. Se pueden crear los siguientes modelos: 0, A, B, C, D, E, F, G y H. El modelo 0 de examen es un modelo especial en el que las preguntas no están reordenadas y la primera respuesta de cada pregunta es la correcta. Además, también se crea el fichero *exam.eye*, necesario para realizar las correcciones a través de Eyegrade.

Por tanto, el gráfico de este proceso es el siguiente:



2.2. Tecnologías de programación para la Web

Existen una gran variedad de entornos para el desarrollo de aplicaciones Web, entre los cuales podemos destacar PHP, J2EE, .NET, Django y Rails. Cualquiera de ellos proporciona herramientas suficientes para realizar este proyecto. A continuación se analizan brevemente estos entornos.

2.2.1. PHP (Hypertext Pre-processor)

PHP [4-6, 10, 11] es un lenguaje de programación de propósito general pero que se usa principalmente para el desarrollo de aplicaciones Web cuyo código se establece en el lado del servidor [7]. El código es interpretado por un servidor Web con un módulo de procesador de PHP que genera la página Web resultante. El cliente solicita una página Web cualquiera, del lado del servidor el código PHP es interpretado, y al cliente se le envía únicamente la página Web resultado de ese proceso (el código HTML). El cliente jamás se entera de las tareas ni de los códigos del lado del servidor.

Fue diseñado inicialmente para el desarrollo Web de contenido dinámico. Fue uno de los primeros lenguajes de programación en el lado del servidor que se podían añadir directamente en el documento HTML en lugar de llamar a un archivo externo que procese los datos. Con la evolución de PHP, se ha incluido una interfaz de línea de comandos que puede ser usada en aplicaciones gráficas independientes.

Características

- ✓ Es multiplataforma.
- ✓ Se distribuye con licencia de software libre, por lo que se presenta como una alternativa de fácil acceso para todos. Es publicado bajo la licencia PHP que es incompatible con la Licencia Pública General de GNU debido a las restricciones del uso del término *PHP* [8].
- ✓ Posee una amplia documentación en su sitio Web oficial y una gran comunidad de programadores. Todas las funciones del sistema están explicadas y ejemplificadas en un único archivo de ayuda.
- ✓ Hay una gran cantidad de bibliotecas que simplifican el desarrollo.

- ✓ Capacidad de expandir su potencial utilizando módulos (llamados *ext's* o extensiones).
- ✓ De fácil aprendizaje. Posee distintas especificaciones simplificadas, como por ejemplo la definición de variables primitivas, que se utilizan arrays PHP.
- ✓ Orientado al desarrollo de aplicaciones Web dinámicas con acceso a información almacenada en bases de datos.
- ✓ Es posible realizar conexiones con la mayoría de bases de datos que se utilizan en la actualidad, destaca su conectividad con MySQL y PostgreSQL.
- ✓ Permite aplicar técnicas de programación orientada a objetos.
- ✓ No requiere definición de tipos de variables aunque sus variables se pueden evaluar también por el tipo que estén manejando en tiempo de ejecución.
- ✓ El código fuente es ejecutado por el servidor y éste envía su resultado HTML al navegador, por lo que el código es invisible al navegador Web y al cliente. Esto hace que la programación en PHP sea segura y confiable.
- ✓ El tiempo de desarrollo es bastante rápido.
- ✓ Gran oferta de servicios de alojamiento en la Web que lo soportan, debido a su implementación simple.
- ✓ Tiene manejo de excepciones (desde PHP5).
- ✓ Si bien PHP no obliga a quien lo usa a seguir una determinada metodología a la hora de programar, aún haciéndolo, el programador puede aplicar en su trabajo cualquier técnica de programación o de desarrollo que le permita escribir código ordenado, estructurado y manejable. Un ejemplo de esto son los desarrollos que en PHP se han hecho del patrón de diseño Modelo Vista Controlador (MVC), que permiten separar el tratamiento y acceso a los datos, la lógica de control y la interfaz de usuario en tres componentes independientes.
- ✓ Debido a su flexibilidad ha tenido una gran acogida como lenguaje base para las aplicaciones WEB de manejo de contenido, y es su uso principal.

Inconvenientes

- Debido a que es un lenguaje interpretado, un script en PHP suele funcionar considerablemente más lento que su equivalente en un lenguaje de bajo nivel, sin embargo este inconveniente se puede minimizar con técnicas de caché tanto en archivos como en memoria. No es recomendable su uso en sitios Web que tengan muchas peticiones por segundo o cargas muy pesadas de acceso a BD.

- Las variables al no ser tipadas dificulta a los diferentes IDEs para ofrecer asistencias para el tipado del código, aunque esto no es realmente un inconveniente del lenguaje en sí.

- Existen muchas versiones de PHP con incompatibilidades entre sí

- En PHP 4 es difícil depurar los errores.

- Si bien la persistencia de datos existe al serializar manualmente o por medio de sesiones, no existe la persistencia de objetos lo cual puede llegar a ser una gran desventaja al programar OOP (programación orientada a objetos).

2.2.2. J2EE

J2EE (Java 2 Enterprise Edition) [12, 13, 16, 17] es una plataforma de programación para desarrollar y ejecutar software de aplicaciones en Java. Permite utilizar arquitecturas de N capas distribuidas: una capa de cliente o de presentación que proporciona el interfaz de usuario, una o más capas intermedias que proporcionan la lógica de negocio de la aplicación y una capa final con los sistemas de información que mantienen aplicaciones y bases de datos corporativas; y se apoya ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones.

Java EE tiene varias especificaciones de API, tales como JDBC, RMI, e-mail, JMS, Servicios Web, XML, etc. y define cómo coordinarlos. Java EE también configura algunas especificaciones únicas para Java EE para componentes. Éstas incluyen Enterprise JavaBeans, servlets, portlets (siguiendo la especificación de Portlets Java), JavaServer Pages y varias tecnologías de servicios Web. Ello permite al desarrollador crear una aplicación de empresa portable entre plataformas y escalable, a la vez que integrable con tecnologías anteriores. Otros beneficios

añadidos son, por ejemplo, que el servidor de aplicaciones puede manejar transacciones, la seguridad, escalabilidad, concurrencia y gestión de los componentes desplegados, significando que los desarrolladores pueden concentrarse más en la lógica de negocio de los componentes en lugar de en tareas de mantenimiento de bajo nivel.

Características

- ✓ Es multiplataforma [14]. Libertad de elección de plataformas de desarrollo y producción.
- ✓ Es de código libre.
- ✓ Su uso está muy extendido, lo que hace que cuente con el soporte de una amplia comunidad de desarrolladores.
- ✓ Utilización de herramientas de código libre que agilizan el desarrollo de software con J2EE y que permiten el funcionamiento en los distintos módulos de ejecución.
- ✓ Se dispone de una biblioteca llamada JDBC (Java Database Connectivity) que proporciona acceso a sistemas de bases de datos relacionales.
- ✓ Es un lenguaje intrínsecamente orientado a objetos.
- ✓ Es un lenguaje compilado.
- ✓ Tiene una gran potencia y velocidad, ya que se usan servlets y existe persistencia de objetos.
- ✓ Posibilidad de altas productividades en el desarrollo de las distintas tecnologías J2EE para la integración de aplicaciones corporativa e integración de sistemas existentes.
- ✓ Mayor escalabilidad al describir las características básicas de transacciones y desarrollando distintos tipos de componentes de aplicación J2EE con modelos flexibles de seguridad.
- ✓ El desarrollo de aplicaciones Web utilizando conceptos de la arquitectura J2EE que permiten la construcción de este tipo de aplicaciones.

Inconvenientes

- × De difícil aprendizaje [15], es un lenguaje complejo y de difícil desarrollo para los que no lo conozcan.
- × Tiempos de desarrollo mayores que con otras tecnologías
- × Escasa oferta de servicios de alojamiento en la Web debido al alto coste de implementación de la tecnología.
- × Poco práctico para pequeños proyectos simples con uso de base de datos sencillo.

2.2.3. Django

Django [27-33] es un framework de desarrollo Web, escrito en Python y siguiendo el patrón MTV (Model-Template-View).Es de código abierto.

El objetivo de Django es facilitar la creación de sitios Web complejos. Django pone énfasis en la reutilización, la conectividad y extensibilidad de componentes, el desarrollo rápido y el principio No te repitas (DRY, del inglés Don't Repeat Yourself). Todo el código está escrito en Python.

Al igual que Ruby on Rails, otro popular framework de código abierto, Django se usó en producción durante un tiempo antes de que se liberara al público.

Python [18-21, 24-26] es un lenguaje de programación dinámico y orientado a objetos. El principal objetivo que persigue este lenguaje es la facilidad, tanto de lectura, como de diseño. Python se desarrolla como un proyecto de código abierto, administrado por la Python Software Foundation. Fue creado por Guido van Rossum en el año 1990 y surgió como un sucesor del lenguaje de programación ABC, capaz de manejar excepciones e interactuar con el SO Amoeba.

Python se usa en grandes plataformas como: Youtube y Google.

Características

- ✓ Multiplataforma [22] y portable: El mismo código funciona en cualquier arquitectura, la única condición es que disponga del intérprete del lenguaje y se encuentre instalado el framework. No es necesario compilar el código una vez para cada arquitectura.

- ✓ Es de libre distribución.
- ✓ Existen muchas librerías de código abierto.
- ✓ Es sencillo de aprender, ya que Python es muy parecido a hablar en lenguaje natural (inglés) y permite realizar muchas acciones básicas y programas sencillos.
- ✓ Soporta varias bases de datos.
- ✓ Python es un lenguaje de programación Multiparadigma, permite varios estilos: programación orientada a objetos, programación estructural y funcional.
- ✓ Python es un lenguaje interpretado de scripts lo cual permite que sea modular y que sea ejecutado en diferentes intérpretes para diferentes arquitecturas computacionales.
- ✓ El código se ejecuta en tiempo real y no es necesario compilarlo de antemano. Esto hace que su desarrollo sea rápido.
- ✓ Sirve para hacer un prototipo rápido del programa previo a su final desarrollo.
- ✓ Gran soporte e integración con otros lenguajes y herramientas.
- ✓ Hay un intérprete interactivo para Python que permite hacer pruebas rápidas de código.
- ✓ Las variables no tienen que ser definidas como de un solo tipo lo que facilita la creación y reduce las palabras necesarias para escribir un programa.

Inconvenientes

- ✗ La documentación de Python es más escasa que en otras tecnologías, tanto en español como en inglés ya que su comunidad de programadores es más pequeña.
- ✗ Lentitud [22]: Los programas interpretados son más lentos que los compilados. Sin embargo los programas interpretados suelen ser cortos, en los que la diferencia es inapreciable.

- Baja oferta de servicios de alojamiento en la Web que lo soportan debido a que es muy complejo implementar esta tecnología en Web.

- Es problemático distribuir el trabajo en grupos de trabajo como lo es con lenguajes de programación como Java.

- Conforme se crean aplicaciones más complejas es más complicado escribir el código.

- No es recomendable para hacer programas complejos, robustos, rápidos y confiables.

- Toda la sintaxis está basada en el acomodo de espacios e indentación.

2.2.4. Rails

Ruby on Rails [35, 36, 38, 40], también conocido como RoR o Rails, es un framework de aplicaciones Web de código abierto escrito en el lenguaje de programación Ruby, siguiendo 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. Ruby es un lenguaje de programación relativamente nuevo. Es orientado a objetos y es principalmente una combinación de los lenguajes de programación Perl y Smalltalk. Ruby permite la meta-programación, 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 solo 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)

Características

- ✓ Es multiplataforma [39].
- ✓ Es de código abierto y libre.
- ✓ Proporciona una buena integración con bases de datos.
- ✓ Es orientado a objetos.
- ✓ Tiempo de programación: con Ruby on Rails es posible terminar el proyecto en semanas que de lo contrario tomar meses para ser completado. Esto hace que tome mejor posición respecto a otras alternativas en precio-tiempo.
- ✓ Buena oferta de servicios de alojamiento en la Web, existiendo ya algunos dedicados especialmente a Ruby on Rails.

Inconvenientes

- ✗ Su comunidad de soporte no es tan grande como las comunidades de otros lenguajes [37]. Por ello tampoco cuenta con tanta documentación.
- ✗ Requiere un tiempo de aprendizaje.
- ✗ Ruby es un lenguaje interpretado, aunque existen utilidades que permiten compilarlo.
- ✗ Es un lenguaje lento, pudiendo ser 20 veces más lento que Java.

- Sus actualizaciones llevan más tiempo que en otras tecnologías. Llegando incluso a ser la última versión de Ruby no compatible con la última versión de Rails.

2.2.5. .NET

.NET [41, 42, 46-49] es una plataforma de Microsoft de desarrollo y ejecución de aplicaciones que hace énfasis en la independencia de plataforma de hardware. Facilita el proceso de construcción de programas multipropósito, tanto en entornos Web como entornos móviles.

Propone ofrecer una manera rápida, segura y robusta de desarrollar aplicaciones, permitiendo una integración más rápida y ágil entre empresas.

.NET se incluye en Windows Server 208, Windows Vista y Windows 7. También puede ser instalada en Windows XP, Windows Server 2000 y Windows Server 2007. Microsoft pretende permitir su instalación en sistemas Windows 95, 98 y ME, agendas electrónicas con Pocket PC y la consola X-box en Web TV. Existe una versión reducida de .NET Framework disponible para Windows Mobile.

Microsoft, HP e Intel trabajaron conjuntamente para estandarizar el lenguaje de programación C# y una infraestructura de lenguaje común (CLI). ECMA e ISO solicitaron que la implementación estuviera bajo términos no discriminatorios y que se dejaran estas patentes disponibles. Sin embargo, esto no se aplicó para Windows Forms, ADO.NET y ASP.NET.

El Proyecto Mono emprendido por Ximian pretende realizar la implementación de la norma para varios sistemas operativos adicionales bajo el marco del código abierto.

Bajo esta norma, se definió una biblioteca de clases base (BCL), que define un conjunto funcional mínimo. Está formada por 4 grupos:

- ASP.NET y Servicios Web XML.
- Windows Forms.
- ADO.NET
- .NET

Diagrama básico de la Biblioteca de Clases Base. Fuente: [42]

La infraestructura de lenguaje común (CLI) permite el desarrollo en más de 20 lenguajes de programación, entre ellos C#, Visual Basic .NET, Delphi, C++, F#, J#, Perl, Python, Fortran y Prolog. Empresas e instituciones están desarrollando lenguajes de programación adicionales para la plataforma (Lexico para hispanoparlantes, ANSI C de la Universidad de Princeton, NET COBOL de Fujitsu, Delphi de Borland, PowerBuilder de Sybase).

Estructura interna del entorno de ejecución en lenguaje común. Fuente: [42]

La herramienta de desarrollo compila el código fuente de cualquiera de los lenguajes soportados por .NET en un código intermedio, el CIL (Common Intermediate Language) antes conocido MSIL (Microsoft Intermediate Language), similar al BYTECODE de Java. Para generarlo, el compilador se basa en la especificación CLS (Common Language Specification) que determina las reglas necesarias para crear el código MSIL compatible con el CLR.

Para ejecutarse se necesita un segundo paso, un compilador JIT (Just-In-Time) es el que genera el código máquina real que se ejecuta en la plataforma del cliente. De esta forma se consigue con .NET independencia de la plataforma de hardware. La compilación JIT la realiza el CLR a medida que el programa

invoca métodos. El código ejecutable obtenido se almacena en la memoria caché del ordenador, siendo recompilado de nuevo solo en el caso de producirse algún cambio en el código fuente.

Un ensamblado es un compilado EXE o DLL que contiene código CIL que se genera desde los diferentes lenguajes .NET, y que es ejecutado por el CLR. Puede contener una o varias clases al igual que uno o varios namespaces. Los ensamblados pueden tener diferentes decoradores que definen el entorno de ejecución de los mismos COM+, DCOM, Remoting, etc.

Características

- ✓ Interoperabilidad Multilenguaje: .NET soporta aplicaciones con componentes en múltiples lenguajes de programación, lo que permite integrar desarrolladores de distintos perfiles. Integración total de LINQ (Language Integrated Query) y del reconocimiento de datos. Permite escribir código en idiomas habilitados para LINQ.

- ✓ Amplia documentación de ayuda (herramientas, debbuggers, editores) incluida en la IDE.

- ✓ Ofrece soporte de ayuda.

- ✓ Biblioteca de clases base que pueden ser utilizadas por cualquier lenguaje, que incluye, entre otras, cifrado de datos, transmisión de datos por distintos medios (XML, TCP/IP), administración de componentes Web, herramientas de seguridad, interacción con otras aplicaciones y manejo de excepciones.

- ✓ Es lenguaje compilado, lo que proporciona un alto rendimiento respecto a versiones interpretadas.

- ✓ Independencia del hardware gracias a la compilación a código máquina.

- ✓ De rápido aprendizaje gracias a su interoperabilidad, documentación y soporte de ayuda.

- ✓ Movilidad. Las aplicaciones pueden ser desplegadas en una amplia variedad de dispositivos. Las aplicaciones Web creadas pueden ser accedidas desde cualquier navegador en cualquier sistema operativo.

- ✓ .NET da respaldo para ejecutar código no seguro.

- ✓ Puede ser orientado o no orientado a objetos dependiendo del lenguaje de programación utilizado.

- ✓ Compatibilidad con el protocolo Web para generar servicios WCF, como AJAX, JSON, POX, RSS, ATOM y distintos estándares WS-*.

- ✓ Con el proyecto Mono se permite ejecutar el servidor en GNU, Linux, MAC OS X, BSD y Solaris. Este proyecto es de código libre.

- ✓ Se encuentran disponibles controles, componentes y bibliotecas de clase adicionales creadas por otras empresas e instituciones. Algunas de ellas son de software libre, distribuible bajo la licencia GPL.

Inconvenientes

- × No es multiplataforma [43-45]. El servidor solo puede ser desarrollado y ejecutado en sistemas Windows.

- × Aunque el proyecto Mono permite ejecutar el servidor en otros sistemas operativos, se encuentra en versión de desarrollo y no implementa todas las funciones de .NET. Puede ocurrir que los programas implementados no funcionen con Mono.

- × Si una aplicación .NET está desarrollada en múltiples lenguajes, su mantenimiento es costoso, puesto que se necesitan expertos en varios lenguajes para entenderla y mantenerla.

- × Es de código cerrado, no hay licencias libres. Por ello representa un alto costo desarrollar en .NET.

- × Menos portable que otros lenguajes.

2.2.6. Comparativa

Veamos por tanto una comparativa de estas tecnologías [9, 23].

Por un lado, J2EE es la tecnología más desarrollada de las cinco, permite realizar aplicaciones Web de gran complejidad y con un acceso rápido a las bases de datos, minimizando los tiempos de respuesta.

Tanto PHP, como J2EE y .NET poseen una gran comunidad de programadores y multitud de librerías y documentación que facilitan el trabajo. Python posee también multitud de librerías, aunque la documentación es más escasa ya que su comunidad de programadores es más pequeña. Para Django cada día va saliendo

más documentación y va ganando adeptos, por lo que su comunidad va creciendo.

Los lenguajes más sencillos de aprender son PHP y Python, siendo J2EE el más complicado. Además .NET permite elegir el lenguaje con el que programar.

La oferta de servidores con PHP, Ruby y .NET es amplia, mientras que con Python y J2EE es reducida. En nuestro caso, la aplicación Web se va a alojar en un servidor propio de la universidad, por lo que este aspecto no lo tendremos en cuenta.

Respecto a Ruby, se trata de un lenguaje muy nuevo, con relativamente poca documentación y soporte, actualizaciones que tardan en salir e incompatibilidades entre la última versión del lenguaje y la última versión de Rails, su framework.

Todas las tecnologías analizadas son de código abierto y multiplataforma excepto .NET.

A continuación, una tabla comparativa de las tecnologías:

Características	**PHP**	**J2EE**	**.NET**	**Django**	**Rails**
Multiplataforma	✓	✓	✘	✓	✓
Código abierto	✓	✓	✘	✓	✓
Amplia documentación	✓	✓	✓	Aumentando	✘
Bibliotecas/ librerías	✓	✓	✓	✓	✘
Amplia comunidad de soporte	✓	✓	✓	Aumentando	✘
Fácil aprendizaje	✓	✘	Según lenguaje utilizado	✓	✘
Bases de datos	✓	✓	✓	✓	✓
Orientado a Objetos	✓	✓	Según lenguaje utilizado	✓	✓
Compilado/ Interpretado	Interpretado	Compilado	Compilado	Interpretado	Interpretado
Tiempo de desarrollo	Bajo	Alto	Medio	Medio-Bajo	Bajo
Oferta de alojamiento en la Web	Amplia	Escasa	Media	Escasa	Amplia

2.2.7. Conclusiones del análisis tecnológico

Tras analizar las características y capacidades de cada lenguaje, además de la magnitud del proyecto y el conocimiento del programador, se han extraído las siguientes conclusiones:

- Se ha considerado que J2EE es demasiado complejo y extenso para la finalidad de este proyecto. Requeriría mucho tiempo desarrollar la aplicación Web en este lenguaje y no necesitamos tantas características como las que ofrece.

- Ruby es demasiado nuevo y con relativamente poco soporte. Probablemente se quedaría corto para nuestras pretensiones. Además requeriría su aprendizaje y hay relativamente poca ayuda en la Web en la que apoyarse.

- .NET no es de código libre, y tampoco multiplataforma. Se desean reducir los costes y permitir al desarrollador utilizar cualquier sistema operativo.

- Entre PHP y Django, se ha decidido utilizar Django ya que el tiempo de respuesta es mejor y Python es un lenguaje más sencillo de aprender.

- Otros motivos por los que se ha elegido Django es que el programador ya conocía el lenguaje Python, y que Eyegrade también está implementado en Python, por lo que se facilita su integración.

2.3. Estudio en profundidad de Django

Se ha elegido este framework para el desarrollo de la aplicación Web, por lo que vamos a profundizar en él.

Django [27-33] proporciona una serie de características que facilitan el desarrollo rápido de páginas orientadas a contenidos. Por ejemplo, en lugar de requerir que los desarrolladores escriban controladores y vistas para las áreas de administración de la página, Django proporciona una aplicación incorporada para administrar los contenidos, que puede incluirse como parte de cualquier página hecha con Django y que puede administrar varias páginas hechas con Django a partir de una misma instalación; la aplicación administrativa permite la creación, actualización y eliminación de objetos de contenido, llevando un registro de todas las acciones realizadas sobre cada uno, y proporciona una interfaz para

administrar los usuarios y los grupos de usuarios (incluyendo una asignación detallada de permisos).

La distribución principal de Django también aglutina aplicaciones que proporcionan un sistema de comentarios, herramientas para sindicar contenido vía RSS y/o Atom, "páginas planas" que permiten gestionar páginas de contenido sin necesidad de escribir controladores o vistas para esas páginas, y un sistema de redirección de URLs.

Características

- ✓ Un mapeador objeto-relacional.
- ✓ Aplicaciones "enchufables" que pueden instalarse en cualquier página gestionada con Django.
- ✓ Soporte de bases de datos con una API robusta. Se recomienda usar PostgreSQL, pero también son soportadas MySQL y SQLite 3. Se encuentra en desarrollo un adaptador para Microsoft SQL Server. Una vez creados los modelos de datos, Django proporciona una abstracción de la base de datos a través de su API que permite crear, recuperar, actualizar y borrar objetos. También es posible que el usuario ejecute sus propias consultas SQL directamente. En el modelo de datos de Django, una clase representa una tabla en la base de datos y las instancias de esta serán las filas en la tabla.
- ✓ Soporte de servidores Web. Incluye un servidor Web liviano para realizar pruebas y trabajar en la etapa de desarrollo. En la etapa de producción, sin embargo, se recomienda Apache 2 con mod_python. Aunque Django soporta la especificación WSGI, por lo que puede correr sobre una gran variedad de servidores como FastCGI o SCGI en Apache u otros servidores (particularmente Lighttpd).
- ✓ Un sistema incorporado de "vistas genéricas" que ahorra tener que escribir la lógica de ciertas tareas comunes.
- ✓ Un sistema extensible de plantillas basado en etiquetas, con herencia de plantillas.
- ✓ Un despachador de URLs basado en expresiones regulares.
- ✓ Un sistema "middleware" para desarrollar características adicionales; por ejemplo, la distribución principal de Django incluye componentes middleware que proporcionan memoria caché, compresión de la salida, normalización de URLs, protección CSRF y soporte de sesiones.

- ✓ Soporte de internacionalización, incluyendo traducciones incorporadas de la interfaz de administración.

- ✓ Documentación incorporada accesible a través de la aplicación administrativa (incluyendo documentación generada automáticamente de los modelos y las bibliotecas de plantillas añadidas por las aplicaciones).

Arquitectura

Aunque Django está fuertemente inspirado en la filosofía de desarrollo Modelo Vista Controlador, sus desarrolladores declaran públicamente que no se sienten especialmente atados a observar estrictamente ningún paradigma particular, y en cambio prefieren hacer "lo que les parece correcto". Como resultado, por ejemplo, lo que se llamaría "controlador" en un "verdadero" framework MVC, se llama en Django "vista"; y lo que se llamaría "vista", se llama "plantilla". Para Django la Vista describe qué datos serán presentados y no cómo se verán los mismos. Aquí es donde entran en juego las plantillas, que describen cómo los datos son presentados.

Gracias al poder de las capas *Mediador* y *Fundación*, Django permite que los desarrolladores se dediquen a construir los objetos Entidad y la lógica de presentación y control para ellos.

- ***Presentación***

  Aquí se maneja la interacción entre el usuario y el ordenador. En Django, esta tarea la realizan el motor de plantillas y el cargador de plantillas que toman la información y la presentan al usuario (vía HTML, por ejemplo). El sistema de configuración de URLs es también parte de la capa de *Presentación*.

- ***Control***

  En esta capa reside el programa o la lógica de aplicación en sí. En Django son representados por las vistas y los manipuladores. La capa de *Presentación* depende de ésta y a su vez ésta lo hace de la capa de dominio.

- ***Mediador***

  Es el encargado de manejar la interacción entre el subsistema *Entidad* y *Fundación*. Aquí se realiza el mapeo objeto-relacional a cargo del motor de Django.

- ***Entidad***

  El subsistema *Entidad* maneja los objetos de negocio. El mapeo objeto-relacional de Django permite escribir objetos de tipo *Entidad* de una forma fácil y estándar.

- ***Fundación***

  La principal tarea del subsistema *Fundación* es la de manejar a bajo nivel el trabajo con la base de datos. Se provee soporte a nivel de *Fundación* para varias bases de datos y otras están en etapa de prueba.

Funcionamiento

Teniendo la arquitectura en cuenta, veremos a grandes rasgos como se procesa una petición.

Cuando Django recibe una petición, lo primero que se hace es crear un objeto *HttpRequest* que la representa y servirá como abstracción para trabajar sobre diferentes servidores.

Luego se realiza la resolución de la URL. Esto consiste en seleccionar la función de la vista (a partir de la URL especificada en la petición) que participará en la creación de la respuesta.

Una vez que hemos resuelto qué función resolverá la URL especificada, se invoca a la función de la vista con el objeto *request* como primer parámetro. Aquí se realiza el trabajo pesado como: consultas a la BD, carga de plantillas y generación de HTML. Se devuelve un objeto *HTTP response* o una excepción.

3. REQUISITOS

La aplicación Web debe cumplir los siguientes requisitos:

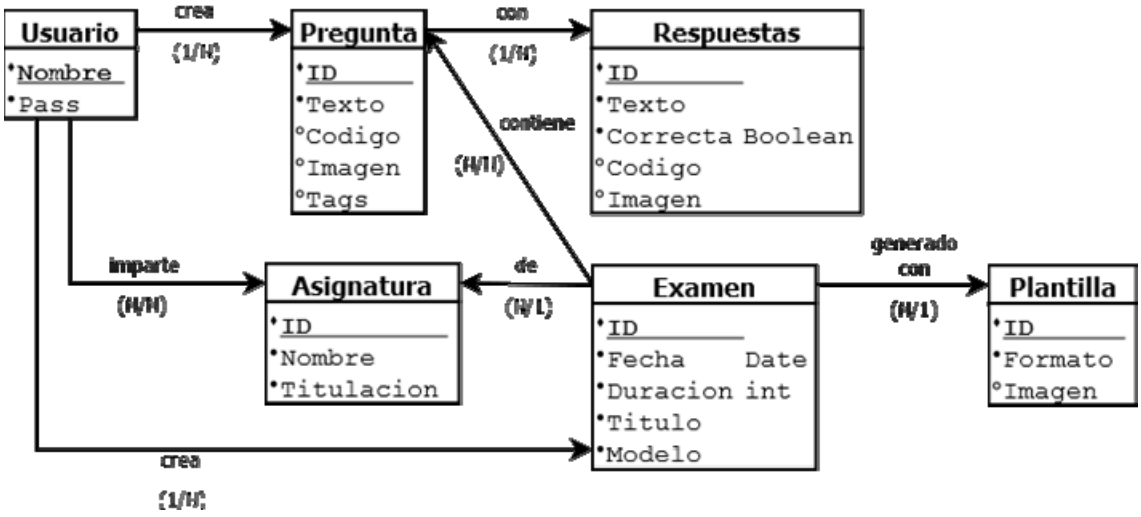
- El acceso debe quedar limitado a los usuarios registrados.
- No debe existir un auto-registro. Los registros deben realizarse por el administrador para asegurar la privacidad de los documentos a los que puede acceder esta Web.
- Han de realizarse pruebas de seguridad. No se puede acceder a los archivos si no estás autenticado o si no eres el autor de dichos archivos.
- Han de realizarse pruebas de consistencia. Las bases de datos deben ser consistentes, y no se pueden crear errores.
- Debe permitirse la creación y modificación de exámenes y preguntas. También debe permitirse el borrado de exámenes.
- Debe permitirse la descarga de ficheros de exámenes, tanto ficheros LaTeX como PDF.
- Debe establecerse una red social en la que los usuarios puedan ver los perfiles de otros usuarios y puedan ver un correo de contacto.
- Debe permitirse la edición del propio perfil.

4. DISEÑO DE LA APLICACIÓN

Una vez que tenemos elegida y analizada la tecnología con la que vamos a desarrollar la aplicación Web, y especificados los objetivos a conseguir, pasamos a desarrollar el diseño de la aplicación.

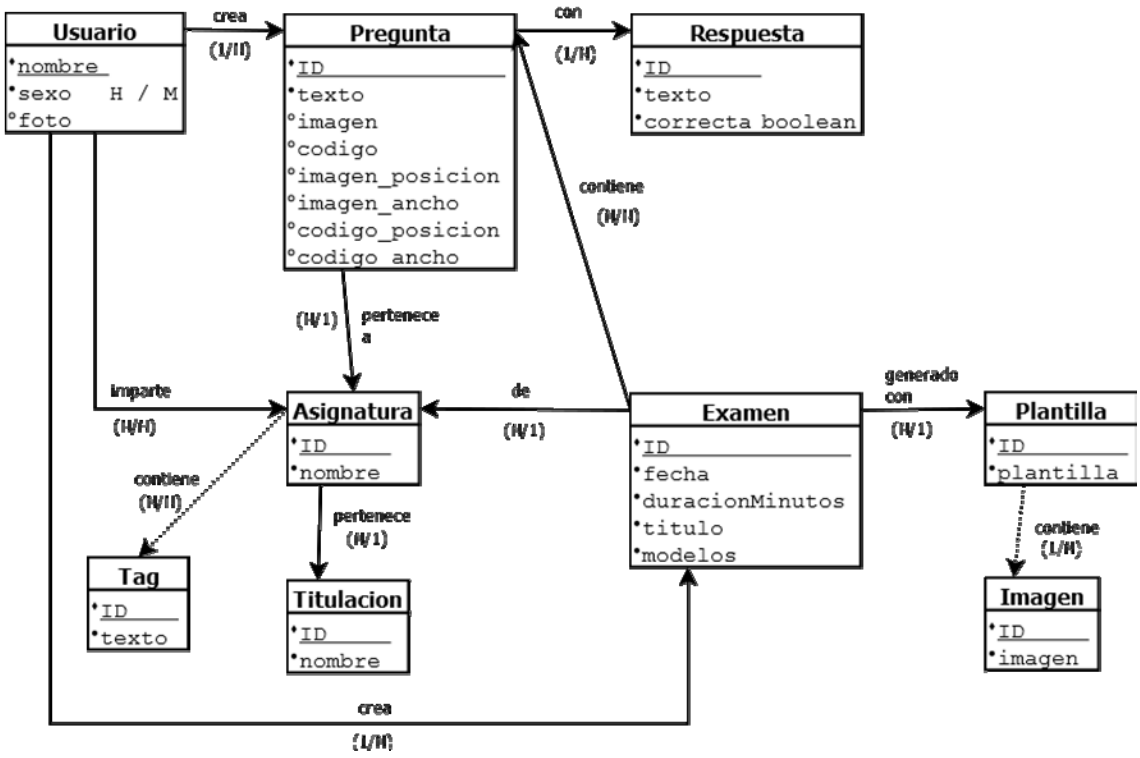
4.1. Modelo de datos

Primero vamos a centrarnos en la estructura de la base de datos. En un principio la estructura era la siguiente:



En este modelo tenemos como tabla principal la tabla *Usuario*, que contiene el nombre y contraseña. Este usuario imparte unas asignaturas y crea una serie de preguntas que contienen varias respuestas. Cada examen, pertenece a una asignatura, y obviamente, pueden existir varios exámenes para la misma asignatura. Cada examen es generado por una plantilla, que puede contener una imagen.

Una vez que se comenzó con la implementación, la estructura tuvo que ser modificada, incluyendo campos y tablas necesarios y eliminando aquellos que eran superfluos para agilizar la conexión y las consultas y ahorrar espacio en disco. De este modo, la estructura final ha sido:



En la tabla *Usuario* se ha eliminado el campo de contraseña, ya que al realizar la implementación de la vista de autenticación, era necesario guardarla en la base de datos. Tras las mejoras en autenticación realizadas (y comentadas más adelante en el apartado *5. IMPLEMENTACIÓN*), este campo era innecesario y fue borrado. Además se le añadieron los campos de sexo y foto. Esto es debido a que Django proporciona varios campos de "perfil" para cada usuario, pero no en concreto una fotografía o su sexo. Necesitamos saber el sexo del usuario ya que se ha implementado que los usuarios que no dispongan de fotografía subida, se muestre una foto por defecto según su sexo.

La tabla *Pregunta* también ha sido altamente modificada. Se han añadido los campos *imagen_posicion, imagen_ancho, codigo_posicion, codigo_ancho*. Esto es debido a que se decidió que el usuario pudiera establecer la posición y el ancho de las imágenes y código.

En la tabla *Respuesta* se han eliminado los campos de código e imagen ya que Eyegrade solo contempla la opción de incluir estos campos en la pregunta.

Debido a que las titulaciones pueden contener varias asignaturas, se decidió eliminar dicho campo de la tabla de asignaturas y crear una tabla nueva para ellas. De este modo, evitamos datos duplicados en la base de datos.

Se ha añadido una tabla nueva, llamada *Tag*, para que se puedan asignar etiquetas a las asignaturas.

Además, se vio la posibilidad de que una plantilla contuviera varias imágenes, y por ello, se eliminó el campo de la tabla *Plantilla* y se creó una tabla nueva para las plantillas.

4.2. Diseño de las vistas

Ahora tenemos que decidir cuáles serán las vistas de nuestra aplicación. La aplicación tendrá muchas vistas, voy a describir las más importantes.

La primera y una de las más importantes será la vista de autenticación. Nadie debe poder acceder al resto de vistas sin haberse autenticado primero en esta vista. Solicitará usuario y contraseña para acceder a la Web.

Una vez autenticado, se verá la vista principal, donde se podrá acceder al resto de vistas.

La vista de perfil permite ver los datos del perfil de la persona autenticada, como nombre, asignaturas impartidas, contacto, fotografía y fecha de alta en el servicio Web. En esta vista se podrá acceder a otras vistas donde modificar la contraseña de acceso, la imagen, y el resto de datos del perfil.

La vista de asignaturas impartidas muestra una lista con las asignaturas impartidas por el profesor. Esta lista puede ser modificada si accedemos a la vista de modificar asignaturas.

La vista *mis exámenes* permite el acceso a las vistas más significativas de la aplicación Web. Tiene acceso a las vistas de ver preguntas, ver exámenes, ver plantillas, crear una pregunta, crear un examen y subir plantillas.

La vista de ver preguntas muestra una lista con las preguntas creadas por el usuario y pueden ser filtradas por asignaturas y posteriormente modificadas. Si la pregunta está siendo usada por algún examen, la modificación provocará que la pregunta se “duplique”, manteniendo la versión anterior y la nueva modificación para evitar inconsistencias con los exámenes.

La vista de ver exámenes muestra todos los exámenes creados, que pueden también ser filtrados por la asignatura a la que pertenecen. Permite modificarlos, eliminarlos y descargar los archivos relativos a él, tales como los archivos LaTeX generados por Eyegrade, los PDF de cada modelo, etc.

La vista de ver plantillas muestra una lista con las plantillas subidas por el usuario y permite borrarlas y modificar las imágenes relacionadas con cada plantilla.

La vista de crear una pregunta permite crear una pregunta, donde se piden campos como asignatura, código, imagen, texto de la pregunta, respuestas y cuál es la opción correcta de la pregunta. Algunos de estos campos son obligatorios y otros opcionales. Las asignaturas que pueden ser elegidas para la pregunta son las asignaturas impartidas por el profesor.

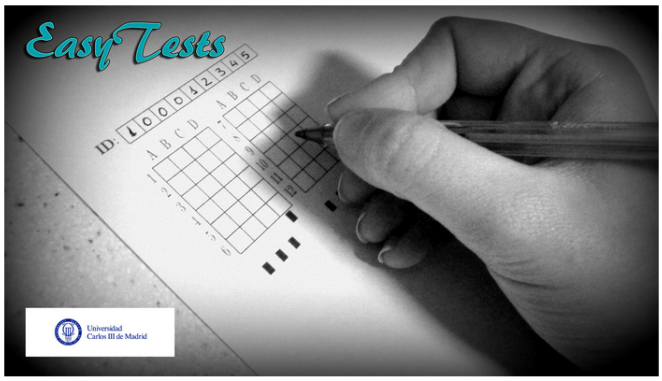
En la vista de crear examen se muestra primero un formulario para rellenar la información básica del examen, como fecha, duración, título, asignatura, modelos que desean generarse y la plantilla que se desea usar. Al igual que en las preguntas, las asignaturas que pueden seleccionarse son las impartidas por el profesor. Las plantillas que pueden usarse son las subidas por el usuario o una plantilla por defecto que se proporciona por nuestra aplicación para aquellos usuarios que no tengan manejo con las plantillas LaTeX. Una vez rellenados estos campos, se pasa a una vista donde aparecen todas las preguntas creadas para la asignatura elegida. En esta vista se selecciona qué preguntas pertenecerán al examen. Una vez pulsado el botón "*Crear*" se generarán todos los archivos del examen y aparecerá otra vista donde el usuario puede acceder a la descarga de dichos ficheros o crear otro examen.

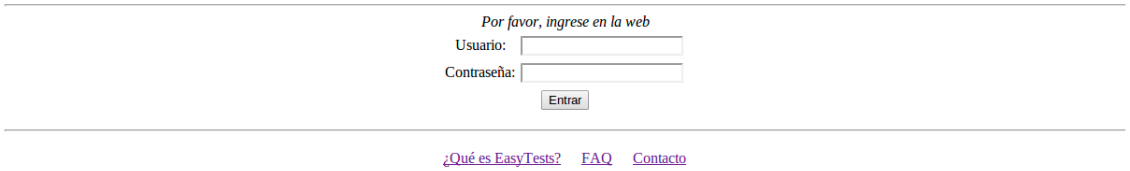
En la vista de subir plantillas se permite subir un fichero LaTeX de una plantilla o subir una imagen. Al seleccionar subir imagen, aparece otra vista donde se selecciona una de las plantillas subidas por el usuario y se sube el archivo de la imagen. De este modo todas las imágenes quedan relacionadas con una plantilla.

Y por último, la vista social que muestra los perfiles de todos los profesores que forman parte de la aplicación Web. También permite filtrar esta lista según las asignaturas impartidas.

4.3. Interfaz de usuario

Ya tenemos el diseño de las vistas básicas, ahora pasamos a la disposición gráfica de los componentes de la interfaz.





La página de autenticación tendrá la imagen o logo de nuestra aplicación, que se llamará EasyTests[1], en la parte superior central. Por debajo del logo aparecerá el formulario con el usuario y contraseña para acceder a la aplicación. En la parte más baja aparecerán los enlaces *"¿Qué es EasyTests?", "FAQ" y "Contacto".*

Una vez que ya se ha accedido a la aplicación, el resto de vistas tendrán una apariencia común. En la parte superior aparecerá una nueva versión del logo de EasyTests situado a la izquierda y a la derecha aparecerá "Bienvenid@, *usuario*", donde usuario es sustituido por el nombre del usuario que ha accedido. Justo debajo de la frase de

[1] EasyTests no es un nombre registrado (en la fecha de creación de esta memoria).

bienvenida aparece la última conexión realizada por este usuario y permite cerrar la sesión.

El cuerpo de la página está dividido en dos secciones. Una más pequeña en la parte izquierda, que contiene los enlaces a las vistas más generales, y una más grande en la parte derecha, donde se muestra el contenido de cada vista.

Como en la vista de autenticación, en la parte inferior aparecerán los enlaces *"¿Qué es EasyTests?", "FAQ" y "Contacto".*

5. IMPLEMENTACIÓN

Para comenzar a implementar este proyecto, primero debemos crear un proyecto en Django. Ejecutamos en un terminal:

```
django-admin.py startproject proyectoElena
```

Esto crea el siguiente árbol de carpetas y archivos:

```
proyectoElena/
    manage.py
    proyectoElena/
        __init__.py
        settings.py
        urls.py
        wsgi.py
```

5.1. Modo de depuración

Ahora que tenemos el proyecto creado, vamos a utilizar el modo de depuración, ya que nos permite, entre otras cosas:

- Cuando un error 404 es alcanzado, Django muestra una página especial para errores 404. Cuando se llama a una url que no coincide con los patrones especificados en el archivo urls.py, Django muestra una página con información para el desarrollador. Contiene la url solicitada, y los patrones que se han comparado.

- Cualquier excepción en nuestra aplicación de Django mostrará una página especial de Django que muestra diversa información. Desde en qué línea de qué archivo se alcanzó la excepción, hasta los valores de las variables locales y los detalles de la petición o respuesta.

- Todas las peticiones a la base de datos son almacenadas en un archivo para poder revisarlas.

- Guardar información extra sobre cada plantilla de Django.

El modo de depuración viene activado por defecto cuando creamos el proyecto de Django. Más adelante, veremos cómo se desactiva para lanzar a nivel global nuestra aplicación Web.

5.2. Interfaz de administrador

Como ya hemos comentado, Django facilita una interfaz de administrador, a través de la cual podemos gestionar nuestra aplicación

y tenemos acceso a los modelos para poder modificarlos. Para activar esta interfaz, primero hay que realizar unos cuantos cambios en el fichero de configuración, "/proyectoElena/proyectoElena/settings.py":

1. Añadir 'django.contrib.admin' en la sección INSTALLED_APPS.
2. Asegurarse que la sección INSTALLED_APPS contiene 'django.contrib.auth', 'django.contrib.contenttypes', 'django.contrib.messages' y 'django.contrib.sessions'.
3. Asegurarse que MIDDLEWARE_CLASSES contiene 'django.middleware.common.CommonMiddleware', 'django.contrib.messages.middleware.MessageMiddleware', 'django.contrib.sessions.middleware.SessionMiddleware' y 'django.contrib.auth.middleware.SessionMiddleware'.

La sección INSTALLED_APPS es una lista de cadenas de caracteres indicando todas las aplicaciones que están habilitadas en esta instalación de Django. Cada cadena de caracteres debe ser una ruta total de Python a un paquete Python que contiene una aplicación de Django. Y la sección MIDDLEWARE_CLASSES contiene una lista de clases middleware para usar.

Después, debemos ejecutar "python manage.py syncdb". Este comando, instalará las tablas extra de la base de datos que la interfaz de administrador utiliza. Y por último debemos descomentar las siguientes líneas del archivo "/proyectoElena/proyectoElena/urls.py/":

```
from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns (",
#…
        (r'^admin/', include(admin.site.urls)),
        #…
)
```

5.3. Apps de Django

En Django, las apps son aplicaciones Web que realizan algún cometido y un proyecto es una colección de apps. Creamos una para nuestro proyecto:

```
python manage.py startapp principal
```

5.4. Modelos

Una vez tenemos la base de nuestro proyecto en Django, estableceremos los modelos, según la estructura de la base de datos que se explicó en el apartado *4.1. Modelo de datos*. Para ello, modificamos el archivo "/proyectoElena/principal/models.py". Un ejemplo de modelo:

```
class Asignatura(models.Model):
        nombre = models.CharField(max_length=200)
        titulacion = models.ForeignKey(Titulacion)
        profesores = models.ManyToManyField(Usuario, blank=True,
                        null=True)
        tags = models. ManyToManyField(Tag, blank=True, null=True)

        def __unicode__(self):
                return u'%s (%s)' (self.nombre, self.titulacion)
```

Tenemos que señalar al proyecto de Django que la app *principal* está instalada. Para ello lo incluimos en el fichero "proyectoElena/proyectoElena/settings.py" de este modo:

```
INSTALLED_APPS = (
        'django.contrib.auth',
        'django.contrib.contenttypes',
        'django.contrib.sessions',
        #...
        #...
        'principal',
)
```

Ahora pasaremos este modelo a MySQL, el servidor de bases de datos elegido para este proyecto. Primeramente, para poder utilizar este servidor, necesitamos descargar el paquete MySQLdb. Una vez descargado e instalado, ejecutamos en consola:

```
python manage.py sql principal
```

Esto genera el código SQL necesario para crear las tablas en la base de datos que hemos especificado en nuestro modelo. Del modo que más cómodo nos sea (mediante consola o interfaz gráfica) ejecutaremos este código SQL en MySQL. Al ejecutarlo, se crearán en la base de datos todas las tablas y campos necesarios para nuestro proyecto.

5.5. Vistas

Y ahora vamos a pasar a crear las vistas, tal y como describimos en el apartado *4.2. Diseño de las vistas*. Un ejemplo de una vista sencilla:

```
def principal(request):
        if request.user.is_authenticated():
                return render_to_response('principal.html', locals(),
                                context_instance=RequestContext(request))
        error = False
```

```
        return render_to_response('login.html', locals(),
                                  context_instance=RequestContext(request))
```

La primera línea, "def principal(request)", define la función (vista) que tomará como parámetro la petición (*request*) HTTP.

Las siguientes líneas forman parte de la autenticación en la aplicación, se explicarán en el apartado *5.9. Vista de autenticación*.
La línea "error = False", establece una variable con valor *False*, en este caso se trata de un error que puede darse en esta vista.

La última línea, "return render_to_response('login.html', locals(), context_instance=RequestContext(request))", indica a Django que se debe devolver la plantilla "login.html" con las variables locales que se encuentran definidas en la función.

5.6. Manejo de errores

Como en toda función, pueden presentarse errores. Veamos cómo tratar estos errores en las vistas.

Los errores en Django se manejan mediante el mecanismo de gestión de excepciones de Python, pero a su vez, Django proporciona excepciones ya programadas para lanzar los códigos de error más habituales de HTTP. Un ejemplo:

```
from django.http import Http404

def detalle(request, asignaturaID):
        try:
                asignatura = Asignatura.objects.get(id=asignaturaID)
        except Asignatura.DoesNotExist:
                raise Http404
        return render(request, 'AsignaturaError.html', {'asignatura':
                        asignatura})
```

Como se ha visto en el punto *5.1. Modo de depuración*, cuando está activado el modo de depuración, Django muestra una página especial para errores HTTP 404. Si el modo de depuración no estuviera activado, se mostraría la plantilla "404.html", como veremos en el punto *6.2. Poniendo en marcha EasyTests.*

5.7. Manejo de URL's

Para que una vista sea llamada al solicitar una página, el servidor Web debe asociar rutas HTTP a cada recurso (en nuestro caso, las vistas). Para ello debemos editar el archivo "proyectoElena/proyectoElena/urls.py". Un ejemplo:

```
from proyectoElena.views import principal

urlpatterns = patterns(‘’,
        url(r’^admin./’, include(admin..site.urls)),
        (r'^perfil/$', perfil),
        (r'^$', principal),
)
```

Como se puede ver en la línea “(r'^perfil/$', perfil)”, cada asociación se especifica mediante un patrón y un identificador del recurso a ejecutar. El servidor recorrerá este archivo en riguroso orden, y cuando la ruta HTTP coincida con alguno de los patrones, se ejecutará el recurso asociado a dicha ruta. Por eso el patrón “(r'^$', principal)” se sitúa en último lugar. Este patrón indica que cualquier dirección solicitada debe ejecutar el recurso “principal”. Al estar situado al final, sólo se ejecuta si la ruta solicitada no coincide con ninguna de las anteriores.

5.8. Plantillas de Django

Además, las vistas hacen uso de plantillas como hemos podido ver en la vista de ejemplo, debemos implementar estas plantillas en “proyectoElena/templates/”. Estas plantillas contienen código HTML en el que se pueden incrustar pequeños bloques de datos e instrucciones. Su finalidad es evitar tener que escribir código HTML directamente desde programas Python. Un ejemplo de una plantilla:

```
{% extends “plantilla.html” %}

{% block contenido %}

<table border=”1” cellpadding=”10%” width=”100%” height=”70%”>
  <tr>
    <td width=”20%” style=”padding:10px;” bgcolor=”#5F9EA0”
              align=”left” valign=”top”>
      <ul>
        <li><a href=”/perfil/”>Perfil</a></li>
        <li><a href=”/asignaturas/”>Asignaturas impartidas</a></li>
        <li><a href=”/misexamenes/”>Mis exámenes</a></li>
        <li><a href=”/social/”>Social</a></li>
      </ul>
    </td>
    <td colspan=”2”></td>
  </tr>
</table>
</center>

{% endblock %}
```

Como se puede ver, existen unas marcas entre “{%” y “%}” que no corresponden con código HTML. Veamos qué significan.

Django proporciona una forma de reducir código en sus plantillas, esta forma permite utilizar la misma parte de código en varias plantillas. Por ejemplo, la cabecera y parte inferior de las páginas suelen ser idénticas en todas las páginas.

En nuestro caso, la plantilla “madre” (*plantilla.html*) o código reutilizado es el siguiente:

```
 <center>
 <table border="0" width="100%" height="15%">
  <tr valign="top">
   <td align="center" rowspan="2"><a href="/"><img src="{{
            MEDIA_URL }}Banner.jpg" align="center" width=800
            height="100"></a></td>
  </tr>
  <tr valign="bottom">
   <td align="right"><p><font color="Teal"><b><i>Bienvenid@, {{
            user.first_name }}</i></b></font></p><p><font
            size=2><i>Última conexión: {{ user.last_login }}
            </i></font><br><a href="/logout/">Cerrar
            sesión</a></p></td>
  </tr>
 </table>

{% block contenido %}{% endblock %}

<!-- Seccion enlaces -->
 <center>
  <table border="0" cellpadding="10%" height="5%">
   <tr>
    <td><a href="/quees/">¿Qué es EasyTests?</a></td>
    <td><a href="/faq/">FAQ</a></td>
    <td><a href="/contacto/">Contacto</a></td>
   </tr>
  </table>
 </center>
```

Aquí también podemos ver unas marcas que no corresponden con HTML. Estas marcas son “{% block contenido %}” y “{% endblock %}”. Indican que serán sustituidas por el bloque llamado *contenido*. Si nos fijamos en la plantilla de ejemplo, podemos ver estas mismas marcas en la segunda y última fila de código. Entre ellas hay código HTML. Éste código será el que se sitúe entre las marcas en *plantilla.html*. Además, en la plantilla ejemplo hay una marca más, “{% extends “plantilla.html” %}”, que indica que la plantilla actual está “extendiendo” a

plantilla.html. Este es un comportamiento similar a la herencia de objetos en Java.

Además, podemos ver otras marcas en *plantilla.html* que no corresponden con código HTML. Se trata de las marcas "{{MEDIA_URL}}", "{{user.first_name}}" y {{user.last_login}}. Todas indican que se debe imprimir el valor contenido entre "{{" y "}}". En el primer caso, MEDIA_URL se trata de un parámetro de configuración de Django que indica la ruta donde se alojan los contenidos multimedia de la aplicación. En los otros dos casos, *user* es el objeto usuario de autenticación de Django, que posee diversos atributos. Explicaremos este objeto en el siguiente apartado. Se puede utilizar cualquier objeto y variable que se hayan pasado a la hora de llamar a la plantilla desde la vista, del mismo modo, albergándolo entre "{{" y "}}" para imprimir su contenido.

5.9. Vista de autenticación

Ahora que ya tenemos la estructura básica creada, vamos a centrarnos en ciertas vistas. El primer punto que debemos afrontar, es la implementación de la vista de autenticación. Para ello, tenemos dos formas de realizarla que explicamos a continuación.

La primera forma es mediante cookies y el uso de la base de datos. Veamos cómo sería la vista de autenticación y cierre de sesión con esta implementación:

```
def login(request):
        if request.method != 'POST':
                raise Http404('Solo POSTs estan permitidos')
        if 'usuario' in request.POST and 'passw' in request.POST:
                usuario = request.POST['usuario']
                passw = request.POST['passw']
                try:
                        u = Usuario.objects.get(nombre=usuario)
                        p = u.password
                        if passw != "":
                                if passw == p:
                                        ##Usuario valido
                                        error=False
                                        unlog=False
                                        request.session['member_id'] = usuario
                                        request.session['SESSION_EXPIRE_ AT
                                                _BROWSER_CLOSE'] = True
                                        ##Sesion establecida
                                        return render_to_response
                                                ('principal.html', locals(),
                                                context_instance=
                                                RequestContext(request))
```

```
                        else:
                                unlog=False
                                error=True
                                return render_to_response('login.html',
                                        locals(), context_instance=
                                        RequestContext(request))
                else:
                        unlog=False
                        error=True
                        return render_to_response('login.html', locals(),
                                context_instance=RequestContext
                                (request))
        except:
                unlog=False
                error=True
                return render_to_response('login.html', locals(),
                        context_instance=RequestContext(request))
    else:
        raise Http404 ("Hay un error con el login")
```

```
def logout(request):

        try:
                del request.session['member_id']
        except KeyError:
                pass
        unlog=True
        error=False
        return render_to_response('login.html', locals(),
                        context_instance=RequestContext(request))
```

De igual modo, cada vez que un usuario solicite una vista, para realizar la comprobación de autenticación la vista debe quedar del siguiente modo:

```
def perfil(request, user):
        usuario=user
        a = Asignatura.objects.get(profesores=usuario)
        asignatura = a.nombre
        titulacion = a.titulacion

        return render_to_response('perfil.html', locals(),
                context_instance=RequestContext(request))
```

La segunda forma consiste en utilizar la herramienta proporcionada por Django para la gestión de usuarios. Las vistas autenticación y cierre de sesión quedarían de la siguiente forma:

```
def login(request):
    if request.method != 'POST':
        raise Http404('Solo POSTs estan permitidos')
    if 'usuario' in request.POST and 'passw' in request.POST:
        usuario = request.POST['usuario']
        passw = request.POST['passw']
        try:
            user = auth.authenticate(username=usuario,
                password=passw)

            if user is not None and user.is_active:
                unlog=False
                auth.login(request,user)
                return render_to_response('principal.html',
                    locals(), context_instance=
                    RequestContext(request))
            else:
                unlog=False
                error=True
                return render_to_response('login.html', locals(),
                    context_instance=
                    RequestContext(request))

        except:
            unlog=False
            error=True
            return render_to_response('login.html', locals(),
                context_instance=RequestContext(request))
    else:
        raise Http404 ("Hay un error con el login")
```

```
def logout(request):

    try:
        auth.logout(request)
        unlog=True
        error=False
        return render_to_response('login.html', locals(),
            context_instance=RequestContext(request))
    except:
        unlog=False
        error=True
        return render_to_response('login.html', locals(),
            context_instance=RequestContext(request))
```

De igual modo, para realizar la comprobación de autenticación cada vez que se solicite una vista, la vista quedaría de esta manera:

```
def perfil(request):
        if not request.user.is_authenticated():
                return render_to_response('login.html', locals(),
                                context_instance=RequestContext(request))
        usuario=request.user.username
        a = Asignatura.objects.get(profesores=usuario)
        asignatura = a.nombre
        titulacion = a.titulacion

        return render_to_response('perfil.html', locals(),
                context_instance=RequestContext(request))
```

Si nos fijamos en las partes resaltadas en negrita, veremos que la autenticación de la segunda forma es más sencilla, ya que, además de contener menos código, no necesitamos estar trasladando el objeto *User* de un método a otro.

Se ha elegido la última forma porque para nuestra aplicación Web es muy importante la seguridad en el acceso al contenido y es más fácil manejarla de esta forma. Además, en la segunda forma disponemos de un objeto *User*, que contiene varios métodos y atributos. Entre los atributos encontramos la contraseña, que Django almacena de forma codificada. Por ello el campo de contraseña que manteníamos en la base de datos en la primera implementación, pasó a ser innecesario y se eliminó.

5.10. Creación de ficheros XML

En una de las vistas, "*enviarexamen*", se requiere crear el fichero XML con la información del examen que utilizará Eyegrade. Este fichero contiene la información general del examen y las preguntas. Para poder manejar ficheros XML utilizamos el paquete xml.dom.minidom. Y creamos otra función llamada *crearxml*. Un ejemplo de esta función:

```
from xml.dom.minidom import Document

def crearxml(subject, degree, title, date, duration, questions, usuario, exa_id):

        doc = Document()
        exam = doc.createElement('exam')
        exam.setAttribute('xmlns','http://www.it.uc3m.es/jaf/eyegrade/ns/')
        exam.setAttribute('xmlns:eye','http://www.it.uc3m.es/jaf/eyegrade/ns/')
        doc.appendChild(exam)

        fecha=doc.createElement('date')
        text=doc.createTextNode(date.strftime("%d/%m/%Y"))
        fecha.appendChild(text)
        exam.appendChild(fecha)
```

```
        #Y así sucesivamente con todos los nodos y atributos…

        #Transformamos los datos a xml tabulado con codificacion UTF-8
        data=doc.toprettyxml(encoding="UTF-8")

        dest="./examenes/"+usuario+"/xml/"+exa_id+".xml"

        ##Escribimos el resultado en el fichero xml
        f=open(dest,"w")
        f.write(data)
        f.close()

        return dest
```

5.11. Interacción con Eyegrade

Con el fichero XML y la plantilla LaTeX, debemos llamar a Eyegrade para que genere los diferentes modelos de examen. Eyegrade posee una función llamada *create_exam* que permite generar los exámenes con los modelos especificados. Como argumentos, introducimos: la ruta del XML, la ruta de destino, la ruta de la plantilla Eyegrade que vamos a utilizar para generar el examen, y los modelos que deseamos generar. Para utilizar esta función, se ha utilizado el comando *subprocess* que proporcionan las librerías de Python, ya que éste nos permite esperar a que el proceso termine y ver el estado de finalización mediante ficheros. Además, si se produce algún error por los argumentos de entrada, éste puede ser tratado desde nuestra aplicación sin necesidad de modificar las librerías de Eyegrade. El código utilizado sería el siguiente:

```
try:
        outfd=open('eyegrade_out','w+')
        errfd=open('eyegrade_err','w+')

        subprocess.call(["python","-m","eyegrade.create_exam", "-e",
                xml,  "-m", modelos, plantilla, "-o", destino],
                stdout=outfd, stderr=errfd)
        outfd.close()
        errfd.close()
except:
        ##Tratamos el error como sea necesario
```

Para utilizar el comando *subprocess* hay que importar a nuestro código la librería:

```
import subprocess
```

5.12. Generación de archivos PDF

También se necesita generar un fichero PDF por cada modelo de examen una vez que Eyegrade ha generado los ficheros LaTeX del examen. Para ello usamos el comando pdflatex. Hay que descargarse e instalar la librería correspondiente. La llamamos de este modo:

```
for m in modelos:
        ##Creamos pdf
        try:
                outfd=open('pdf_out','w+')
                errfd=open('pdf_err','w+')
                origen=directorio+"/latex/"+str(exa.id)+"-"+m+".tex"
                destino=directorio+"/pdf"
                subprocess.call(["pdflatex", "--output-directory",
                                destino, origen], stdout=outfd, stderr=errfd)
                outfd.close()
                errfd.close()
                ##Borramos archivos innecesarios
                os.remove(destino+"/"+str(exa.id)+"-"+m+".aux")
                os.remove(destino+"/"+str(exa.id)+"-"+m+".log")
                ##Pdf de las soluciones
                outfd2=open('pdf_out_2','w+')
                errfd2=open('pdf_err_2','w+')
                origen=directorio+"/latex/"+str(exa.id)+"-"+m+
                        "-solutions.tex"
                destino=directorio+"/pdf"
                subprocess.call(["pdflatex", "--output-directory", destino,
                        origen], stdout=outfd2, stderr=errfd2)
                outfd2.close()
                errfd2.close()
                ##Borramos archivos innecesarios
                os.remove(destino+"/"+str(exa.id)+"-"+m+"-solutions.aux")
                os.remove(destino+"/"+str(exa.id)+"-"+m+"-solutions.log")
        except:
                print "ERROR PDF"
```

5.13. Seguridad en la aplicación

Llegados a este punto, nos planteamos la seguridad de estos archivos generados, de las plantillas y de las imágenes que se almacenan sobre preguntas. Dichos archivos no deben estar accesibles si no es el autor el que los solicita. Dado que estos archivos no los hemos asociado a la base de datos, puesto que esto la sobrecargaría, nos encontramos ante un problema de seguridad: Django puede garantizar el acceso a cualquier dato de la base de datos a ciertos usuarios, pero no puede garantizar el acceso de alguien no autorizado a un archivo que se

encuentra bajo su directorio de archivos multimedia si se conoce la URL.

Para solucionar este problema, se ha decidido crear una estructura de carpetas fuera del directorio general. La estructura es la siguiente:

```
proyectoElena/
        examenes/
                pepeflores/
                        imagenes/
                        latex/
                        pdf/
                        plantillas/
                        xml/
                nereajimenez/
                        imagenes/
                        latex/
                        pdf/
                        plantillas/
                        xml/
```

Cada usuario tiene su propia carpeta, y dentro de esa carpeta tiene una serie de subcarpetas donde se almacenan las imágenes de las preguntas, los archivos LaTeX, los archivos PDF, las plantillas subidas y los XML de los exámenes.

De modo que en nuestra vista, solo permitimos la descarga de los archivos al usuario que los creó.

6. PRUEBAS

Ahora que ya tenemos la implementación de nuestra aplicación Web, vamos a realizar una serie de pruebas para eliminar el máximo número de fallos posibles. Además, veremos cómo poner en marcha EasyTests.

6.1. Pruebas realizadas

Lo primero que debemos hacer es lanzar el servidor. Para ello ejecutamos en un terminal desde el directorio “proyectoElena”:

```
python manage.py runserver
```

Este comando lanzará el servidor en la IP 127.0.0.1 y en el puerto 8000. Para visitar la página Web, habría que acceder con un navegador a la dirección: http://127.0.0.1:8000/.

Django permite cambiar la dirección IP y el puerto en el que es lanzado el servidor. Tan sólo tenemos que modificar el comando:

```
python manage.py runserver 8080
```

```
python manage.py runserver 0.0.0.0:8000
```

Para comenzar con las pruebas, utilizaremos el primer comando, lanzando el servidor de forma local. Más tarde, se puede escoger una IP estática de la universidad para su lanzamiento.

Las primeras pruebas que se realizaron eran sobre el funcionamiento general de Django. Para ello se utilizaron ejemplos de páginas Web y se iban modificando según avanzaba la lectura de algunas guías de Django.

Una vez que se tenía un conocimiento general práctico sobre el funcionamiento de Django, se comenzaron las pruebas de los distintos modos de realizar la vista de autenticación, viendo ventajas y desventajas y decantándonos por uno de ellos.

El siguiente paso que se dio fue establecer la disposición gráfica de los componentes de la interfaz de dicha vista. Se fue probando cuál era la colocación más funcional e intuitiva, el tamaño del logo, los colores, la tipografía, etc. Además se realizó una búsqueda por Internet de un logo con licencia abierta que pudiera ser modificado y adaptado a nuestras necesidades. Tras la infructuosa búsqueda, ya que las imágenes y vectores disponibles en Internet no podían ser usados como parte de un logo, se decidió hacer una fotografía propia que reflejara el espíritu de EasyTests.

A medida que se iban implementando el resto de vistas, se iba viendo su diseño, colocación de los elementos, comprobando los enlaces, probando los formularios, pensando posibilidades de mejora (tanto visuales como funcionales), etc.

Una de las mejores formas de realizar pruebas para detectar fallos es introducir datos incoherentes en formularios o dejar campos vacíos que deberían ser obligatorios. En las páginas dónde se envía algún tipo de información es donde más vulnerable a fallos es nuestra aplicación Web.

Por ejemplo, en nuestro formulario para crear una pregunta, el texto es un campo obligatorio. Probamos a enviar el formulario sin rellenar este campo. Tras escribir en el código de la vista las comprobaciones de que ese campo posea información antes de continuar, se decide qué hacer cuando el campo esté vacío. En nuestro caso, se decide retornar la misma plantilla HTML pero con un error.

El código de ejemplo sería:

```
if request.POST['texto']!="":
        #Acciones a realizar
else:
        request.session["error_texto"]=True
        return crearpregunta(request)
```

Por otra parte, existen campos de nuestros formularios que deben mantener un formato específico. Por ejemplo la fecha del examen. En este caso se decidió que para que la compatibilidad entre formatos fuera más sencilla, se creó un menú desplegable para el día, otro para el mes y otro para el año. De este modo aseguramos que los datos introducidos son correctos. También disponemos de un campo en el que se indica la duración en minutos de un examen. En este caso, utilizando un código similar al arriba indicado, se comprueba que ese campo contenga únicamente información numérica.

Además, tras realizar varias pruebas con la parte del formulario donde se marca cuál es la respuesta correcta para una pregunta de las escritas, se decidió poner por defecto marcada la opción 1, ya que así se evita que el usuario olvide marcarla y agiliza la tarea para aquellos profesores que tienen organizadas sus preguntas de modo que la respuesta correcta es la primera.

Otras pruebas realizadas son las de consulta de preguntas, exámenes, plantillas, imágenes y perfiles. Para ello se introdujeron datos en la base de datos para ir probando todas las funciones. Entre estas funciones, destaco la implementada en los perfiles con las fotografías de usuario. Cuando un usuario no tiene establecida una fotografía, bien porque no haya sido subida o bien porque ha sido eliminada, se establece una foto

por defecto según el sexo de la persona. Para ello se crearon varias cuentas con distintos usuarios de distintos sexos.
Otras de las pruebas realizadas son la modificación de la contraseña de usuario, la modificación de los datos básicos de perfil, la modificación de las asignaturas impartidas, el filtrado de preguntas y exámenes por asignatura, la subida y descarga de ficheros, la modificación y eliminación de exámenes, etc.

6.2. Poniendo en marcha EasyTests

Una vez que la aplicación ha sido desarrollada y testeada, tenemos que realizar unos pasos adicionales para ponerla a disposición del público general [34].

Lo primero, es desactivar el modo de depuración. Para ello abrimos el archivo "/proyectoElena/proyectoElena/settings.py" y establecemos:

```
DEBUG = False
```

Hay que asegurarse que TEMPLATE_DEBUG tome los valores de DEBUG o *False*.

El siguiente paso es crear nuestras propias plantillas de error HTTP 404 y 500, ya que una vez que hemos desactivado el modo de depuración, cuando Django reciba una solicitud de una dirección que no tenemos asociada, mostrará la página de error HTTP 404 y cuando ocurra un error a nivel del código Python, mostrará la página de error HTTP 500. Estas plantillas se guardan en el directorio proyectoElena/templates/ con los nombres *404.html* y *500.html* respectivamente.

Por último, ejecutamos el servidor en una IP fija de la universidad en el puerto 80 (puerto predeterminado de HTTP) y contratamos un dominio, por ejemplo www.easytests.com[2], que asociaremos con la IP que hemos establecido.

Por ejemplo[3]:

```
python manage.py runserver 163.117.136.61:80
```

Una vez ejecutado este comando, podremos acceder a nuestra aplicación Web accediendo al dominio contratado.

[2] Esta dirección Web es únicamente un ejemplo. En ningún caso está asociada a la aplicación Web ni es vinculante.
[3] Esta dirección IP es únicamente un ejemplo. En ningún caso está asociada a la aplicación Web ni es vinculante.

7. CONCLUSIONES Y TRABAJOS FUTUROS

7.1. CONCLUSIONES

En este proyecto se ha desarrollado una aplicación Web para facilitar la interacción de la aplicación ya existente Eyegrade desde cualquier dispositivo.

Tras realizar un análisis tecnológico sobre las distintas tecnologías de programación para la Web, se decidió que dicha Web fuera realizada en Python y usando el framework Django.

La aplicación Web ha conseguido los siguientes objetivos:

- Acceso limitado a usuarios registrados.
- Se limitan los registros para el administrador, impidiendo que cualquier persona pueda acceder.
- Permite la creación de preguntas individuales con sus respectivas respuestas. Estas preguntas quedan almacenadas en una base de datos y solo son accesibles por la persona que las creó.
- Permite la creación de exámenes, eligiendo las preguntas ya creadas que se desean utilizar y los modelos de examen que se desean crear, así como los datos básicos de asignatura, fecha, título, etc. Al igual que las preguntas, solo pueden ser accedidos por su autor.
- Una vez creado el examen, se crean los correspondientes PDF. Éstos, junto a los ficheros LaTeX creados por Eyegrade (los cuales son necesarios para otras opciones del programa) pueden ser descargados desde la Web.
- Además, la aplicación Web permite la modificación de preguntas y exámenes. En el caso de la modificación de preguntas, si ésta está siendo usada por un examen, la modificación provocará una nueva pregunta, manteniendo la versión anterior de la pregunta para evitar inconsistencias.
- En cualquier momento, pueden ser borrados los exámenes creados.
- Se dispone de un perfil de usuario. Este perfil contiene datos como el nombre, los apellidos, una fotografía, las asignaturas impartidas y un correo de contacto. Dicho perfil es público dentro de la comunidad de usuarios de la aplicación Web. En cualquier momento, el usuario puede modificar sus datos introduciendo su contraseña como confirmación. Estos perfiles se han creado para

que la Web pueda ser ampliada en un futuro de forma que se puedan compartir los exámenes entre profesores.

7.2. TRABAJOS FUTUROS

Este proyecto deja abierta la posibilidad (como ya comentaba al final del punto 7.1) de implementar la compartición de exámenes. Se podría dejar elegir al usuario qué exámenes y preguntas desea compartir, y si lo desea compartir con los profesores que imparten esa asignatura o con toda la red de usuarios. Además se podría crear una lista de "bloqueados", de tal modo que si cierto usuario pertenece al grupo de usuarios que tienen permiso para ver el examen de otro usuario, pero el autor le ha denegado el permiso explícitamente, éste no pueda acceder a dicho examen.

Otra mejora que podría implementarse es la internacionalización de la aplicación Web. De este modo, la Web podría ser visitada por usuarios que no entendieran el idioma español.

Por otra parte, Eyegrade posee de más funciones que no se han implementado en esta aplicación Web. Una de ellas es la corrección automática de exámenes mediante una cámara Web. También permite introducir una lista de usuarios en la que aparece el identificativo (NIA) y su nombre. Además, Eyegrade realiza una serie de estadísticas con los resultados obtenidos de las correcciones. Todo esto podría implementarse para poder ser realizado a través de la aplicación Web.

Con la experiencia adquirida podrían realizarse otras aplicaciones Web de mayor dificultad, experimentar con las cargas de la base de datos y ver cuáles son los límites del lenguaje y framework elegidos. Siempre pueden compartirse los conocimientos adquiridos participando en el foro de programadores y compartiendo librerías propias.

8. PRESUPUESTO

Se presenta a continuación el presupuesto del proyecto completo. Para realizar el cálculo se deben tener en cuenta la duración de las fases y tareas, los costes de personal y el coste de material.

8.1. Coste de personal

Por distintos motivos, los tiempos que se han necesitado para completar las diferentes fases del proyecto se han visto dependientes de la variabilidad de horas dedicadas por día y de ciertos periodos de tiempo en los que la dedicación tuvo que ser menor. Sin embargo en un entorno de trabajo real se parte del hecho de que existe un número de horas de dedicación prefijadas y constantes. Puesto que el presente presupuesto pretende dar un coste en un entorno como el comentado, se han realizado los siguientes promedios para adaptar los tiempos que se han empleado a una situación profesional real. El promedio de horas dedicadas al día realmente ha sido de aproximadamente 4 horas. Dado que en un entorno laboral real, el promedio de horas diarias es de 8, el sueldo estimado de 2694,39 € mensuales pasa a ser:

2694,39€ / 8 horas diarias * 4 horas diarias = 1347,20€ mensuales.

Por tanto:

Nombre	Categoría	Dedicación	Coste	Coste
Elena Martín García	Ingeniero Técnico	12 meses	1347,20€ / mes	16.166,34€
Jesús Arias Fisteus	Director de proyecto	30 horas	40,00€ /hora	1200,00€
			Total	**17.366,34€**

8.2. Coste de equipos y software

- **Coste de equipos**

Descripción	Coste	% Uso	Dedicación (meses)	Periodo de depreciación (meses)	Coste
Ordenador de sobremesa	900,00€	70%	12	60	120,00€
Router ADSL	50,00€	70%	12	60	7,00€
				Total	**127,00€**

- **Coste de software**

Nombre	Coste	Dedicación (meses)	Periodo de depreciación (meses)	Coste
S.O. Ubuntu	---	---	---	---
Microsoft Office Professional 2003	365,95€	4	30	48,79€
Adobe Acrobat XI Pro	207,01€	4	30	27,60€
Navegador Firefox	---	---	---	---
Navegador Chrome	---	---	---	---
Editor de texto	---	---	---	---
MySQL	---	---	---	---
Django	---	---	---	---
Dia (para diagramas)	---	---	---	---
Photoshop CS4	249,00€	1	30	8,30€
GanttProject (para diagrama de Gantt)	---	---	---	---
Java v7	---	---	---	---
Python	---	---	---	---
Eyegrade	---	---	---	---
			Total	**84,69€**

8.3. Costes indirectos

Los costes indirectos del proyecto suponen aproximadamente el 20% del subtotal de los costes de personal, equipos y software. Por tanto:

Descripción	Coste
Personal	17366,34€
Equipos	127,00€
Software	84,69€
Subtotal	**17578,03€**
Total costes indirectos (20%)	**3515,61€**

8.4. Resumen de costes

Concepto	Coste
Personal	17366,34€
Equipos	127,00€
Software	47,77€
Costes indirectos	3515,61€
Total (sin IVA)	**21.093,64€**

Concepto	Coste
Total (sin IVA)	21.093,64€
IVA (21%)	4.429,66€
TOTAL	**25.523,30€**

9. REFERENCIAS BIBLIOGRÁFICAS

Eyegrade:

[1] Jesús Arias Fisteus, "Eyegrade: Grading multiple choice questions with a webcam". http://eyegrade.org/

[2] Jesús Arias Fisteus, "Eyegrade User Manual". http://eyegrade.org/doc/user-manual/

[3] Jesús Arias Fisteus, Abelardo Pardo and Norberto Fernández García, "Grading Multiple Choice Exams with Low-Cost and Portable Computer-Vision Techniques". Journal of Science Education and Technology. http://dx.doi.org/10.1007/s10956-012-9414-8

PHP:

[4] The PHP Group, "PHP Hypertext Preprocessor". http://php.net/

[5] The PHP Group, "Manual de PHP". http://www.php.net/manual/es/

[6] Wikipedia, "PHP". http://es.wikipedia.org/wiki/PHP

[7] Alegsa, "Definición de Server-side": http://www.alegsa.com.ar/Dic/server-side.php

[8] GNU, "Varias licencias y comentarios acerca de las mismas". http://www.gnu.org/licenses/license-list.html#GPLIncompatibleLicenses (Accedido el 02/09/13)

[9] Rubén Gómez, "PHP vs. Java". http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=PHPVsJava

[10] Peter MacIntyre, "Programming PHP", ISBN: 1-4493-9277-6, 978-1-4493-9277-2, fecha: 01.02.2013, editorial: O'Reilly Media.

[11] Rasmus Lerdorf, "PHP Pocket Reference", ISBN: 1-56592-769-9, 978-1-56592-769-8, fecha: 01.01.2000, editorial: O'Reilly Media.

J2EE:

[12] Oracle Tecnology Network, "Java EE Documentation", http://www.oracle.com/technetwork/java/javaee/documentation/index.html

[13] Wikipedia, "Java EE". http://es.wikipedia.org/wiki/Java_EE

[14] Miguel Ángel Garrido Pino, “Evaluación Comparativa de aplicaciones Web entre J2EE y Microsoft.NET”. http://es.scribd.com/doc/63627033/13/Caracteristicas-de-J2EE

[15] Universidad de Alicante, “Productos para desarrollar aplicaciones Web: ASP, CFM, JSP y PHP”. http://rua.ua.es/dspace/bitstream/10045/14015/4/2-asp-cfm-jsp-php.pdf

[16] Paul J. Perrone, “J2EE developer's handbook”, ISBN: 0672323486, fecha: 2003, editorial: Indianapolis (Indiana) : Sams.

[17] Richard Monson-Haefel, “J2EE Web Services”, ISBN: 0-321-14618-2, 978-0-321-14618-2, fecha: 20.10.2003, editorial: Addison-Wesley Professional.

Python:

[18] Python Software Foundation, “Python Programming Language – Oficial Website”, http://www.python.org/

[19] Wikipedia, “Python”, http://es.wikipedia.org/wiki/Python

[20] Juanlu001, “Introducción a Python para científicos e ingenieros”. http://pybonacci.wordpress.com/2012/03/16/introduccion-a-python-para-cientificos-e-ingenieros/

[21] Celia Clemente Castillo, Lara Fajardo Ibáñez, Valentín Cruz Rodríguez y Amanda Garci Núñez (Universidad Carlos III de Madrid), “Python”. http://www.it.uc3m.es/spickin/docencia/comsoft/presentations/spanish/doc/Python.pdf

[22] Jonatan Pin García, “Python - Ventajas y Desventajas”. http://jonatanpin.blogspot.com.es/2011/04/python-ventajas-y-desventajas.html

[23] Web++, “Comparativa Python vs. PHP vs. Java”. http://webplusplus.blogspot.com.es/2011/11/comparativa-python-vs-php-vs-java.html

[24] Ricardo Wong, “Programming Python”. http://ricardowong.tumblr.com/post/2693948431/python

[25] James Payne, “Python: Using Python 2. 6 and Python 3. 1”, ISBN: 0-470-41463-4, 978-0-470-41463-7, fecha: 28.01.2010, editorial: Wrox Press Ltd.

[26] Mark Lutz, “Programming Python, 4th Edition”, ISBN-13: 978-0-596-15810-1, fecha: 31.12.2010, editorial: O'Reilly Media, Inc.

Django:

[27] Django Project Community, “Django”. https://www.djangoproject.com/

[28] Wikipedia, “Django Framework”. http://es.wikipedia.org/wiki/Django_%28framework%29

[29] Django Project Community, “Django Documentation”. https://docs.djangoproject.com/en/1.4/

[30] Adrian Holovaty y Jacob Kaplan-Moss, “The Django Book”. http://www.djangobook.com/ (versión 1.0 ya no disponible)

[31] Zenx IT, “Django en Español”. http://www.django.es/

[32] Adrian Holavaty, Jacob Kaplan-Moss, “The Definitive Guide to Django: Web Development Done Right (Experts Voice in Web Development”, ISBN-10: 143021936X, fecha: 08.07.2009, editorial: Apress.

[33] Jeff Forcier, Paul Bissex, Wesley Chun, “Python Web Development With Django”, ISBN-10: 0132356139, fecha: 03.11.2008, editorial : Addison-Wesley Professional.

[34] Adrian Holovaty, Jacob Kaplan-Moss, “Chapter 12: Deploying Django”, http://www.djangobook.com/en/2.0/chapter12.html

Ruby:

[35] Comunidad de Ruby, “Lenguaje de programación Ruby”, https://www.ruby-lang.org/es/

[36] Wikipedia, “Ruby on Rails”. http://es.wikipedia.org/wiki/Ruby_on_Rails

[37] Steve Johnson (traducido por Daniel Gómez Villegas), “Las desventajas de trabajar con Ruby”. http://www.ehowenespanol.com/desventajas-programar-ruby-info_205683/

[38] Mancuso Emiliano, “Ruby on Rails”. http://www.slideshare.net/NewBU/ruby-on-rails-una-breve-introduccin

[39] Burply, “Ruby on Rails. Ventajas sobre otros lenguajes de programación”. http://es.burply.com/los-entornos-de-aplicaciones-web/web-20/lenguaje-de-programaci%C3%B3n-ruby-200858.html (Accedido el 05/09/13)

[40] David Thomas, “Programming Ruby: the pragmatic programmers guide”, ISBN: 0974514055, fecha: 2006, editorial: The pragmatic bookself.

.NET:

[41] Microsoft, “.NET Downloads, Development Resources & Case Studies”. http://www.microsoft.com/net

[42] Wikipedia, “Microsoft .NET”. http://es.wikipedia.org/wiki/Microsoft_.NET

[43] Yahoo! Respuestas, “¿.NET es multiplataforma?”. http://espanol.answers.yahoo.com/question/index?qid=20061129192703AA8jM4d

[44] Todoexpertos, “Desarrollo multiplataforma”. http://www.todoexpertos.com/categorias/tecnologia-e-internet/desarrollo-de-sitios-web/asp/respuestas/541155/desarrollo-multiplataforma

[45] LWP, “ASP - Ventajas y desventajas del asp”. http://www.lawebdelprogramador.com/foros/ASP/381741-Ventajas_y_desventajas_del_asp.html

[46] Antonio Elena, “Ventajas y desventajas de asp.net mvc”. http://aelena.com/blog/2009/05/13/ventajas-y-desventajas-de-aspnet-mvc/

[47] Tuyub, “Microsoft .NET”. http://tuyub.wordpress.com/2007/07/18/microsoft-net/

[48] Ivannov, “Y cuando quiero aprender .NET, ¿por dónde comienzo?”. http://blogs.msdn.com/devcolombia/archive/2006/09/08/746465.aspx

[49] Keith Ballinger, “.Net Web Services: Architecture and Implementation”, ISBN: 0-321-11359-4, 978-0-321-11359-7, fecha: 12.02.2003, editorial: Addison-Wesley Professional