UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA TÉCNICA DE TELECOMUNICACIÓN:
TELEMÁTICA

Universidad
Carlos III de Madrid

PROYECTO FINAL DE CARRERA

EGSTATS: APLICACIÓN DE ESTADÍSTICAS DE EXÁMENES DE RESPUESTA MÚLTIPLE

AUTOR: RAÚL CABALLERO ORTEGA

TUTOR: JESÚS ARIAS FISTEUS

Octubre de 2015

Agradecimientos

En primer lugar quiero agradecer a mi tutor, Jesús Arias Fisteus, su paciencia, apoyo y directriz durante la realización de este proyecto, así como las facilidades que me ha ofrecido todo este tiempo.

Mi gratitud eterna a mi mujer, Inés, probablemente quien más ha sufrido mi aislamiento durante el desarrollo de este proyecto. Gracias por tus ánimos continuos y lo mucho que me has ayudado a poder centrarme en él, con paciencia casi infinita.

A mis amigos y compañeros Nuria, Germán, Nacho, Mamen, Rafa, José, Marta, Estrella, Santi, Aitor, Enrique, Ivan, José Luis y otros muchos, presentes y pasados que de un modo u otro han contribuido a que esté donde estoy.

Y entre todos ellos, mi especial agradecimiento y mención a Juan José, que después de todos estos años sigue aguantándome y dándome sabios consejos, aunque no siempre los siga. Más que un amigo... un hermano.

Tampoco puedo olvidarme de todos los profesores que han contribuido a que por fin llegue este día. Imposible citarles a todos, pero desde aquí, mi gratitud hacia todos ellos.

A todos los citados, y a aquellos que se han quedado en el tintero... gracias.

Resumen

El objetivo de este proyecto es crear un complemento para el sistema Eyegrade[1][7], de Jesús Arias Fisteus.

Eyegrade es un sistema que permite corregir automáticamente exámenes multirrespuesta, usando para ello una sencilla cámara *web*. El resultado son unos ficheros de correcciones en formato CSV (*Comma Separated Values*) fácilmente exportables e interpretables a través de otras herramientas.

Precisamente esa es la idea de este proyecto. Ofrecer una plataforma a la que subir esos resultados, analizarlos, y facilitar datos estadísticos a través de los cuales extraer información que permita la detección de los puntos más problemáticos del proceso de aprendizaje de estas asignaturas.

Con Eyegrade se pueden extraer los resultados con facilidad y rapidez. Lo que se persigue ahora es facilitar la interpretación de estos resultados.

Mediante las gráficas y datos presentados por Egstats, se podrán identificar fácilmente las preguntas que con mayor frecuencia presentan dificultades para los alumnos, los errores que se repiten más a menudo, o las partes del temario que más les cuesta asimilar.

Además el servidor permitirá ofrecer a los alumnos el resultado de su examen, señalando tanto los errores cometidos como las respuestas correctas. Al ver las equivocaciones anteriores, será un apoyo a la hora de repasar la materia.

[1] `http://www.eyegrade.org`

Índice general

Lista de Figuras

Lista de Tablas

Capítulo 1

INTRODUCCIÓN

1.1. Motivación del proyecto

Uno de los métodos de evaluación más empleados actualmente es el examen de respuesta múltiple. Se trata de un sistema en el que el alumno debe elegir la respuesta correcta de entre las que se le plantean, debiendo en ocasiones realizar algún cálculo paralelo para poder extraer la concusión. El resultado de esta labor es una plantilla de respuestas con marcas señalando las repuestas seleccionadas (y en ocasiones marcando con una señal distinta alguna equivocación o cambio de opinión).

La corrección de este tipo de exámenes es laboriosa y no exenta de probabilidad de error, por lo que de un tiempo a esta parte se está extendiendo el uso de dispositivos y programas para hacer esta labor de forma automatizada, reduciendo el tiempo necesario, así como el riesgo de error humano.

Esta tarea normalmente requiere de *hardware* específico, en forma de escáneres diseñados con esta tarea en mente, que unidos al *software* adecuado son capaces de reconocer las marcas hechas por los estudiantes. Esto, obviamente, tiene un coste económico en concepto de compra del escáner, y licenciamiento del programa asociado.

Ante estas limitaciones aparece Eyegrade, un proyecto de *software* libre emprendido por Jesús Arias Fisteus que plantea una alternativa: realizar la corrección a través de una cámara *web* genérica con un programa desarrollado en Python[1][15].

Eyegrade permite la corrección automatizada, y la generación de ficheros de resultados. De

[1] `https://www.python.org/`

esta forma se puede evaluar a los alumnos de una forma rápida y reduciendo los costes a la compra de una simple cámara *web*.

Una vez obtenidas las notas de los alumnos, cabe preguntarse si no se puede hacer algo más con esa información. El siguiente paso que se puede plantear es si se podrían extraer datos o conclusiones adicionales. De los datos de evaluación se podrían extrapolar informaciones como la localización de las preguntas o temáticas que más dificultades o dudas plantean a los alumnos, qué preguntas se suelen dejar más sin responder, etc.

La forma de llegar a estas conclusiones es mediante estadísticas que usen como fuente de datos los exámenes, y presenten al profesor la información de una forma visualmente atractiva y fácil de analizar. Este es el paso que se pretende dar con Egstats: tomar los datos extraídos con Eyegrade y presentarlos en forma de gráficos estadísticamente representativos. Además se intentará facilitar la interacción con esta información a través de las propias gráficas y enlaces, conduciendo a información relacionada o relevante.

El objetivo es que el profesor tenga, a pocos "*clicks*" de distancia, la mayor cantidad de información posible acerca de los resultados de sus exámenes.

1.2. Objetivos

Los objetivos que se tratarán de alcanzar serán:

- Desarrollar una aplicación *web* que permita incorporar datos de exámenes evaluados mediante Eyegrade y generar estadísticas relevantes sobre los mismos.
- Facilitar el trabajo del profesor con la herramienta, mostrando la información de forma gráfica e interactiva, haciendo que los datos sean fácilmente localizables e interpretables.
- Permitir que los alumnos revisen sus exámenes y los resultados obtenidos.
- Facilitar el desarrollo de trabajos futuros que complementen la funcionalidad de esta herramienta.

1.3. Plan de trabajo

A fin de alcanzar los objetivos propuestos, se diseña un plan de trabajo, concretándolo en una lista de tareas que se acometerán de forma secuencial o en paralelo, en función de cada tarea

en concreto:

- Estudio del problema:
  - Estudio de Eyegrade
    - Conceptos básicos del programa
    - Ficheros empleados por Eyegrade (formato, datos incluidos, etc.)
      - Fichero de examen
      - Fichero de configuración del examen
      - Fichero de respuestas de los alumnos
  - Análisis de alternativas para el desarrollo de la aplicación
    - Aplicación *web* frente a aplicación nativa
    - Lenguaje de programación idóneo
    - Sistema de almacenamiento de datos idóneo
    - Punto donde se generarán las gráficas (servidor o cliente)
    - Plataforma para la generación de las gráficas
  - Selección de las herramientas más adecuadas y familiarización con su uso
- Diseño de la aplicación
  - Diseño de la estructura de la aplicación
    - Modelo de datos
    - Componentes *software* que la formarán
      - Plataforma
      - Plataforma de desarrollo para la aplicación
      - Plataforma de desarrollo para los gráficos
  - Diseño del uso de la aplicación
    - Tipos de usuario que existirán
    - Tipos de datos a los que tendrá acceso cada tipo de usuario
    - Forma de acceso a los datos, interacción con la aplicación
- Implementación de la aplicación

- Validación y pruebas
  - Toma de métricas de rendimiento y optimización
  - Diseño de pruebas
  - Ejecución de pruebas y validación de la aplicación
- Redacción de la memoria

Puede verse gráficamente en un diagrama de Gantt:

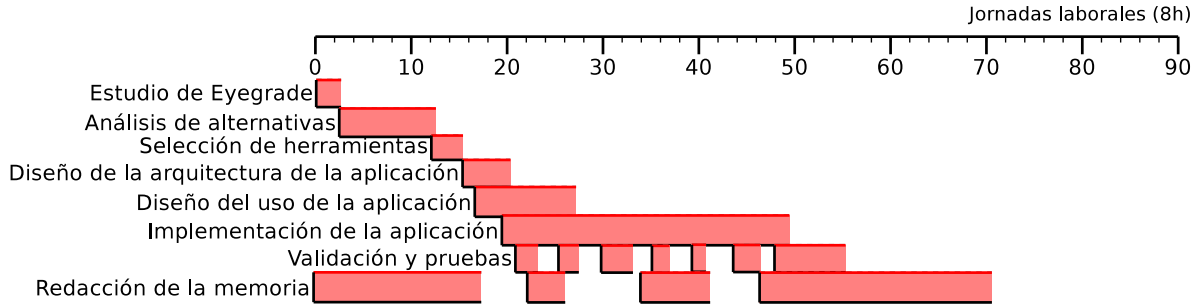

Figura 1.1: Arquitectura software de las gráficas

1.4. Introducción al resto de la memoria

En los siguientes apartados se van a esbozar en qué consistirán cada uno de los siguiente capítulos de esta memoria.

Estado del arte

Aquí se tratará de ponerse en situación con respecto a las diferentes alternativas que existen actualmente para dar respuesta a las necesidades del proyecto, a nivel de servidores, plataforma de desarrollo y todas las herramientas disponibles. Igualmente se expondrá el por qué se elegirán unas u otras.

Requisitos

En este apartado se enumerarán las necesidades a cubrir, ineludiblemente, durante el desarrollo de la aplicación. Se trata de condiciones indispensables para que ésta cumpla su cometido.

Arquitectura del sistema

En esta sección se describirán las diferentes piezas *software* a utilizar durante la implementación del proyecto, así como las interacciones entre las mismas. Se hará mención también a terceras partes que servirán de apoyo para ciertas tareas.

Diseño

Aquí se entrará más en detalle sobre cada uno de los módulos que compondrán el proyecto, dando una idea algo más detallada de su funcionamiento y destacando las diferencias con otras alternativas desechadas.

Implementación

Se describirán los aspectos más importantes y/o interesantes del desarrollo de la aplicación, haciendo énfasis en las dificultades encontradas, así como las soluciones adoptadas.

Validación y pruebas

En este apartado se describirán las pruebas realizadas sobre la aplicación a fin de verificar que cumple con los requisitos establecidos al inicio del proyecto, así como su correcto funcionamiento.

Conclusiones y trabajos futuros

Esta sección se dividirá en dos subapartados. El primero se centrará en las conclusiones alcanzadas al finalizar el proyecto, mientras que en el segundo se plantearán diferentes líneas de trabajo que podrían ampliar o complementar lo aquí realizado.

Capítulo 2

ESTADO DEL ARTE

En este apartado se describirá la situación actual, tanto en lo relativo al objetivo del proyecto, como respecto a las diferentes alternativas disponibles para acometer su desarrollo.

2.1. Eyegrade

Eyegrade es un desarrollo de Jesús Arias Fisteus, con el que se ofrece un sistema de corrección de exámenes de respuesta múltiple a través de una sencilla cámara *web*.

A partir de la hoja de respuestas del alumno, Eyegrade es capaz de emplear mecanismos de visión por computador para reconocer el identificador del alumno, así como las respuestas marcadas, extrayendo esta información y volcándola a un fichero de resultados.

Para hacerlo posible se basa en tres tipos de fichero:

Fichero de examen Es un fichero XML (*eXtensible Markup Language*) que contiene toda la información relativa a:

- Título del examen (convocatoria, parcial, etc).
- Asignatura a la que pertenece.
- Titulación a la que pertenece.
- Fecha de realización.
- Duración.
- Preguntas (incluyendo autor y etiquetas aplicables).
- Opciones de respuesta, marcando la correcta.

Fichero de configuración de examen Es un fichero con extensión propia (*.eye*) que contiene parámetros de configuración del propio examen, y que permitirán su tratamiento dentro de la aplicación:

- Dimensiones (en la hoja de respuestas, número de tablas, y filas y columnas de cada tabla).
- Número de dígitos del identificador de alumno.
- Ponderación de cada tipo de pregunta (correcta, incorrecta, en blanco)
- Soluciones para cada modelo de examen.
- Permutaciones a realizar desde el examen original a cada modelo de examen

Cada modelo de examen representa una modificación del examen original en el que tanto las preguntas, como las opciones de respuesta dentro de cada pregunta sufren una permutación, de forma que se dificulta la copia entre compañeros.

Fichero de respuestas Se trata de un fichero CSV que contiene:

- Número correlativo que hace de índice del fichero.
- Identificador del alumno.
- Modelo de examen realizado por el alumno.
- Número de respuestas acertadas.
- Número de respuestas fallidas.
- Nota numérica obtenida
- Respuestas marcadas por el alumno.

Conociendo la información de corrección del examen, el programa detecta las respuestas del alumno, y las corrige de forma automática, como puede verse en la imagen:

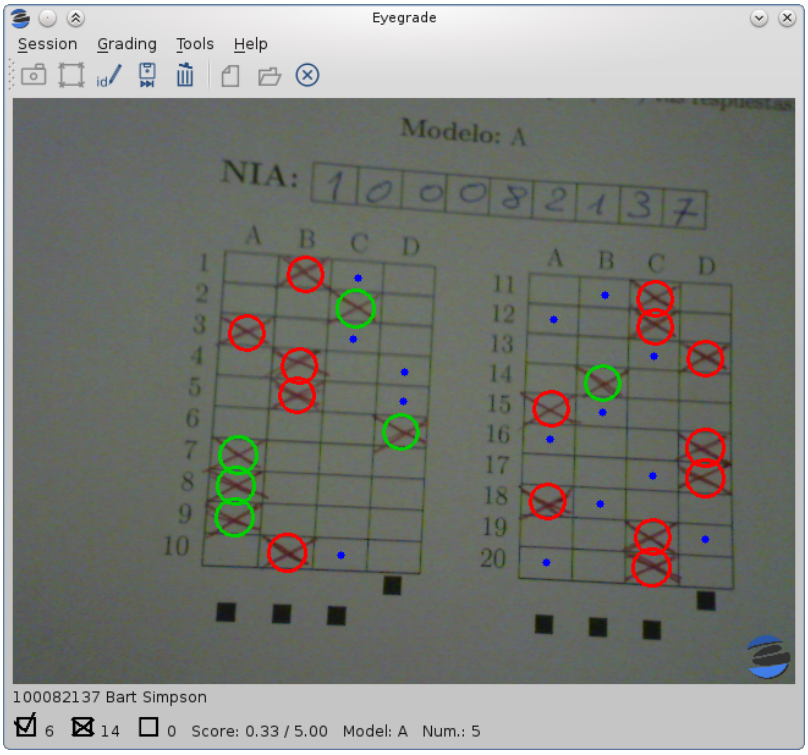

Figura 2.1: Eyegrade en funcionamiento

Su relación con este proyecto consiste en que se intentará complementar el uso de Eyegrade, ofreciéndole una plataforma a la que subir esos resultados, agruparlos, analizarlos y ofrecer datos y estadísticas que faciliten la extracción de conclusiones.

Existen en el mercado alternativas a Eyegrade, como PKT JRI[1] o QCM Direct[2], pero la mayoría de ellas se trata de *sofwtare* no libre (lo que implica un gasto en concepto de licencias) y requieren de *hardware* específico.

La versión más reciente de Eyegrade es la 0.6, y está íntegramente desarrollado en Python.

2.2. Tipo de aplicación

A la hora de desarrollar la aplicación en sí, cabe plantearse dos opciones:

- Aplicación nativa
- Aplicación *web*

[1] `http://www.pktsystems.com/es/`
[2] `http://www2.neoptec.com/es/productos/qcm-direct-la-correccion-automatica`

2.2.1. Aplicación nativa

Dado que uno de los objetivos es que la información se almacene de forma centralizada, habría que desarrollar dos partes de la aplicación: un cliente y un servidor.

Esto implicaría que para cada uno de ellos habría que diseñar:

- Interfaz de usuario
- Mecanismos de autenticación/autorización
- Protocolo de comunicación
- Sistemas de seguridad de red (control de ataques de fuerza bruta contra las claves de usuario, mecanismos anti-*DDoS -Distributed Denial of Service-*, etc)

Además conlleva la necesidad de actualizar personalmente el software como respuesta a nuevas amenazas de seguridad, modificaciones de restricciones o protocolos.

2.2.2. Aplicación web

La otra alternativa es plantearlo como una aplicación *web*, apoyándose sobre protocolos, plataformas, e implementaciones ya consolidadas, que ofrecen suficientes garantías a nivel de estabilidad, seguridad y confiabilidad.

El uso de *software* sobradamente probado y testado, y de implementaciones convertidas casi en estándar "de facto", aporta la seguridad de tener una mejor adaptación a cualquier cambio o actualización de seguridad posterior, especialmente en lo relativo a amenazas de red.

De este modo, además, se evita al usuario final la necesidad de instalar un programa cliente específico, ya que el acceso lo realizaría a través de su propio navegador.

Como respuesta al objetivo de centralizar la información en un único punto, parece que la elección de una solución basada en *web* es la que más beneficios ofrece.

2.3. Lenguaje de Programación y plataformas de programación web

Aun centrando la atención en un desarrollo *web* se plantea un amplio abanico de lenguajes de programación candidatos para la implementación del proyecto.

2.3.1. Java

Java[3][19] es un lenguaje de programación multiplataforma, multipropósito, orientado a objetos y que cuenta con una infinidad de librerías y plataformas con las que acometer cualquier proyecto.

Está muy extendido, lo que facilita el encontrar documentación y bibliografía que sirva de guía durante el desarrollo. El hecho además de que sea uno de los lenguajes más empleados en las asignaturas vistas supone otro punto a su favor.

A la hora de plantearse un desarrollo *web* con Java se abren varias alternativas:

Applets

Se trata de una primera aproximación de Java a la *web*[4], y consiste en la ejecución de una pequeña aplicación Java incrustada en una página *web*. A diferencia de otros casos que se verán, requiere la ejecución de una máquina virtual de Java en el cliente, algo que cada vez están restringiendo más los navegadores actuales. Otro problema que plantea es la necesidad de descargar todo el código del *applet* al cliente para su ejecución, cuando quizás únicamente se desee hacer uso de una pequeña parte de la funcionalidad que ofrece.

JavaServer Pages

Dando un paso más en la evolución de aplicaciones *web* basadas en Java aparecen las JavaServer Pages[2]. A diferencia de los *applets*, el código Java que hay tras un JSP se ejecuta en el servidor, y no en el cliente. La persistencia de los hilos de ejecución en el servidor agilizan la carga de las páginas, y al no tener que descargarse el código máquina del programa concreto, también se aligera la carga de red necesaria para poder usar la aplicación.

JavaServer Faces

JavaServer Faces[3] es una plataforma de desarrollo de aplicaciones *web* en Java, que se apoya en JSP pero progresa en que sigue el patrón MVC (*Model View Controller*), al tiempo que forma parte del estándar J2EE[4].

[3] `https://www.java.com/es/`

[4] `https://docs.oracle.com/cd/E19159-01/819-3680/abfar/index.html`

Spring

Spring[5][22] es otra plataforma para implementar aplicaciones Java para la *web*, que sigue el paradigma de programación orientada a aspectos.

Apache Struts 2

Struts 2[6][18] es la última plataforma que se mencionará basada en Java, siguiendo los preceptos de J2EE, e implementando el patrón MVC, aunque de un modo ligeramente distinto a las Javaserver Faces, ya que la gestión de las peticiones se realiza a nivel de controlador, y no de vista.

2.3.2. PHP

PHP[7][20] es un lenguaje orientado a la *web*, aunque también permite la implementación de *scripts* locales interpretados. De forma similar a las JavaServer Pages, facilita combinar la lógica de la aplicación con el aspecto gráfico de la misma, incrustando fragmentos de código dentro de la programación HTML.

CakePHP

CakePHP[8][10] es un marco de desarrollo *web* basado en PHP que pretende facilitar el desarrollo y la implementación del paradigma MVC. De esta forma separa claramente la interacción con la base de datos, la lógica de negocio y la presentación final al usuario. La forma de gestionar las peticiones, al igual que Struts, se hace a nivel del controlador.

Sympfony

Sympfony[9][24] es otra plataforma de desarrollo para PHP que implementa MVC, y proporciona al programador una amplia variedad de librerías y tareas comunes ya incorporadas, con el objetivo de agilizar el desarrollo.

[5] `https://spring.io/`
[6] `http://struts.apache.org/`
[7] `https://secure.php.net/`
[8] `http://cakephp.org/`
[9] `http://symfony.es/`

2.3.3. Ruby

Ruby[10][8] es un lenguaje de programación multipropósito y dinámico que pretende simplificar el código y hacerlo fácilmente legible. Además permite desarrollar el código de forma funcional o imperativa, y es un lenguaje fuertemente orientado a objetos de herencia simple.

Ruby on Rails

Ruby on Rails[11][14] es posiblemente uno de los factores que más ha impulsado la propagación de Ruby como lenguaje de programación. Es una plataforma para desarrollo *web* implementada en dicho lenguaje. Como muchos de los marcos de programación anteriormente citados, sigue el patrón MVC, y ofrece toda la estructura necesaria para desarrollar la aplicación. También permite la rápida integración de numerosos módulos complementarios.

2.3.4. .NET

Una alternativa más es la plataforma .NET[9], refiriéndose como tal al *Microsoft Common Language Runtime*, independientemente de que internamente se desarrolle empleando C# o la alternativa que se prefiera. La implementación original está preparada para sistemas operativos Windows, pero existe una implementación libre llamada Mono, y que funciona también en otros sistemas operativos.

2.3.5. Perl

Perl[12][5] es un lenguaje interpretado, principalmente de *scripting*, aunque sus numerosísimos módulos permiten realizar casi cualquier tarea con él. Su uso más extendido es la implementación de *scripts* locales, y sus primeras incursiones en la *web* fueron en forma de CGI's (*Common Gateway Interface*)[13].

Actualmente existen varias plataformas que permiten utilizarlo en entornos *web* de una forma más eficiente y moderna que a través de los *Common Gateway Interfaces*.

[10] `https://www.ruby-lang.org/es/`
[11] `http://rubyonrails.org/`
[12] `https://www.perl.org/`
[13] `http://www.w3.org/CGI/`

Mojolicious

Mojolicious[14] es un marco de desarrollo *web* con énfasis en la ligereza y flexibilidad (no requiere módulos de Perl específicos más allá de las librerías estándar). Se caracteriza por estar pensado para aplicaciones de tiempo real, alto rendimiento, HTTP 2 (*HyperText Transfer Protocol* versión 2) y *web sockets*.

Catalyst

Catalyst[15][6] es una plataforma para la programación de aplicaciones *web* que se apoya en Perl y sigue el patrón MVC, lo que agiliza el desarrollo a costa de exigir una estructura concreta del software a programar.

2.3.6. Python

Python tiene varios puntos en común con Perl, por ejemplo se trata de un lenguaje interpretado, muy empleado como lenguaje de *scripting* para la administración de sistemas, y posee un catálogo inmenso de módulos, librerías y plataformas que le convierten en un lenguaje de propósito general que poco a poco va ganando adeptos.

Con el tiempo también han ido apareciendo marcos de programación *web* a través de los cuales se ha ido haciendo un hueco entre los lenguajes más empleados para la programación de páginas y portales.

Zope

Zope[16][1] fue el primer marco de desarrollo que salió para la *web* basado en Python, y en su momento supuso un avance importante en cuanto al modo en que se concebían estas aplicaciones, hasta tal punto que aún hoy muchas plataformas siguen varias de las pautas en las que Zope fue pionero.

Aunque hoy su desarrollo está estancado y tan sólo se publican algunas actualizaciones de seguridad, destaca por los altos estándares que ha mantenido siempre en este ámbito. Además soporta funcionamiento de edición de contenidos desde el propio portal, localización del interfaz

[14] `http://mojolicio.us/`
[15] `http://www.catalystframework.org/`
[16] `http://www.zope.org/`

de forma nativa, motor de búsqueda integrado y varias características más en las que supuso un hito.

Flask

Flask[17][12] es una plataforma de desarrollo *web* programada en Python que se caracteriza por su minimalismo. En lugar de implementar completamente una pila propia de componentes, se apoya en *software* de terceros, como el lenguaje de plantillas *web* Jinja2, dando opción al desarrollador para elegir los componentes de su preferencia. También destaca por dar un especial grado de libertad a la hora de elegir la forma en la que se desea implementar la aplicación.

Pyramid

Pyramid[18][16] es otro marco de programación *web* para Python, algo más completo y robusto que Flask, y que forma parte del proyecto Pylons. Se orienta hacia las aplicaciones de página única, persiguiendo la máxima simplicidad de código posible, aunque también soporta multitud de módulos y complementos para ampliar la funcionalidad ofrecida.

Django

Django[19][13][11] es el más completo de los marcos de programación en Python para la *web*. Su diseño sigue el patrón MVC, e incorpora de serie todos los componentes necesarios para desarrollar el proyecto (en contraposición a Flask, por ejemplo). Esto permite agilizar el desarrollo al reducir las decisiones a tomar, y marcar en cierto sentido una línea de trabajo concreta, lo que por otra parte le resta flexibilidad. Sin embargo, la facilidad que aporta a la hora de evolucionar los proyectos, y la potencia de sus librerías (tanto las básicas como los múltiples módulos que se le pueden añadir) hace que para aplicaciones de tamaño medio-grande sea la alternativa más popular.

2.3.7. Comparativa

Se puede plantear una comparativa entre las diferentes alternativas analizando diferentes aspectos:

[17] `http://flask.pocoo.org/`
[18] `http://www.pylonsproject.org/`
[19] `https://www.djangoproject.com/`

Plataforma	Lenguaje	MVC	ORM
Applets	Java		
JavaServer Pages	Java		
JavaServer Faces	Java	•	(J2EE)
Spring	Java	•	(J2EE)
Apache Struts 2	Java	•	•
CakePHP	PHP	•	•
Sympfony	PHP	•	•
Ruby on Rails	Ruby	•	•
Mojolicious	Perl	No explícito	(DBIx::CLass)
Catalyst	Perl	•	(DBIx::CLass)
Zope 2	Python	•	•
Flask	Python		(plugin)
Pyramid	Python	•	No explícito
Django	Python	•	•

Tabla 2.1: Comparativa lenguajes y plataformas de programación

Valorando los parámetros analizados, y comparándolo con lo deseado para el desarrollo de la aplicación, la elección lógica es Django, porque:

- Está basado en el mismo lenguaje de programación que Eyegrade, lo que permite una integración inmediata del código de éste.
- Su diseño está basado en el paradigma MVC, lo que facilita y agiliza el desarrollo.
- Al incorporar un ORM (*Object-Relational Mapping*) permite abstraerse del sistema de bases de datos que se emplee por debajo, dando libertad para elegir uno u otro (o incluso cambiarlo más adelante si se hiciera necesario).

2.4. Plataforma de desarrollo para la generación de gráficas

Dado que el objetivo es desarrollar una aplicación que permita generar estadísticas a partir de los datos obtenidos mediante Eyegrade de los exámenes realizados por los alumnos, una parte muy

importante de dicho desarrollo será la presentación de gráficas que faciliten la interpretación de las cifras obtenidas. Centrándose en el mundo *web*, se pueden hacer dos distinciones fundamentales:

- Generación de gráficas en el servidor
- Generación de gráficas en el lado del cliente

2.4.1. Generación de gráficas en el servidor

Consiste en usar código que desde el servidor *web* convierta los datos numéricos en imágenes, que muestren las gráficas correspondientes. Posteriormente estas imágenes se envían al navegador, que las dibujará como parte de la aplicación *web*.

Este método tiene la ventaja de contar con toda la potencia del servidor para generar las gráficas de forma rápida y enviarlas al navegador, que al tener únicamente que mostrar una imagen ve aligerada su parte del trabajo.

Sin embargo presenta el inconveniente de que limita drásticamente las posibilidades de interacción con las gráficas, ya que si se trata de imágenes estáticas, cualquier modificación que se quisiera hacer en respuesta a las acciones del usuario final, requeriría de una petición (síncrona o asíncrona) al servidor para generar la nueva gráfica y presentarla al usuario, refrescando la parte de la página necesaria.

2.4.2. Generación de gráficas en el lado del cliente

Otra alternativa es enviar los datos numéricos al navegador y emplear tecnologías de programación en el lado del cliente (fundamentalmente Javascript[20] y sus diferentes plataformas) para generar las gráficas y presentarlas al usuario final.

Este sistema tiene el inconveniente de depender de la potencia del dispositivo en que se esté visualizando la información para procesar los datos y generar las gráficas. En el caso de acceder desde un ordenador de recursos limitados o un dispositivo móvil poco potente esto puede suponer un cierto retraso en la visualización de la información.

Sin embargo presenta la ventaja de que la interacción entre el usuario y las gráficas es más directa, permitiendo modificarlas "en vivo" sin necesidad de hacer ninguna nueva petición al servidor, permitiendo la inclusión de animaciones y efectos visualmente más atractivos que la recarga de una parte de la página.

[20] `https://www.javascript.com/`

En general, este tipo de funcionamiento suele dar resultados más satisfactorios tanto a nivel visual como de experiencia de uso. Por tanto se optará por esta alternativa.

Algunas de las plataformas más empleados para la generación de gráficas en el lado del cliente son:

- Google Charts
- Raphael.js
- D3
- NVD3

Google Charts

Google Charts[21] es un marco de desarrollo bastante completo, y muy visual. Incorpora las gráficas más utilizadas, y contempla en su implementación la posibilidad de interactuar con ellas.

Admite el paso de datos en formato JSON (*JavaScript Object Notation*), la integración con código Javascript, y ofrece varios complementos para integrarlo con varios lenguajes de programación (Java, C#, Python, PHP,...).

Sin embargo, en el momento de cargar la página, ésta ha de descargar de los servidores de Google parte del código, de forma dinámica, para poder funcionar. Los términos de uso de Google Charts no permiten la descarga de dicho código al disco duro para poder trabajar sin conexión.

Por este mismo motivo, si en el futuro Google decidiera abandonar el servicio, las gráficas quedarían inaccesibles.

Raphael.js

Raphael.js[22][23], más que una plataforma de desarrollo de cuadros y gráficos estadísticos es un marco de programación de gráficos vectoriales en general para la *web*. Esto tiene la ventaja de permitir el desarrollo de gráficas más diversas y personalizadas, pero el inconveniente de que el vínculo entre los datos y la gráfica debe ser implementado explícitamente.

[21] `https://developers.google.com/chart/?hl=es`
[22] `http://raphaeljs.com/`

D3

D3[23][21][17] es una plataforma de desarrollo que aventaja a todos los anteriores en lo que a potencia y versatilidad se refiere, ya que trabaja a un nivel mucho más bajo que ellos. Esto tiene el lado positivo de dar mucho más control sobre el resultado, facilitando el desarrollo de gráficas mucho más personalizadas, completas, detalladas y que cumplan completamente los requisitos establecidos. Sin embargo estas mismas prestaciones se convierten en su principal inconveniente, ya que para una sola gráfica requiere de mucho más trabajo (y mucho más código) que cualquiera de las otras alternativas. Además limita la reusabilidad de ese mismo código, ya que cuanto más lo ciñamos a los requisitos de una tarea concreta, más nos alejaremos de lo que podríamos reutilizar en otra gráfica.

NVD3

NVD3[24]no es realmente una plataforma independiente, sino una herramienta que se apoya en D3 para generar gráficas de un modo más directo e intuitivo, facilitando la reutilización del código, y reduciéndolo sensiblemente frente a lo que sería necesario si se trabajara directamente con D3. Sin embargo sigue cediendo acceso completo a las estructuras D3 subyacentes, de forma que se sigue contando con ese control de bajo nivel a los elementos de la gráfica, y de este modo se pueden realizar cuantas modificaciones "a medida" sean precisas.

2.4.3. Elección

Con la información recogida en los apartados anteriores, finalmente se opta por el binomio D3 + NVD3 para la generación de las gráficas, ya que aunan la versatilidad y flexibilidad de D3 con la facilidad de implementación y reutilización de NVD3, al tiempo que mantiene un vínculo implícito entre los datos y la gráfica.

[23] `http://d3js.org/`
[24] `http://nvd3.org/`

Capítulo 3

REQUISITOS

A la hora de acometer la implementación del proyecto se plantean los siguientes requisitos:

RQ1 La custodia de los datos subidos ha de ser centralizada

RQ2 Debe permitir la inclusión de nuevos datos a través de tres ficheros diferentes, aunque interdependientes:

RQ3 Ficheros de examen (.xml) conteniendo las preguntas y opciones de respuesta

RQ4 Ficheros de configuración de examen (.eye) con los diferentes modelos de examen y permutaciones de preguntas/opciones

RQ5 Ficheros de respuesta (.csv) con las respuestas dadas por los alumnos

todo ello en los formatos empleados por la aplicación Eyegrade, de forma que no requieran conversión.

RQ6 Debe generar estadísticas de diversos datos y métricas, que ofrezcan información que resulte de utilidad a la hora de detectar posibles problemas con la asimilación de la materia

RQ7 Las gráficas y estadísticas deben permitir el nivel de interacción que sea posible y en cada caso tenga sentido, incluyendo:

RQ8 Reordenación de los datos de la gráfica para facilitar la localización de máximos/-mínimos u otros datos de interés

RQ9 Acceso a los datos que hay tras la estadística pinchando en la porción correspondiente de la gráfica

RQ10 El acceso a la aplicación ha de hacerse de forma autenticada. Esta autenticación, además deberá ser posible de forma delegada en un servidor LDAP (*Lightweight Directory Access Protocol*). En el caso de la UC3M deberá ser compatible con el servidor LDAP corporativo, permitiendo usar el mismo login y clave que el usuario ya tenga en la Universidad.

RQ11 De ser necesario deberá ser posible la creación de usuarios locales que no existan en el servidor LDAP.

RQ12 Deberán distinguirse dos tipos de usuarios, con acceso a apartados concretos de la aplicación:

RQ13: Alumno Deberá tener acceso a sus propias estadísticas y resultados, entre otras:

RQ14 Exámenes realizados (visualizando todas las preguntas y opciones, destacando la opción elegida por el alumno, y también la correcta). Puede ser una forma visual y efectiva de comunicar a un alumno el resultado obtenido en el examen, así como la corrección del mismo.

RQ15 Histórico de respuestas acertadas

RQ16 Histórico de respuestas fallidas, en las que además se mostrará la opción correcta. Esto facilitará el repaso de la materia en caso de tener que presentarse nuevamente a la misma asignatura.

pero únicamente a las suyas. No deberá tener acceso a los resultados de ningún compañero, ni a otras estadísticas que no sean estrictamente suyas.

RQ17: Profesor Deberá tener acceso a las estadísticas de los exámenes que él mismo suba a la herramienta más aquellos a los que se le conceda acceso. Entre otras gráficas, podrá ver:

RQ18 Preguntas escritas por cada profesor. Puede ser interesante a modo de histórico, así como de repaso de posibles preguntas ya planteadas en otras convocatorias (bien para facilitar su reutilización o bien para evitarla).

RQ19 Porcentaje de respuestas que han sido asignadas a cada opción de cada pregunta, incluyendo el porcentaje de respuestas en blanco. De este modo se podrán localizar las preguntas que causan más dificultades o confusión a los alumnos.

RQ20 Número de respuestas acertadas/fallidas por cada alumno. Al pinchar en la porción de la gráfica se visualizará el texto de las preguntas acertadas/fallidas. De

esta forma se puede hacer un rápido repaso del histórico del alumno. Si además se ordena la primera gráfica por número de aciertos, se podrá ver en qué se suelen equivocar con más frecuencia los alumnos que obtienen mejores resultados, y al revés.

El profesor además deberá ser capaz de:

RQ21 Borrar exámenes que haya subido (sólo si es él mismo quien lo subió)

RQ22 Conceder/quitar acceso a los exámenes que haya subido a otros profesores (sólo si es él mismo quien lo subió)

RQ23 Sin estar autenticado no se deberá tener acceso a ninguna gráfica ni dato dentro de la aplicación.

RQ24 El acceso a los datos se hará mediante peticiones tipo REST (*REpresentational State Transfer*)

RQ25 Los datos deberán poder filtrarse por diversos criterios, entre los que estarán:

RQ26 Por profesor autor de la pregunta

RQ27 Por examen

RQ28 Que no necesite de programas específicos para su uso.

Capítulo 4

ARQUITECTURA DEL SISTEMA

En este capítulo se expondrán los diferentes componentes *software* que hacen posible que la aplicación lleve a cabo su cometido.

4.1. Arquitectura general

Los principales bloques que se pueden distinguir son a nivel de servidores por un lado, y a nivel de requisitos *software* en el lado del usuario por otro.

Se verá más claramente en una imagen:

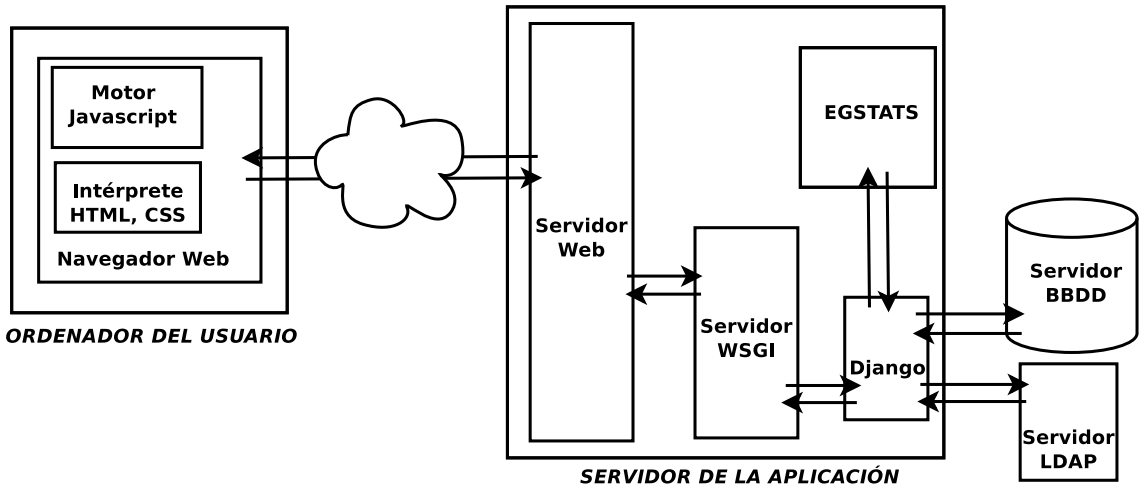

Figura 4.1: Arquitectura

Como se puede observar, en el lado del cliente únicamente es necesario un navegador capaz de interpretar correctamente código HTML, CSS (*Cascading Style Sheets*) y que tenga un motor

de Javascript. Actualmente todos los navegadores modernos cumplen estos requisitos.

En el lado del servidor de la aplicación se pueden distinguir varios componentes software que se necesita que interactúen para llegar al resultado deseado:

Servidor web Será el responsable de recibir las peticiones HTTP, realizar las comprobaciones de seguridad pertinentes, identificar a qué recurso *web* corresponde la petición, y pasársela. Para cumplir este papel se podría usar Apache[1] o Nginx[2].

Servidor WSGI Puesto que la aplicación está programada en Python empleando a Django como plataforma de desarrollo, se necesita un elemento *software* que sea capaz de intermediar entre el servidor *web* y los procesos Python necesarios. Esta función la realiza un servidor WSGI (*Web Server Gateway Interface*), cuya responsabilidad es lanzar (o mantener) los intérpretes de Python necesarios para atender las peticiones que le lleguen desde el servidor *web*, y pasárselas al servidor de aplicaciones Python. En caso de usar Apache, ya dispone de un módulo para este fin (`mod_wsgi`), y en caso de usar Nginx se podría instalar `uWSGI`.

Servidor de aplicaciones web En este caso el elegido es Django, quien recibe las peticiones *web* a través del servidor WSGI y realiza todas las operaciones necesarias para conformar una respuesta y facilitársela para que siga el camino inverso hasta el navegador del usuario. El servidor *web* ya ha hecho las comprobaciones de seguridad básicas (filtrado de IPs conocidas como maliciosas, control del ritmo de peticiones para evitar DDoS, detección de peticiones mal formadas, etc.), pero le corresponde a Django realizar las funciones de autenticación y autorización.

Servidor de Bases de Datos El ORM de Django utiliza un sistema de gestión de bases de datos para almacenar la información relativa a los modelos de la aplicación, así como de otros datos que necesite almacenar el servidor para su funcionamiento (como los correspondientes al superusuario, necesario para realizar tareas de gestión del servidor). Actualmente Django soporta SQLite[3], PosgreSQL[4] y MariaDB/MySQL[5] de serie, aunque hay módulos que permiten utilizarlo sobre bases de datos NoSQL (como mongoDB[6], o el sistema de

[1] `https://httpd.apache.org/`
[2] `https://www.nginx.org/`
[3] `https://www.sqlite.org/`
[4] `http://www.postgresql.org/`
[5] `https://mariadb.org/`
[6] `https://www.mongodb.org/`

bases de datos empleado en Google App Engine).

Servidor LDAP Como parte del diseño de la aplicación se ha optado por utilizar un servicio LDAP para la autenticación, permitiendo a los usuarios conservar su login y clave que ya utilicen en otras aplicaciones. Por tanto el servidor Django debe recurrir a un servidor LDAP para confirmar que el usuario que intenta acceder al programa existe, y que la clave que facilita efectivamente se corresponde con la almacenada en el servidor.

Egstats El código del proyecto se articulará como una aplicación dentro de Django, componiéndose de varios ficheros de código Python, especificados por la estructura de Django y desarrollados desde cero para implementar las funcionalidades marcadas como requisitos.

A través de las herramientas provistas por el servidor de aplicaciones, este código accederá a LDAP, a las bases de datos y a los recursos del servidor.

Dentro de este apartado quedarían también incluidos otros ficheros (hojas de estilo CSS, librerías Javascript, etc), que si bien no han sido desarrollados específicamente para este proyecto, forman parte del código empleado por Egstats para llevar a cabo su cometido.

Capítulo 5

DISEÑO

En este capítulo se intentarán plasmar las principales piezas que componen el proyecto, cómo se relacionan entre sí y cómo se utilizan para que la aplicación cumpla su función.

5.1. Patrón MVC

Al tratarse de una aplicación *web*, se decide optar por el patrón de diseño MVC, por facilitar el desarrollo de la funcionalidad futura. Esta forma de trabajo define tres componentes bien diferenciados:

5.1.1. Modelo

Se trata de la parte del diseño centrada en especificar los datos con los que trabajará la aplicación, cómo se estructuran, cómo se relacionan entre sí, y de qué modo se podrá acceder a ellos. El esfuerzo aquí debe centrarse en que el modelo de datos resultante evite el almacenamiento redundante de información, al tiempo que mantenga todos los datos necesarios accesibles y correctamente relacionados, de forma que se pueda vincular cada parámetro a cualquier otro que pueda hacer falta para generar las estadísticas.

Los datos con los que se tendrá que trabajar son todos los elementos, parámetros y circunstancias reflejados en los ficheros de Eyegrade, ya descritos en el capítulo 2.

Centrándose en una visión entidad-relación, se extraen las entidades intervinientes:

Titulación cada una de las titulaciones en cuyas asignaturas se use Eyegrade como método de corrección.

Asignatura cada una de las asignaturas en cuyos exámenes se use Eyegrade.

Examen serán cada una de las realizaciones de un examen.

Permutación contendrá una de las diferentes variantes de exámenes resultantes de reordenar preguntas y respuestas.

Pregunta cada una de las preguntas planteadas, con su texto, y si procediera, código fuente.

Profesor será cada docente que tenga relación con el examen. Se considerará que el examen puede estar redactado por varios profesores, repartiéndose las preguntas.

Autorización representará los permisos (como responsable del examen o como autorizado a visualizarlo) que tendrán los profesores implicados.

Etiqueta cada una de las temáticas sobre las que versen las preguntas.

Elección cada una de las posibles soluciones a cada pregunta.

Respuesta esta entidad es necesaria para incorporar el concepto de modelo de examen relleno por el alumno.

Alumno será cada uno de los estudiantes que realicen un examen corregido con Eyegrade.

Se pueden ver las relaciones existentes entre cada entidad, así como su cardinalidad, en la figura 5.1.1

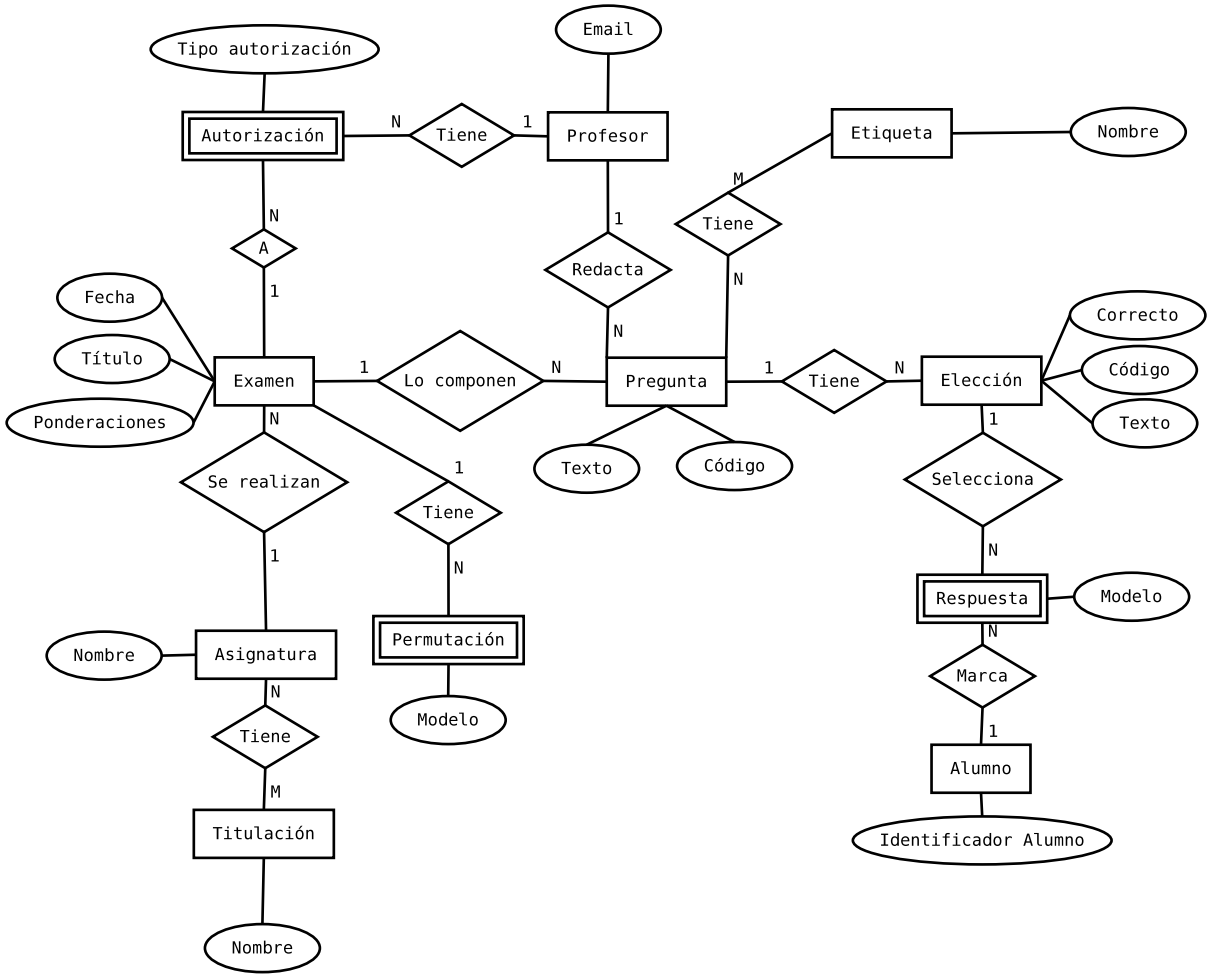

Figura 5.1: Diagrama Entidad-Relación

5.1.2. Controlador

Se trata del "cerebro" de la aplicación, la parte encargada de acceder a los datos necesarios para responder a las peticiones o acciones del usuario, tratarlos como sea necesario, gestionar cualquier información adicional, y facilitar a la vista la información necesaria para presentar el resultado final. En este sentido se trabajará en diseñar un sistema de gráficas reutilizable y flexible.

Los principales controladores de la aplicación serán:

Controlador de direcciones es un controlador implícito en el código de Django que se encarga de gestionar las peticiones, identificando mediante expresiones regulares a qué controlador

debe enviar la petición.

Comprobación de validación también se trata de un controlador implementado en el propio código de Django que permite comprobar, antes de entrar al código de un controlador concreto, si el usuario tiene una sesión válida, para poder decidir si se le da acceso o si se le envía a la pantalla de validación.

Obtener datos para la gráfica es el responsable de atender las peticiones tipo REST en las que se solicitan los datos para la gráfica. Recogerá los parámetros requeridos y formará la respuesta tipo JSON a devolver.

Subir fichero de respuestas se encargará de incorporar al ORM los datos de repuestas del fichero seleccionado por el usuario.

Subir fichero de configuración recogerá el fichero de configuración seleccionado y añadirá los datos al ORM.

Subir fichero de examen añadirá al ORM los datos contenidos en el fichero elegido por el usuario.

Mostrar examen recopilará todos los datos necesarios para calcular las estadísticas básicas del examen y se los facilitará a la vista correspondiente.

Mostrar datos de examen reconstruirá a partir del ORM el examen para facilitárselo a la vista correspondiente.

Borrar examen eliminará el examen solicitado

Gestionar autorización para examen permitirá añadir o eliminar autorizados a visualizar los datos de un examen.

Mostrar profesor recopilará la información necesaria para mostrar las estadísticas básicas de un profesor, y se lo pasará a la vista responsable de su visualización.

Mostrar datos de un profesor localizará en el ORM todas las preguntas redactadas por un profesor (a las que tenga acceso el usuario autenticado) y se las facilitará a la vista correspondiente.

Mostrar alumno recogerá del ORM los datos necesarios para calcular las estadísticas básicas del alumno.

Mostrar datos del alumno en función de lo solicitado, recogerá la información relativa a las preguntas acertadas, fallidas o no contestadas por el estudiante para permitir su visualización a través de la vista correspondiente.

Mostrar estadística será el responsable de recibir la petición de una estadística concreta, interpretarla, identificarla, y preparar la información necesaria para la vista. Al ser un controlador muy reutilizado, una de sus principales funciones será la de modificar los textos y títulos que necesita la vista para mantenerse coherente a la petición concreta.

Validar usuario se encargará de comprobar si el usuario que solicita acceso a la aplicación debe tenerlo, y en su caso, con qué perfil. En caso de ser la primera vez que el usuario accede, recopilará a través de una petición LDAP los datos necesarios.

Comprobar permisos es una función que no lleva a una vista concreta, sino que es consultada por todos los demás controladores para confirmar que el solicitante tiene acceso a la información que desea ver.

Cerrar sesión permitirá a un usuario desvincularse de la sesión activa en la aplicación.

Generar barra de navegación es un controlador al que llaman otros para obtener la barra de navegación que se debe mostrar al usuario. Almacena datos en la sesión abierta para facilitar una navegación fluida y coherente.

5.1.3. Vista

Es la parte de la aplicación que define la visualización que se presentará al usuario, separando toda información relativa al aspecto gráfico de los datos propios de la aplicación, y de la parte de la misma que se encargará de gestionarlos. Al trabajar en esta parte se hará especial énfasis en buscar un aspecto limpio, sencillo y atractivo, al tiempo que se tratará de facilitar cualquier mejora estética futura mediante la separación del código responsable de este tema de las demás piezas del patrón MVC.

Las principales vistas serán:

Validación es la que permitirá a los usuarios autenticarse.

No autorizado informará a los usuarios que están tratando de acceder a una información a la que no tienen acceso. Lo habitual será que aparezca cuando intenten copiar y pegar una URL (*Uniform Resource Locator*), o modificarla para llegar a otros datos que los que se les ofrecen.

Mostrar estadística será una vista muy reutilizada, ya que es la encargada de mostrar las gráficas, independientemente de su tipología o datos a mostrar.

Mostrar lista de estadísticas mostrará un índice de los tipos de estadísticas a los que se puede acceder.

Mostrar estudiante permitirá la visualización de estadísticas del alumno, incluyendo enlaces a otras vistas propias del alumno.

Mostrar datos del estudiante mostrará las preguntas acertadas, fallidas, o sin responder del alumno, dependiendo de cómo se invoque a esta vista.

Mostrar examen realizado por estudiante permitirá ver las respuestas seleccionadas por parte del estudiante en un examen concreto, destacando las acertadas y las fallidas, y en su caso la opción que habría sido la correcta.

Mostrar profesor plasmará en pantalla los datos básicos del profesor, con sus estadísticas esenciales.

Mostrar datos del profesor mostrará las preguntas redactadas por el profesor.

Mostrar examen será la vista responsable de mostrar la información básica del examen, junto con las estadísticas de notas obtenidas durante el mismo, y enlaces a las acciones a disposición del usuario (que dependerán del nivel de acceso que tengan a esa prueba en concreto).

Mostrar datos del examen presentará en pantalla el texto del examen, con las elecciones disponibles para cada pregunta, remarcando la correcta.

Gestionar examen permitirá añadir/eliminar autorizados a visualizar el examen, así como el borrado del mismo.

Subir examen permitirá elegir el fichero que contiene las preguntas del examen para incorporarlo a la base de datos.

Subir configuración de examen será la vista responsable de facilitar la selección del fichero .eye para añadir esta información a la ya disponible en la aplicación de las preguntas del examen.

Subir fichero de respuestas facilitará la selección del fichero CSV que contiene las respuestas de los alumnos para un examen dado.

Borrar examen permitirá confirmar el borrado de un examen.

A continuación se muestra un diagrama en el que se pueden ver las principales vistas y controladores. A fin de facilitar la visualización, la gráfica no es exhaustiva.

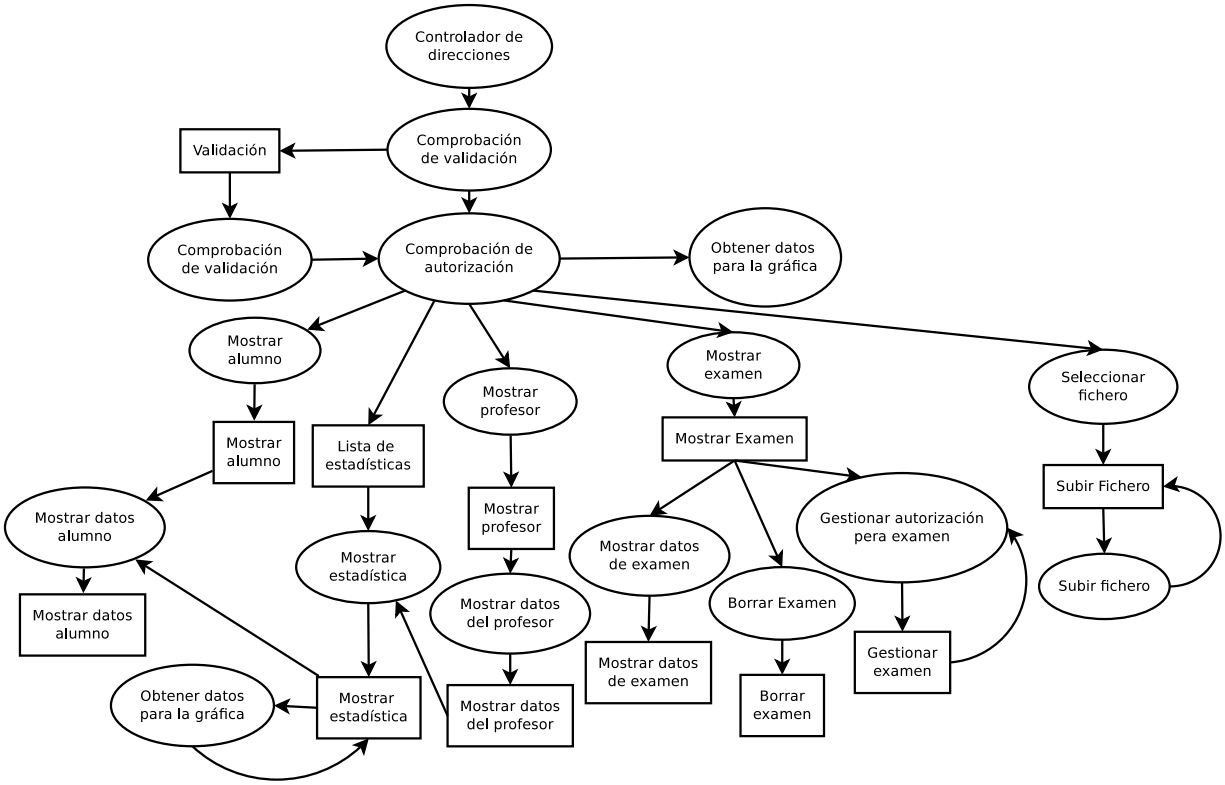

Figura 5.2: Vistas y controladores

5.2. Usuarios, autenticación y autorización

El objetivo de la aplicación es que sea empleada por unos usuarios para obtener métricas y resultados a partir de la información de exámenes aportados. Por tanto será necesario definir qué

perfiles de usuario existirán, de qué forma se distinguirá a unos usuarios de otros, y cómo será posible garantizar que ninguno acceda a información que no le corresponda.

5.2.1. Usuarios

Se contempla el acceso a la plataforma por parte de dos tipos de usuarios:

Profesores

Será el tipo principal de usuario, ya que el objetivo de la aplicación es presentar estadísticas de exámenes a fin de facilitar la localización e interpretación de patrones en los resultados (preguntas más falladas, preguntas más acertadas, distribución de notas,...).

Alumnos

Los alumnos también serán capaces de acceder a la aplicación, aunque su visión de ella se restringirá a los datos relativos a sus propias respuestas. En este sentido serán muchas menos las gráficas a las que tengan acceso.

5.2.2. Autenticación

Puesto que en el programa se almacenarán datos que podrían ser considerados sensibles, es importante que se garantice que quien accede sea quien diga ser. Para ello se diseña un mecanismo de autenticación basado en LDAP, de modo que el par usuario + contraseña se coteje en un servidor centralizado. Al hacerlo de este modo se evita al usuario tener que recordar una contraseña más, al tiempo que se evita el acceso de personal ajeno a la comunidad universitaria.

No obstante, dado que en determinadas ocasiones podría ser necesario el acceso en circunstancias que impidan el cotejo del usuario y clave contra el servidor LDAP, se diseña una segunda opción, con almacenamiento de los datos en el propio programa, accediendo únicamente a nivel de código.

5.2.3. Autorización

No es importante únicamente que el usuario que accede sea quien dice ser, sino también que se pueda garantizar que solamente pueda acceder a los datos a los que legítimamente tiene acceso.

Para ello se hace un diseño de a qué vistas y datos deberá tener acceso cada perfil, y dentro de cada perfil, cada usuario:

Alumno Tendrá acceso a:

- Sus propias estadísticas de respuestas acertadas/fallidas/en blanco.
- Sus propias respuestas a cada examen, junto con las opciones correctas en caso de que su respuesta haya sido fallida

Profesor Podrá visualizar:

- Todas las estadísticas relativas a los exámenes que suba
- Todas las estadísticas relativas a los exámenes a los que se le dé acceso
- Estadísticas y datos relativos a los alumnos que hayan participado en los exámenes a los que tenga acceso por los criterios anteriores
- Estadísticas y datos relativos a los profesores que hayan participado (redactado preguntas) en los exámenes a los que tenga acceso por los criterios anteriores

Además podrá:

- Subir exámenes (preguntas, configuración y respuestas).
- Borrar exámenes que él mismo haya subido.
- Modificar las autorizaciones a los exámenes que haya subido, añadiendo o eliminando profesores autorizados a ver la información relacionada.

5.3. Presentación gráfica

Otro apartado que merece especial interés en el tema de arquitectura es el de la presentación gráfica de la aplicación, ya que en él intervienen varios componentes *software* dignos de mención.

5.3.1. Estilo general

Para el estilo general de la página se ha utilizado Material Design Lite[1], un pequeño marco de desarrollo que ayuda a conferir a nuestra aplicación un aspecto acorde a la filosofía Material

[1] `http://www.getmdl.io`

Design[2]. Se ha optado por este estilo porque es limpio, versátil, y la mayor parte de usuarios potenciales de la aplicación estarán familiarizados con él al ser el empleado por Android desde su versión 5. Además facilita enormemente cambiar el tema cromático de la aplicación con apenas modificar unas líneas.

5.3.2. Generación de gráficas

Como ya se ha comentado en otros puntos de la presente memoria, las gráficas son las protagonistas del proyecto, por lo que merecen especial atención.

Ya se ha comentado que se generan a través de dos plataformas: D3 y NVD3. Para entender mejor cómo interactúan las piezas lo más práctico es verlo en un diagrama:

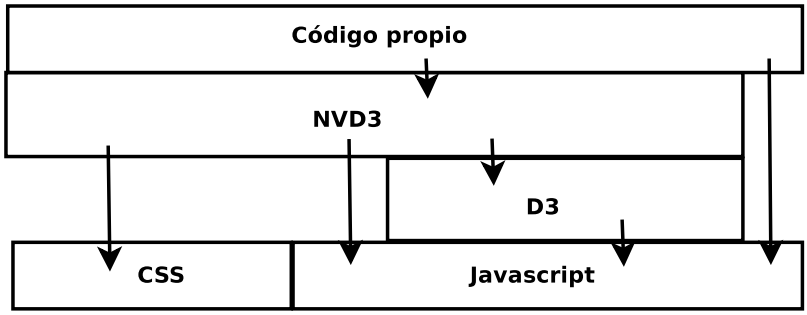

Figura 5.3: Arquitectura software de las gráficas

En el apartado 6.7 se entrará a describir en mayor detalle cómo se implementará esta estructura.

[2] `https://www.google.com/design/spec/material-design/introduction.html`

Capítulo 6

IMPLEMENTACIÓN

En este apartado se intentarán presentar los aspectos más relevantes de la implementación, haciendo especial énfasis en aquellos que por haber supuesto un mayor reto han requerido un esfuerzo significativo en el desarrollo del proyecto.

6.1. Usuarios

A la hora de implementar los diferentes tipos de usuario se ha intentado mantener de la forma más sencilla y limpia posible:

Alumnos Se considerará como alumno todo usuario cuyo identificador de usuario sea únicamente numérico.

Profesores Se considerará como profesor a todo aquel cuyo login no sea estrictamente numérico.

En ambos casos la autenticación y autorización se harán como se explica en secciones siguientes. Para integrar la autenticación basada en LDAP, la posibilidad de establecer usuarios locales, y los perfiles reflejados en el modelo de la aplicación lo que se ha hecho ha sido enlazarlo todo a través del modelo de usuario definido por Django. De este modo el correo de los profesores y el login de los alumnos permite asociar cada usuario de Django a un alumno o profesor del sistema.

6.2. Autenticación

La autenticación, como parte de los requisitos debe permitir la comprobación del par login+contraseña contra un servidor LDAP, que en este caso será el servidor LDAP corporativo

de la UC3M. De este modo los usuarios podrán acceder empleando el mismo login y clave que usan para el resto de servicios electrónicos de la Universidad, haciendo innecesario recordar una nueva clave. Puesto que esta es una funcionalidad soportada por uno de los módulos disponibles para Django, basta con configurar los parámetros adecuados:

- Servidor LDAP
- DN (*Distinguished Name*) base del árbol LDAP
- Filtro de búsqueda para el usuario a validar

El funcionamiento de esta autenticación se hace de la forma siguiente:

1. Se realiza una búsqueda anónima para localizar el DN (ubicación exacta del usuario buscado dentro del árbol LDAP). Si la búsqueda no devuelve ningún resultado (o más de uno), se considera que la autenticación ha fracasado.

2. Una vez conocido el DN del usuario, se intenta realizar una nueva conexión al servidor LDAP, esta vez intentando autenticarse con el DN y la clave facilitada por el usuario en el formulario de acceso. Si el servidor LDAP acepta la nueva autenticación, se da acceso al usuario.

Cuando la autenticación es correcta, se registrará además el correo electrónico del usuario. Lo podemos ver en la siguiente figura:

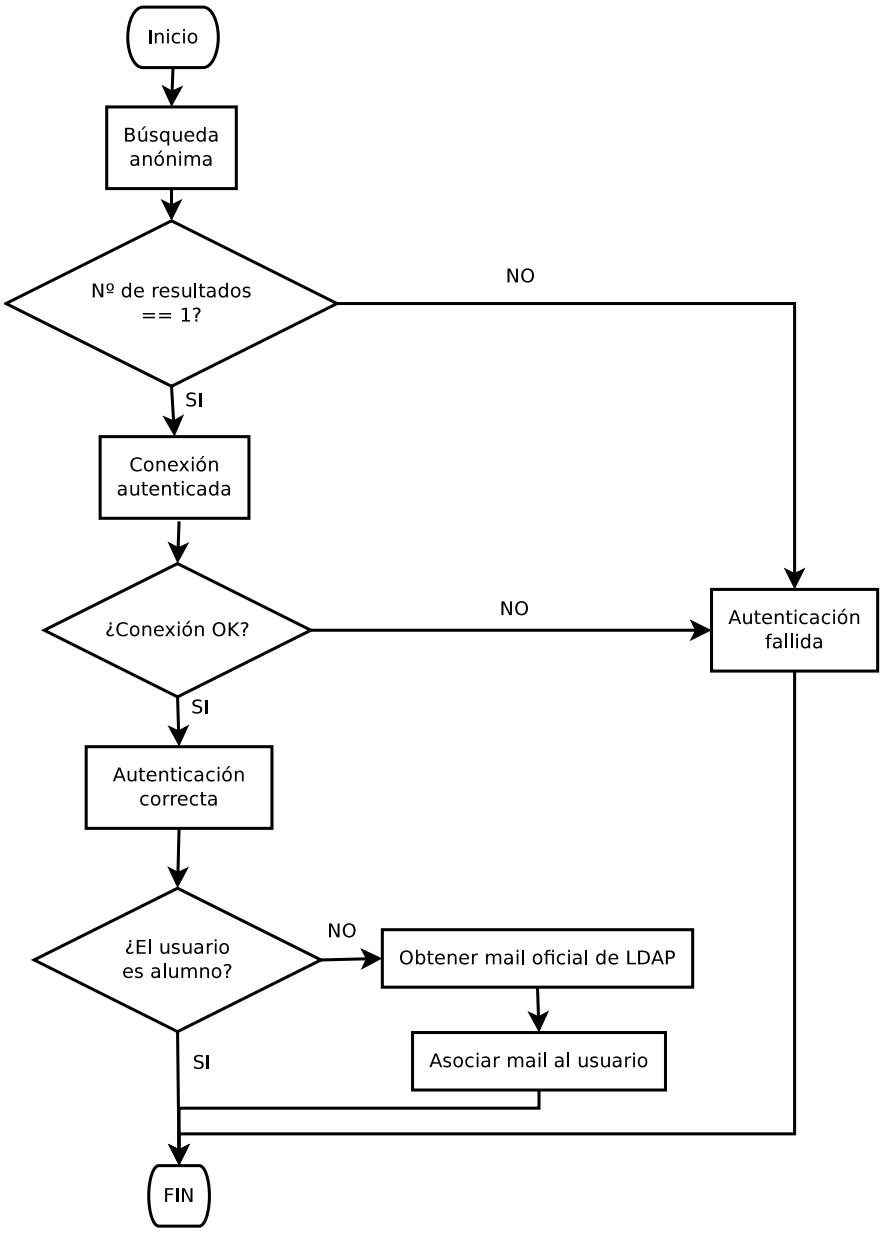

Figura 6.1: Autenticación

6.3. Autorización

Los mecanismos de autorización implementados por Django están diseñados para separar privilegios a nivel de clases en su ORM, permitiendo por ejemplo que ciertos usuarios pudieran ver cierto tipo de datos y no otros, o que pudieran visualizarlos pero no modificarlos de ningún modo.

Sin embargo en este proyecto es necesario que diferentes perfiles accedan a los mismos tipos de datos, estableciendo los límites en cuanto al dato concreto (por ejemplo, tanto profesores como alumnos deben ser capaces de visualizar los resultados de los exámenes, con la diferencia de que el alumno únicamente puede ver sus exámenes).

Este es un tipo de diferenciación que Django no permite hacer "de serie", algo muy específico a las necesidades de este proyecto y que por tanto fue necesario desarrollar de cero.

Para resolverlo hubo que implementar una función de comprobación de privilegios a la que se llamase ante cualquier petición que se hiciera a la aplicación, afectando tanto a las vistas *web* como al servicio REST que genera los datos que luego se representan en las gráficas.

Hay que tener en cuenta que al llegar a esta llamada ya se ha hecho efectiva la fase de autenticación, de forma que si la petición no está debidamente autenticada, antes siquiera de hacer esta comprobación, ha recibido una negativa a su intento de acceder a la información, y se le ha presentado la pantalla en la que se le solicita login y clave.

Comprobando el usuario autenticado, y la URL solicitada, la función actúa de la siguiente manera:

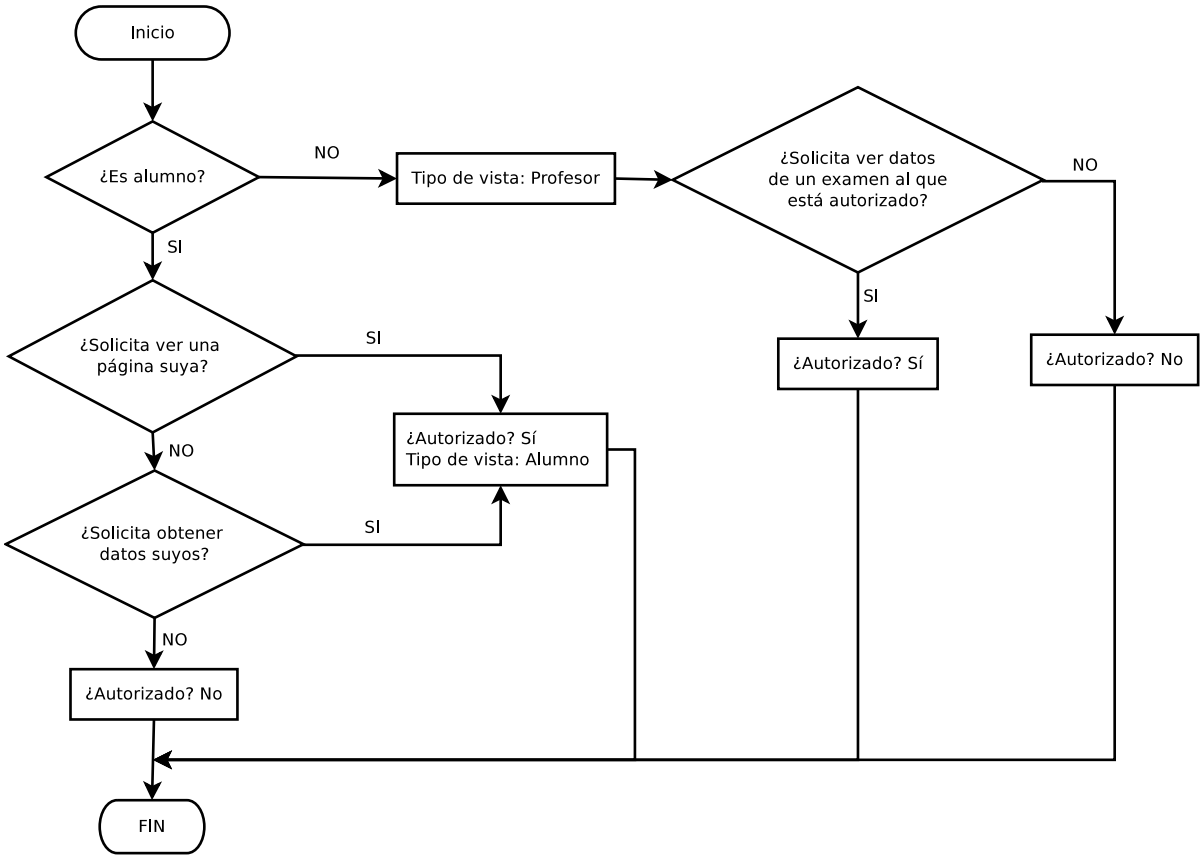

Figura 6.2: Autorización

6.4. Implementación del patrón MVC con Django

Para el desarrollo del proyecto se opta por Django como plataforma *web*. Esta decisión encaja perfectamente con la opción de emplear el patrón MVC en el diseño de la aplicación, ya que la propia arquitectura de esta plataforma de desarrollo *web* sigue este paradigma.

6.4.1. Modelo

El modelo de la aplicación reside en el fichero `Models`, en el que se establecen las entidades y relaciones entre ellas. Estas serán las definiciones que regirán el almacenamiento de la información.

Models

En este fichero se deben representar las entidades y relaciones (con sus cardinalidades) que permitirán un almacenamiento estructurado de toda la información necesaria para que la aplicación cumpla su cometido. Django utiliza un ORM que permite abstraerse de los detalles de la interacción con el sistema gestor de bases de datos que se emplee por debajo. Esto tiene la ventaja de que en un futuro se puede cambiar el sistema de bases de datos con modificar unas pocas directivas en la configuración del programa, pero tiene el inconveniente de que las peticiones a la base de datos pueden ser menos óptimas que si se diseñan de forma específica al gestor de bases de datos concreto que se esté empleando en cada momento.

Para ilustrar el modo en que el ORM representa la citada abstracción, se muestra una clase del ORM a modo de ejemplo:

```
class Question(models.Model):
    exam = models.ForeignKey(Exam)
    teacher = models.ForeignKey(Teacher)
    tag = models.ManyToManyField(Tag, blank=True)
    text = models.CharField(max_length=2084, blank=True)
    code = models.CharField(max_length=4096, blank=True)
    order = models.IntegerField()
```

En este fragmento de código se declara una nueva clase (que vendría a traducirse en una tabla de la base de datos) que extiende a `Model` y que tiene como atributos (columnas):

- `exam`, que equivale a un clave foránea en la base de datos, y que representa la relación 1 a N entre un examen y las preguntas que lo componen. `Exam` ha de ser otra clase declarada en el modelo.

- `teacher`, que es otra clave foránea, indicando la relación 1 a N entre un profesor y las preguntas que ha redactado. `Teacher` ha de haber sido declarada en el modelo. Podría quedar vacío.

- `tag` indica la relación N a M entre las preguntas y las etiquetas que se le asignen. En este caso, esta relación puede quedar en blanco. Previamente se debe haber declarado `Tag` como una clase del modelo.

- `text` marca la existencia de un atributo (columna en la base de datos) de tipo literal, con una longitud máxima de 2084 caracteres, y que albergará el texto de la pregunta.
- `code` representa un nuevo atributo de tipo literal, que podría quedar en blanco y de no ser así albergaría el texto de tipo código fuente que pudiera necesitar la pregunta.
- `order` viene representado por un entero que indica el orden que sigue esta pregunta en el texto original del examen.

6.4.2. Controlador

Como se comentó en el apartado 5.1.2, el controlador es el responsable de toda la lógica de negocio del programa, recibiendo las peticiones de los usuarios, accediendo a los datos necesarios para atenderlas, realizando todas las operaciones pertinentes, y enviando la información necesaria a la vista para que se encargue de la visualización final.

Todas estas tareas se reparten entre los siguientes archivos:

- URLs
- Views
- Ficheros de Eyegrade
- Fichero auxiliar
- Ficheros js

URLs

Este fichero será el encargado de analizar la URL solicitada en cada petición, y seleccionar la función que se encargará de su tratamiento. Utiliza un sistema de expresiones regulares, dotando al sistema de bastante flexibilidad y versatilidad.

Se puede ver un poco más claramente con el siguiente fragmento de código:

```
from django.conf.urls import url


from . import views
```

```
urlpatterns = [
    url(r'^Exam$', views.select_exam, name='select_exam'),
    url(r'^Exam/(\d+)$', views.show_exam, name='show_exam'),
    url(r'^Exam/(\d+)/data$', views.show_exam_data, name='show_exam_data'),
    url(r'^Exam/(\d+)/manage$', views.manage_exam_auth, name='manage_exam_auth'),
    url(r'^Exam/(\d+)/delete$', views.delete_exam, name='delete_exam'),
    ...
]
```

Se puede ver claramente las expresiones regulares que aplicadas a la parte de la ruta en la URL solicitada acaban marcando a qué función controladora se reenviará la petición.

Como en cada ocasión en que se emplean expresiones regulares, hay que ser cuidadosos a la hora de diseñarlas si no se quiere llegar a resultados indeseados (como el que ocurriría si en la primera expresión regular que se ve no se pusiera el '`$`' final).

Cualquier URL solicitada que no se corresponda a ninguna de las expresiones regulares recogidas en este fichero será rechazada con un error 404 (página no encontrada).

Views

Este fichero contiene las funciones a las que Django entregará las distintas peticiones, según el reparto especificado en el fichero de URLs. Aunque su nombre podría llevar a asociarlo a la función de "vista" de un modelo MVC tradicional, no es esta su tarea. En este fichero es donde además se establecerán las diferentes restricciones de acceso, tanto a nivel de autenticación como de autorización. Una vez satisfechos todos los requisitos, estas funciones pondrán a trabajar a la lógica de la aplicación para facilitar al usuario la información solicitada. Una vez obtenidos los datos necesarios, se los entregará al sistema de plantillas para llegar a la representación gráfica de los mismos.

Para entender cómo interaccionan estos controladores con el ORM de Django, y con las vistas lo mejor es ver algunos ejemplos de código:

Insertar un objeto en el ORM Suponiendo que:

- `question` es un diccionario que contiene los datos de la pregunta extraídos del fichero XML de preguntas.

- `format_string` es una función que sustituye códigos específicos de Latex y otras etiquetas que pudieran estar presentes en el texto por formatos equivalentes para la *web*. El segundo parámetro indica si se deben sustituir los saltos de línea por su equivalente en HTML o no.
- `newExam` es un objeto de tipo `Exam` declarado en nuestro modelo, inicializado en código previo.
- `current_teachers` es una lista de profesores que se usa a modo de caché de los profesores existentes en el ORM.
- `get_teacher_by_mail` es una función que busca en la "caché" un profesor cuyo correo electrónico coincida con el buscado.

Se puede ver el código empleado durante la inclusión de un nuevo examen, en la parte en que se inserta una nueva pregunta en el ORM:

```
newQuestion = Question()
if question.text.text is not None:
    newQuestion.text = format_string(question.text.text[0][1], False).strip()
if question.text.code is not None:
    newQuestion.code = format_string(question.text.code, True).strip()
...
newQuestion.exam = newExam
newQuestion.order = question_order
author = question.author
if author is None or author == "":
    author = "Unknown"
my_teacher = get_teacher_by_mail(current_teachers, author)
if my_teacher is None:
    my_teacher = Teacher(email=author)
    my_teacher.save()
    current_teachers.append(my_teacher)
newQuestion.teacher = my_teacher
newQuestion.save()
```

Este código crea el objeto `newQuestion`, inicializando sus atributos en función de la información contenida en `question`. Cabe destacar que si no existe en el ORM un profesor cuyo correo electrónico coincida con el indicado en la pregunta, se crearía un nuevo objeto de clase `Teacher` antes de referenciarlo en el objeto `Question` que se está creando en este fragmento de código.

Acceder a objetos existentes en el ORM Para recuperar objetos insertados en el ORM se emplea el atributo de clase `objects` que contiene todos los objetos de dicha clase que se encuentren almacenados en el ORM. A este atributo se le pueden aplicar filtros y funciones con equivalencia más o menos directa con peticiones tradicionales en lenguajes tipo SQL (*Simple Query Language*).

Por ejemplo, para obtener el número de preguntas de un examen dado (representado por el objeto `my_exam`) se usaría:

```
num_questions = Question.objects.filter(exam=my_exam).count()
```

También se pueden hacer consultas más complejas, en las que sería necesario hacer un `join`, como al requerir los estudiantes que han participado en ese mismo examen:

```
students = Student.objects.filter(answer__choice__question__exam=my_exam)
```

En este caso, el ORM de Django, antes de resolver la petición ha de hacer internamente una operación de unión de las tablas correspondientes a las clases respuesta (en la que existe la referencia al alumno en forma de clave foránea), elección, pregunta y examen.

Envío de datos a la vista Una vez ejecutada toda la lógica de negocio correspondiente al controlador llega el momento de enviar a la vista la información necesaria para su presentación ante el usuario. Esto se hace con la llamada:

```
return render_to_response('showExam.html', data,
                          context_instance=RequestContext(request))
```

Donde:

- El primer parámetro es el nombre de fichero de la plantilla responsable de la presentación de los datos.

- `data` es un diccionario de Python que contiene toda la información obtenida y procesada por el controlador para que la plantilla pueda mostrarse correctamente. Cada elemento del diccionario puede ser una variable, una lista, u otro diccionario.
- El tercer parámetro es el conjunto de datos de la petición, por si son necesarios en alguna tarea en la propia plantilla.

Integración del código fuente de Eyegrade

Una de las principales ventajas de emplear Python en el desarrollo del proyecto es precisamente la de ser el mismo lenguaje en que está programado Eyegrade. De este modo se puede usar el código fuente de dicho programa para importarlos desde la aplicación y hacer llamadas a sus funciones (como por ejemplo analizar los diferentes ficheros de examen, configuración de examen y respuestas). Para alguna de las tareas será necesario reescribir algo de código, concretamente al analizar el fichero de preguntas del examen (código que se ha aportado al proyecto Eyegrade original). Esta forma de trabajar facilitaría en un futuro adaptar la aplicación a nuevas versiones, tan sólo sustituyendo los ficheros de Eyegrade.

Fichero auxiliar

Para algunas tareas muy repetidas durante la ejecución de Egstats, así como para funciones muy voluminosas que podrían extender demasiado el fichero `views`, se creará un fichero aparte, que hará las veces de "librería de utilidades", ayudando a mantener ordenado y "limpio" el código del fichero antes citado.

Ficheros js

El código NVD3 empleado para la presentación de las gráficas e interacción con las mismas se almacenará en ficheros .js individuales. De este modo se facilitará la reutilización de este código en todas aquellas partes del programa que lo requieran.

6.4.3. Vista

La vista de la aplicación es aquello a través de lo cual se establece el aspecto visual de la misma. Todo lo relacionado con interfaz gráfica se gestiona en este apartado, de forma que un diseñador gráfico podría modificar el código HTML que contienen y rediseñar por completo el

aspecto del programa sin afectar a la ejecución del mismo (siempre y cuando respete las pocas etiquetas propias del lenguaje de plantillas de Django que se corresponden a estructuras de control). Los ficheros en los que se almacenará la información serán:

Plantillas

Se trata de un conjunto de ficheros en los que se detallará toda la información necesaria para establecer cómo se deberá mostrar la información al usuario a través de su navegador. Contienen el código HTML5, así como algunas etiquetas y estructuras de control propias del lenguaje de plantillas de Django, que permite, entre otras cosas:

- Incluir unas plantillas dentro de otras, estableciendo una jerarquía y facilitando llevar a cabo la filosofía de DRY (Don't Repeat Yourself).
- Hacer comprobaciones en la propia plantilla mediante operadores condicionales.
- Recorrer listas (*arrays*, diccionarios, etc.) de elementos directamente en la propia plantilla.

La principal ventaja de estas estructuras es poder separar casi completamente la lógica de la aplicación de su presentación gráfica (ya que si no pudiera hacerse en la propia plantilla, la lógica del programa debería generar el código HTML necesario para incorporarlo en la página final).

Para verlo mejor se presentan algunos ejemplos:

Mostrar el valor de una variable Se hace simplemente incluyendo en el código de la plantilla:

```
{{username}}
```

Este código inserta en ese punto el valor de la variable con nombre "`username`" (que en un programa Python convencional se invocaría como `data.username`, siendo `data` el nombre del diccionario pasado como segundo parámetro a la función `render_to_response`).

A estas variables se les pueden aplicar ciertos filtros para conseguir determinados efectos, por ejemplo:

- `{{texto | safe}}` se emplea para permitir la inserción a través de la variable de código HTML, que de otro modo se vería bloqueado como medida de seguridad.

- Operadores matemáticos, como `{{numvar|add:"5"}}` para sumar 5 a una variable numérica.

Aplicar estructuras condicionales Para mostrar una información u otra en función de alguna condición, se pueden usar estructuras condicionales:

```
{% if view_type == "student" %}
    <!-- mostrar información del perfil estudiante -->
{% else %}
    <!-- mostrar otra información -->
{% endif %}
```

Recorrer una lista de variables Se puede recorrer una lista de variables de longitud indeterminada hasta el momento de ejecución, e incluso recorrer listas de listas. Por ejemplo, el código siguiente permite mostrar en una tabla una lista de pares (listas de dos elementos) pregunta-respuesta:

```
<table>
    <tr><td>Question</td><td>Answer</td></tr>
    {% for value in values %}
        <tr>
            {% for val in value %}
                <td>{{val}}</td>
            {% endfor %}
        </tr>
    {% endfor %}
</table>
```

Recorrer un diccionario Es posible recorrer un diccionario mediante estructuras de control en la propia plantilla, de una forma análoga a la anterior, cambiando solamente:

```
<table>
    <tr><td>Key</td><td>Value</td></tr>
    {% for key, value in info.items %}
```

```
            <tr>
                <td>{{key}}</td>
                <td>{{value}}</td>
            </tr>
        {% endfor %}
    </table>
```

Extender otras plantillas Esto permite crear una estructura de plantillas jerárquica, evitando tener que reescribir cierto código (que sea común a todas las plantillas) tantas veces como plantillas haya.

Para ello se emplea:

```
{% extends 'index.html'}
```

El nombre de fichero indicado es la plantilla "general", en la que se incrustará la que se está editando. Para ello, en la general se ha debido incluir el código

```
...
{% block principal %}{% endblock %}
...
```

Y en la plantilla que se está editando, todo el código que se desea incrustar debe ir entre las etiquetas:

```
{% block principal %}
...
{% endblock %}
```

Cargar otros ficheros En ocasiones puede ser necesario cargar otros ficheros para tener disponibles ciertas funcionalidades. Esto se hace con la orden `load`:

```
{% load staticfiles %}
```

Ficheros CSS

En la implementación de la parte gráfica de la aplicación se usarán varios ficheros CSS que contendrán la información de estilo de cada parte del programa. En particular se emplearán ficheros de estilo correspondientes a la plantilla general, a las gráficas NVD3, y a los componentes de estilo Material Design empleados.

Con un gráfico se podrá ver más claramente cómo se relacionan todos los elementos de modelo, controladores y vistas:

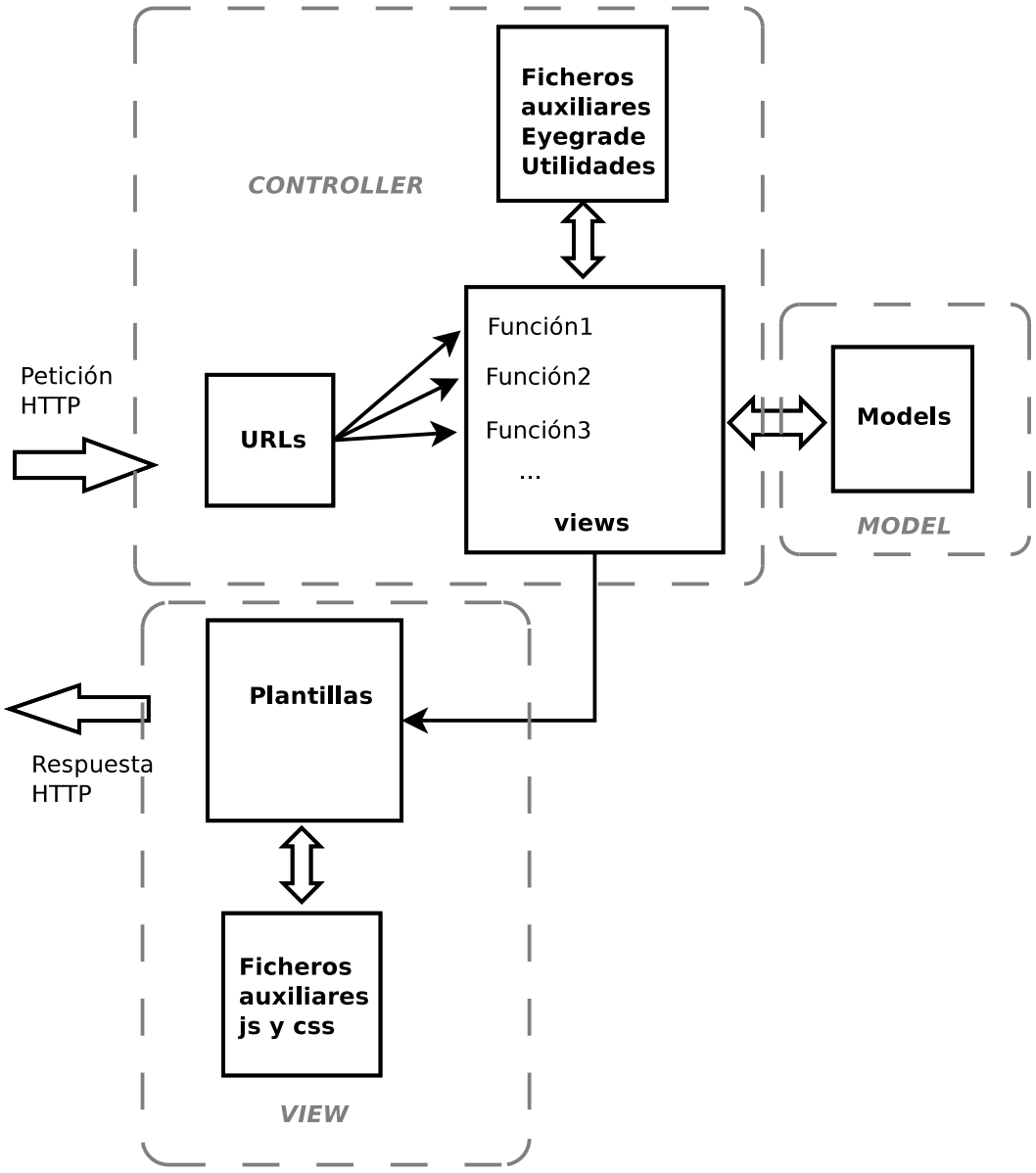

Figura 6.3: Relación entre componentes de Django y el modelo MVC

6.5. Obtención de datos para las gráficas

Con el fin de dotar de la mayor versatilidad posible a la aplicación se ha optado por generar las gráficas mediante la aplicación de una plataforma de programación Javascript en el lado del cliente sobre unos datos obtenidos mediante una petición de tipo REST, que los facilita en una respuesta con formato JSON. Este desacoplamiento permitirá reutilizar las gráficas con más facilidad, haciendo sencilla la incorporación de nuevas estadísticas al proyecto.

Para permitir este uso se han aprovechado las facilidades ofrecidas por el formato de URL empleado en el desarrollo del proyecto. Todos los parámetros requeridos para la extracción y filtrado de los datos pueden pasarse de forma sencilla en la URL, sin necesidad de incorporar parámetros adicionales a la petición HTTP.

De este modo, con el mismo código Javascript, se puede cambiar simplemente la URL con la que se piden los datos para la gráfica, y mostrar una estadística completamente nueva.

El formato de la URL es:

`/egstats/chartData/<tipo_gráfica>/<dato1>/<dato2>/<unordered|ordered>[/filter<datos_filtro>]`

Donde:

egstats es el nombre de la aplicación.

chartData es la forma de indicarle que se está solicitando el JSON con los datos necesarios para generar una gráfica.

tipo_gráfica es el tipo de gráfica que se va a generar (necesario porque cada tipo de gráfica espera los datos con una estructura concreta dentro del JSON).

dato1 es el primer tipo de dato sobre el que se quiere obtener la información (por ejemplo, profesor, examen, pregunta, etc).

dato2 es el segundo tipo de dato sobre el que se desea obtener la información.

unordered|ordered indica si se quiere que los datos se envíen tras ordenarlos previamente. La mayoría de gráficas permite hacerlo una vez generado el conjunto de datos.

filter<datos_filtro> es un parámetro opcional mediante el cual se puede solicitar que los datos se proporcionen previamente seleccionados en base al filtro indicado.

6.6. Colisión de estilos CSS

Al tratarse de una aplicación *web* en la que se hace uso de diferentes marcos de programación y herramientas, uno de los problemas con los que ha habido que lidiar ha sido la colisión de estilos aplicados a los mismos elementos entre las diferentes hojas de estilo empleadas (especialmente la que permite establecer el estilo general de la aplicación, tratando de seguir las especificaciones de Material Design, y la que se necesita emplear para hacer uso de NVD3).

Una alternativa habría sido independizar la parte ocupada por la gráfica (y que por tanto necesita hacer uso de la hoja de estilo de NVD3) en un `iframe`. Los problemas de compatibilidad de este mecanismo con ciertos navegadores, y las dificultades adicionales que podría suponer para la aplicación el mantener en correcta sincronía la información dentro fuera del `iframe` hizo que se descartara esta solución.

En su lugar se optó por utilizar una etiqueta `<style>` en la que importar la hoja de estilo de las gráficas sólo cuando sea necesaria, y reduciendo el ámbito al cual se aplica.

Alguna pequeña colisión restante se resolvió sobreescribiendo la propiedad objeto de la colisión donde fuera necesario.

6.7. Generación de las gráficas

Dado que el principal propósito de este proyecto es ofrecer una herramienta que facilite estadísticas acerca de los resultados obtenidos por los alumnos al rellenar exámenes corregidos con Eyegrade, la generación de las gráficas es uno de los puntos que más atención merece. Como se vio en el apartado 2.4.3, el sistema elegido para esta tarea se basa en la combinación de dos plataformas: D3 y NVD3.

El primero de ellos (D3) es el que se encarga de proporcionar un altísimo grado de potencia y versatilidad, permitiendo generar casi cualquier tipo de gráfica que podamos imaginar. Al ser un sistema muy potente y extendido, existe bibliografía que trata de suavizar la pronunciada curva de aprendizaje que presenta.

El segundo (NVD3) es el que hace posible reducir la cantidad de código necesaria para llegar a los principales tipos de gráfica, facilitando la reutilización de ese mismo código. Sin embargo tiene una pega importante, y es la ausencia casi total de documentación. Exceptuando unos ejemplos básicos de cómo implementar los principales tipos de gráfica, no existe apenas información sobre

cómo trabajar con las estructuras generadas.

Para cada gráfica es necesaria una función Javascript, que apoyándose en NVD3 (y por tanto, usando D3 por debajo) se encarga de tomar unos datos (extraídos de un JSON que puede cambiar a cada petición) y mostrar en un SVG la gráfica solicitada.

Uno de los principales problemas encontrados fue cómo poder trabajar con datos generados (y modificados) con cada petición HTTP, en lugar de estar programados en el propio código de la aplicación.

Intentando que el código sea lo más reutilizable posible, las líneas necesarias para generar cada tipo de gráfica se encuentra en un fichero .js, que se incluye en la página cada vez que es necesario. Para poder hacer este código independiente de la URL a través de la cual se obtienen los datos, se emplea un campo oculto en el código HTML de forma que a través de nuestro código Javascript sea posible recogerlo y hacer la llamada adecuada en casa ocasión.

El problema se agravó al tratar de incorporar varias gráficas en la misma página. Especialmente cuando la página requiriese la visualización de varias gráficas del mismo tipo, ya que cada una de ellas debe hacer uso de diferentes conjuntos de datos, y hacerlo de forma independiente.

La ausencia de documentación afectó especialmente en este punto, hasta que se dio con la idea de encapsular el código dentro de funciones, de forma que aunque se usara el mismo código (y por tanto los mismos nombres de variables) cada gráfica fuera independiente de la anterior. Además se emplea una sencilla navegación por el árbol DOM de la página para independizar cada par datos + código Javascript (ya que las llamadas a objetos del árbol DOM por su identificador no eran válidas para diferenciar las distintas gráficas).

Presentando el código que sirve para incorporar cada gráfica a la página:

```
<div id="chart">
<style scoped>
@import "/egstats/static/nvd3/build/nv.d3.css";
</style>

<svg id="chartsvg" style='height:600px; width:500px;'/>
<form><input type="hidden" id="data_url" value="/egstats/chartData{{mydata}}"/></form>
<script src="/egstats/static/js/{{chartjs}}" type="text/javascript"></script>
</div>
```

En él se declara un elemento `<div>` dentro del cual:

1. Se incorpora a la cascada CSS la hoja de estilo de NVD3.

2. De declara el elemento SVG en el que se visualizará la gráfica.

3. Se declara un formulario con un único elemento oculto que servirá para comunicar al código Javascript la URL a la que deberá solicitar los datos para la gráfica. `mydata` es una variable facilitada a la plantilla de forma dinámica, y que contiene la URL correspondiente en cada petición.

4. Se incluye el fichero .js que contiene el código para generar la gráfica. Como a priori no se sabe qué gráfica se deberá mostrar, se emplea de nuevo el lenguaje de plantillas de Django para incluir ese dato en el momento de recibir la petición.

De este modo se encapsula en un `div` toda la información necesaria para mostrar la gráfica. El código Javascript, en lugar de buscar el dato que necesita a través de un ID, navega por el DOM, partiendo del elemento que en ese momento se está ejecutando, y llegando a su elemento "hermano" dentro de ese elemento `div`.

Esto, sumado a la forma en la que se usa el código Javascript, permite:

- Reutilizar el máximo código posible.

- Establecer en tiempo de ejecución, tanto el tipo de gráfica como los datos que se deberán mostrar.

- Encapsular el código HTML necesario para una gráfica dentro de un único elemento `<div>`, y el código Javascript dentro de una función de forma que cada gráfica sea independiente y por tanto se puedan mostrar varias en la misma página.

Esto facilita la implementación de nuevas gráficas (incluso nuevos tipos de gráfica) y nuevas pantallas para la aplicación con el mínimo de cambios.

6.8. Velocidad de subida de nuevos archivos

Otro punto cuyos resultados no fueron completamente satisfactorios al principio fue la subida de archivos, ya que el tiempo necesario para hacerla efectiva era excesivo (unos 25 segundos para

subir un archivo de preguntas de examen normal, y unos 69 segundos para subir un archivo de resultados también normal).

Para diagnosticar el problema se hizo uso de la Django Debug Toolbar[1], herramienta que permite obtener métricas de todas las peticiones hechas al servidor Django, así como información acerca de las peticiones que éste hace a su vez al sistema gestor de bases de datos.

Gracias a esta herramienta se pudo ver que el problema radicaba en un excesivo número de peticiones hechas a la base de datos, (del orden de 425 en el caso del fichero de preguntas, y más de 1.700 en el caso del fichero de respuestas).

Cabe destacar que las llamadas a la base de datos no se programan explícitamente, sino que se delega en el ORM de Django, quien traduce las peticiones que se hacen a los objetos en consultas SQL. Esta abstracción tiene la ventaja de poder cambiar el *software* de bases de datos empleado sin necesidad de tocar una sola línea de código (aparte de la línea correspondiente en el fichero de configuración de Django). Sin embargo tiene el problema de que no se tiene un control preciso de las consultas que se realizan.

La eficiencia máxima se podría conseguir empleando funciones de Django que permiten componer las consultas explícitamente, pudiendo realizar las optimizaciones que se deseen a nivel de operaciones SQL. El problema es que al hacerlo se correría el riesgo de atarse a un *software* concreto.

Para tratar de evitarlo se decidió optimizar estas métricas usando otros métodos:

- Implementar un sencillo sistema de caché de datos en las operaciones de incorporación de nuevos datos (subida de ficheros). De esta forma se pueden hacer las comprobaciones previas sobre objetos ya cargados en memoria, en lugar de hacer las peticiones al SGBDD.
- Emplear funciones de carga masiva de datos que provee Django, en lugar de hacer las inserciones una a una.

Con estas medidas se consiguió reducir la subida de archivos de preguntas a 9 segundos y 140 peticiones, y los de respuestas a menos de un segundo y 35 peticiones.

[1] `https://github.com/django-debug-toolbar/django-debug-toolbar`

Capítulo 7

VALIDACIÓN Y PRUEBAS

Antes de dar por finalizado el proyecto es necesario confirmar que cumple con todos los requisitos. Lo primero que se hará será realizar las pruebas pertinentes siguiendo el listado de requisitos, tratando de contemplar todas las opciones posibles a cada una.

- ✓ Datos almacenados en base de datos única (RQ1)
- ✓ Subida de ficheros de preguntas .xml de Eyegrade correcta con ficheros reales (RQ3)
- ✓ Subida de ficheros de configuración .eye de Eyegrade correcta con ficheros reales (RQ4)
- ✓ Subida de ficheros de respuestas .csv de Eyegrade correcta con ficheros reales (RQ5)
- ✓ Generación de gráficas de diferentes tipos y contenidos (RQ6)
- ✓ Gráficas ordenables (RQ8)
- ✓ Porciones de las gráficas pinchables para acceder a la información correspondiente a dicha porción (RQ9)
- ✓ Acceso a la aplicación mediante el mismo login y clave de la Universidad (alumno). Se comprueba que tiene acceso a sus datos (exámenes, preguntas respondidas, preguntas acertadas, preguntas fallidas). (RQ10, RQ14, RQ15, RQ16)
- ✓ Acceso a la aplicación como alumno. Fracasa el intento de acceder a otras pantallas o gráficas copiando URLs obtenidas en sesiones iniciadas con cuenta de personal (RQ13)

- ✓ Acceso a la aplicación como alumno. Fracasa el intento de acceder a otras pantallas o gráficas modificando el identificador de alumno en la URL tratando de acceder a los datos de otro alumno (RQ13).
- ✓ Acceso a la aplicación mediante el mismo login y clave de la Universidad como profesor (RQ10)
- ✓ Acceso a la aplicación con cuenta de personal, se comprueba que tiene acceso a todas las gráficas implementadas mostrando los datos correspondientes a los exámenes que tenga autorizados (RQ17)
- ✓ Se comprueba que al introducir una clave que no corresponde a la correcta en el servidor LDAP se rechaza el acceso (RQ10)
- ✓ Se comprueba que al introducir un nombre de usuario no existente en LDAP se rechaza el acceso (RQ10).
- ✓ Se crea un usuario local dentro de la aplicación con perfil de alumno (RQ11)
- ✓ Se accede al sistema con el usuario local correctamente (pese a no existir en LDAP) (RQ11).
- ✓ Fracasa el intento de acceder a cualquiera de las pantallas sin haberse autenticado (nuevo navegador, ventana de incógnito o de navegación privada). Pide login y clave (RQ24)
- ✓ Fracasa el intento de acceder a cualquiera de las pantallas después de haber cerrado la sesión. Pide login y clave (RQ24).
- ✓ Se accede correctamente a las gráficas de preguntas redactadas por cada profesor (RQ18).
- ✓ Se accede correctamente a las gráficas de respuestas a cada pregunta (RQ19).
- ✓ Se accede correctamente a las gráficas de respuestas a cada pregunta, filtrando por profesor (RQ19).
- ✓ Se accede correctamente a las gráficas de respuestas acertadas y fallidas por cada alumno. Al pinchar en cada barra se ven las preguntas en cuestión (RQ20).
- ✓ Se borra un examen completo, confirmando que desaparecen todas las preguntas, opciones y respuestas (RQ21).

- ✓ Se intenta acceder a la pantalla de borrado de examen desde un usuario que no fue el que lo subió a la plataforma: no se puede (RQ21).
- ✓ Se añade un profesor como autorizado a ver un examen, funciona correctamente (RQ22).
- ✓ Se elimina a un profesor como autorizado de un examen. Funciona correctamente (RQ22).
- ✓ Se intenta acceder a la pantalla para añadir/eliminar autorizados de un examen que no ha subido el usuario: fracasa (RQ22).
- ✓ Estando autenticado en la aplicación, se hace una llamada al servicio REST que devuelve los datos para las gráficas: se recibe un JSON con la información (RQ24).
- ✓ Sin estar autenticado en la aplicación se hace una llamada al servicio REST: se deniega el acceso (RQ23).
- ✓ Se comprueba que es posible filtrar y ver la gráfica de respuestas por opción y pregunta, filtrando para que aparezcan únicamente las escritas por un profesor: funciona (RQ26)
- ✓ Se visualiza la gráfica de respuestas por opción y pregunta, filtrando de forma que sólo se vean las de un examen concreto: funciona (RQ27).
- ✓ El acceso se hace en todo momento desde un navegador (RQ28)

Adicionalmente se solicita a una persona sin conocimientos técnicos que acceda a la aplicación y la use. Esta prueba, aunque general e poco específica, a menudo ayuda a localizar problemas que pasarían desapercibidos a ojos de alguien que ya está acostumbrado a usar la aplicación, y que por tanto va directo a la acción deseada, omitiendo alternativas que podrían provocar errores. Los comentarios que esta persona nos hace son de gran utilidad también a la hora de valorar la usabilidad del interfaz *web*.

Capítulo 8

CONCLUSIONES Y TRABAJOS FUTUROS

Tras finalizar el proyecto se han alcanzado algunas conclusiones, y también han surgido ideas que si bien quedan fuera del ámbito de esta aplicación, podrían suponer el germen de futuros proyectos que podrían complementar a Eyegrade y Egstats.

8.1. Conclusiones

Las principales conclusiones a las que se ha llegado durante la realización del proyecto, y especialmente tras su finalización son:

- Un buen diseño del modelo de datos, y de la forma de acceso a los mismos es un punto crítico para facilitar el posterior desarrollo de la aplicación, ya que cualquier necesidad no prevista inicialmente podría forzar a una modificación en algo tan crítico como el modelo de datos, que obligaría a realizar otros tantos cambios en cadena.

- A la hora de diseñar una aplicación de estadísticas hay que tener en cuenta que no todas las estadísticas o gráficas tienen la misma relevancia, por lo que no basta con desplegar todas las alternativas sin ningún tipo de análisis previo.

- Enlazando con lo anterior, resulta de vital importancia contar con la opinión de quien genera los datos, y quien posteriormente hará uso de las estadísticas que se generen, para alcanzar la perspectiva que permita seleccionar las gráficas más útiles.

- A la hora de valorar un sistema de generación de gráficas es importante ponderar no sólo el atractivo visual de las mismas, o su impacto, sino también las posibilidades que ofrece de personalización e interacción.

- La elección del patrón MVC como modelo a seguir resultó ser una opción acertada, y la implementación que hace Django de dicho modelo se mostró como una alternativa ágil y sólida, facilitando el desarrollo así como la acometida de las necesarias modificaciones para cumplir con los requisitos de la aplicación.

Con todo ello, finalmente se alcanzaron los objetivos previstos, obteniendo como resultado una aplicación que se espera complemente a Eyegrade y facilite y fomente su uso entre nuevos usuarios.

8.2. Trabajos futuros

Partiendo de este proyecto hay varias líneas de trabajo posibles que podrían complementar la plataforma Eyegrade + Egstats, como por ejemplo:

8.2.1. Generador de informes periódicos

Una posibilidad sería la implementación de un sistema que generase de forma automatizada informes periódicos a partir de la información recopilada en Egstats. Dependiendo de las posibilidades o necesidades podría trabajar a dos niveles:

- Solicitando a la interfaz REST los datos necesarios para generar sus propias gráficas, mostrándolos además en forma de tablas o de texto descriptivo.

- Aprovechando el sistema de generación de gráficas completo (datos JSON + NVD3 + D3)

De ser necesario también se podría implementar otro método REST para devolver los datos en un formato específico para rellenar estos informes, que además podrían ser a varios niveles:

- Para cada uno de los docentes partícipes de la plataforma, enviando un correo electrónico cuatrimestral con los resultados obtenidos por los alumnos en sus preguntas.

- Para cada coordinador de asignatura, con los datos correspondientes a cada materia.

- Un informe global, a un nivel de abstracción mayor, para responsables de Departamento.

8.2.2. Interfaz REST para obtención de datos anonimizados

Una de las principales virtudes de este proyecto es la simplicidad con la que ahora es posible generar las gráficas. Esto se podría aprovechar para incorporar algunas de estas gráficas en diferentes páginas *web* (de Departamento, de Grupos de Trabajo de Docentes, etc).

Incrustar de este modo las gráficas en lugar de añadir una simple imagen tiene la ventaja de que los datos que se muestran corresponden al momento actual, y no a aquel en que se tomó la imagen. Además podría conservar parte de la capacidad de interacción que ofrecen las gráficas de Egstats.

Para hacer viable esta exportación de gráficas a otros sistemas, posiblemente no autenticados, sería necesario anonimizar los datos de forma que no se vulnerase la LOPD[1]. Por tanto la mejor forma de hacerlo sería implementar un interfaz REST paralelo que tuviera algunas diferencias con el actual:

- Ya que la petición no vendría de Egstats, en lugar de autenticar a través del sistema implementado en Django sería necesario emplear algún otro sistema, como por ejemplo OAuth2 para obtener acceso temporal al recurso y recibir los datos sin necesidad de hacer el proceso de login interactivo.
- El nuevo interfaz debería ofrecer un abanico de posibilidades diferentes (no todas las gráficas ofrecidas en Egstats tienen sentido o interés si se despersonalizan).
- Todos los datos deberían enviarse anonimizados, de forma que no sea posible vincular la información a una persona física.

8.2.3. Generador de exámenes de práctica

Si se estimase útil, otra posibilidad consistiría en un sistema capaz de gestionar exámenes de práctica, basados en preguntas de exámenes anteriores, y quizás incluso utilizando el porcentaje de aciertos/errores a cada pregunta para decidir la frecuencia con que éstas deberían aparecer en dichos exámenes de "prueba".

Ya que la información está almacenada en Egstats, no sería complicado formar exámenes pseudo-aleatorios y permitir que los alumnos los realizaran, recibiendo la autocorrección en tiempo real (puesto que el modelo de datos también contiene la respuesta correcta).

[1] `http://www.boe.es/buscar/act.php?id=BOE-A-1999-23750`

Algo que debería incluir este nuevo sistema sería la posibilidad de permitir el acceso únicamente a los alumnos que en ese momento estuvieran cursando la asignatura.

APÉNDICES

APÉNDICE A

PRESUPUESTO DEL PROYECTO

Un apartado importante en todo proyecto es la correcta estimación de costes y elaboración del presupuesto, ya que de ello depende en buena parte la viabilidad final de la idea.

Para el cálculo del presupuesto tendremos en cuenta las siguientes consideraciones:

- Se considerará un único tipo de perfil, que hará las funciones de arquitecto, analista y programador.
- Teniendo en cuenta el perfil múltiple seleccionado, se estima en 40 euros la hora trabajada.
- Se estiman en 570 las horas de trabajo necesarias, algo menos de 72 días laborables en jornadas de 8 horas.
- Todo el *software* empleado, desde el sistema operativo hasta los programas servidores, pasando por el entorno de desarrollo a emplear, será gratuito, por lo que el coste de licencias será cero.
- La amortización de equipamiento informático se calcula tomando como base 3 años de amortización de un equipo de sobremesa, dos monitores, teclado y ratón, para un total de 1.291 euros.

Tipo	Concepto	Cantidad	Total
Costes de Personal	Salario empleado	17.100,00	
	Seguros sociales	5.700,00	
Total costes de personal		**22.800,00**	
Costes materiales	Amortización equipamiento informático	118,12	
	Licencia sistema operativo	0,00	
	Licencia entorno de desarrollo	0,00	
	Licencia programa servidor	0,00	
	Licencia entorno de desarrollo	0,00	
Total costes materiales		**118,12**	
Total costes indirectos		**4.583,62**	
Subtotal			**27.501,74**
IVA			**5.775,37**
TOTAL			**33.277,11**

Tabla A.1: Presupuesto

APÉNDICE B

Manual de usuario de Egstats

Egstats es una aplicación de generación de estadísticas basadas en los resultados obtenidos por los estudiantes en exámenes de respuesta múltiple corregidos con Eyegrade.

Se pueden distinguir dos perfiles de usuario, con diferente nivel de privilegios:

- Alumno (nombre de usuario estrictamente numérico)
- Profesor (nombre de usuario alfanumérico)

Existen 6 grandes grupos de acciones que se pueden hacer dentro de la aplicación, tres de ellas accesibles a ambos perfiles:

- Inicio de sesión.
- Cierre de sesión.
- Consulta de datos o estadísticas.

Y otros tres tipos de acciones que únicamente estarán a disposición de los usuarios que dispongan de perfil de docente:

- Subir un nuevo examen.
- Gestionar (añadir/eliminar) profesores autorizados a visualizar un examen.
- Borrar un examen subido.

En los apartados siguientes se expondrá la forma de trabajar con cada uno de ellos.

B.1. Inicio de sesión

Para poder empezar a operar con la aplicación es necesario iniciar sesión. Para ello hay que acceder a la URL en que se haya instalado, que tendrá la forma

http[s]://<servidor>[:puerto]/egstats.

Al hacerlo aparecerá la pantalla de inicio de sesión, como se ve en la figura B.1

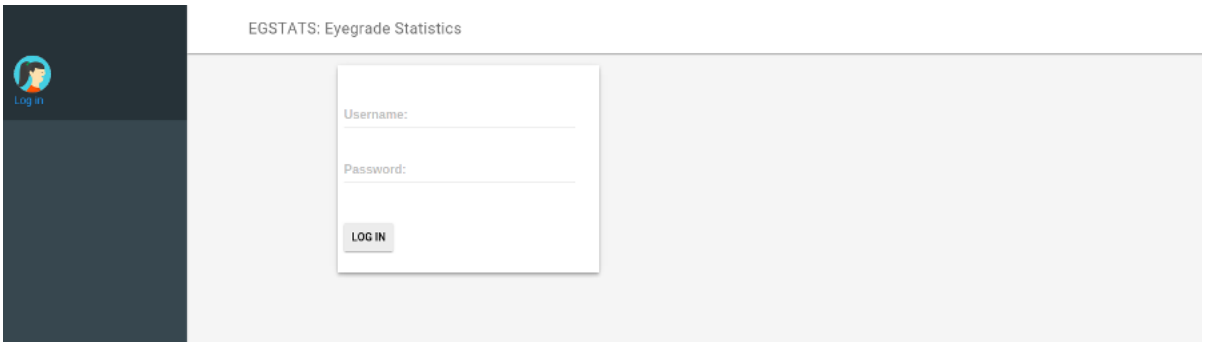

Figura B.1: Inicio de sesión

Para poder acceder se deberá emplear un nombre de usuario y clave válidos en el servicio LDAP que se emplee. En el caso de la UC3M deberán corresponder a un alumno o personal docente, investigador o administrativo existente.

Antes de iniciar sesión no habrá acceso a ninguna de las vistas, ni al servicio REST de datos.

B.2. Cierre de sesión

Es importante, especialmente cuando se acceda desde un equipo compartido, cerrar la sesión una vez se termine con las tareas que se deseen hacer.

Para ello basta con pinchar en el triángulo que habrá a la derecha del login (parte superior izquierda de la pantalla) y después en "*logout*", como se puede ver en la figura B.2.

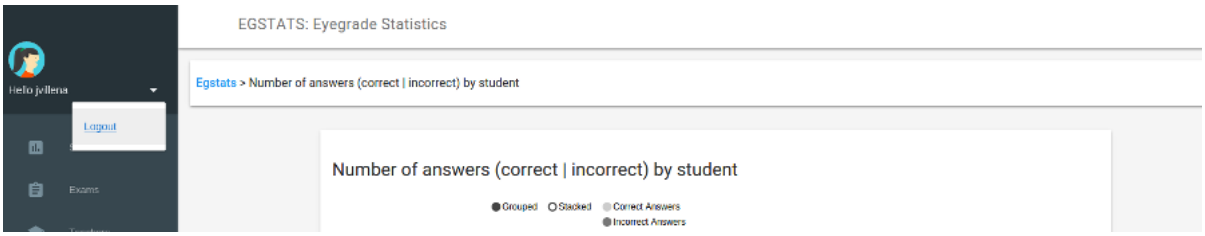

Figura B.2: Cierre de sesión

Una vez hecho esto, si se desea visualizar cualquier estadística o gestionar algún examen, será necesario iniciar nuevamente la sesión.

B.3. Consulta de datos o estadísticas

Habiendo iniciado la sesión se podrán visualizar a los datos a los que tenga acceso el perfil que corresponda al usuario autenticado.

Existen notables diferencias entre los datos a los que puede acceder un alumno, y aquellos a los que puede acceder un usuario con perfil de personal, por lo que se tratará en subapartados independientes.

B.3.1. Consultas con perfil de alumno

Los alumnos únicamente podrán acceder a sus propios resultados en los exámenes que se hayan subido a la aplicación, por lo que el menú de la izquierda únicamente albergará el acceso directo a sus estadísticas personales, como se puede ver en la imagen B.3.1.

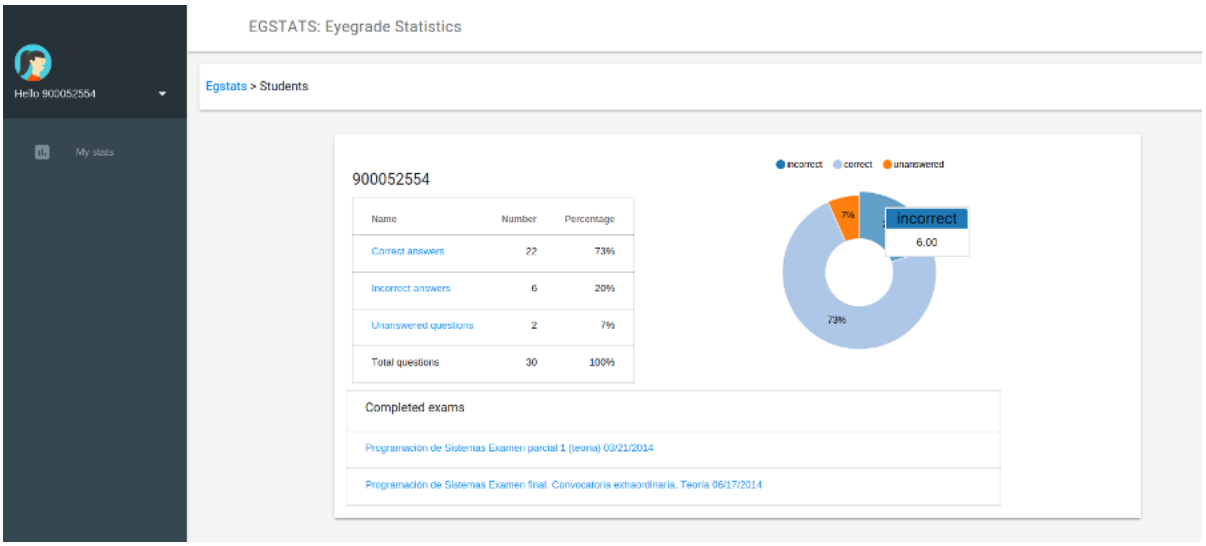

Figura B.3: Pantalla inicial alumno

Lo primero que verá será la gráfica con sus estadísticas globales de preguntas acertadas, fallidas y sin contestar. Los datos numéricos concretos se pueden visualizar en la tabla de estadísticas, o bien pasando el ratón sobre la porción deseada de la gráfica. Pinchando en la misma porción o en el enlace de la tabla se mostrarán las preguntas correspondientes a la opción elegida (acertadas/fallidas/sin contestar).

Abajo se pueden ver los exámenes realizados por el alumno (y que hayan sido subidos a la aplicación). Pinchando en ellos se podrá visualizar el texto del examen, apareciendo destacadas las opciones seleccionadas por el alumno (marcadas en rojo cuando son errores), así como las correctas (en verde).

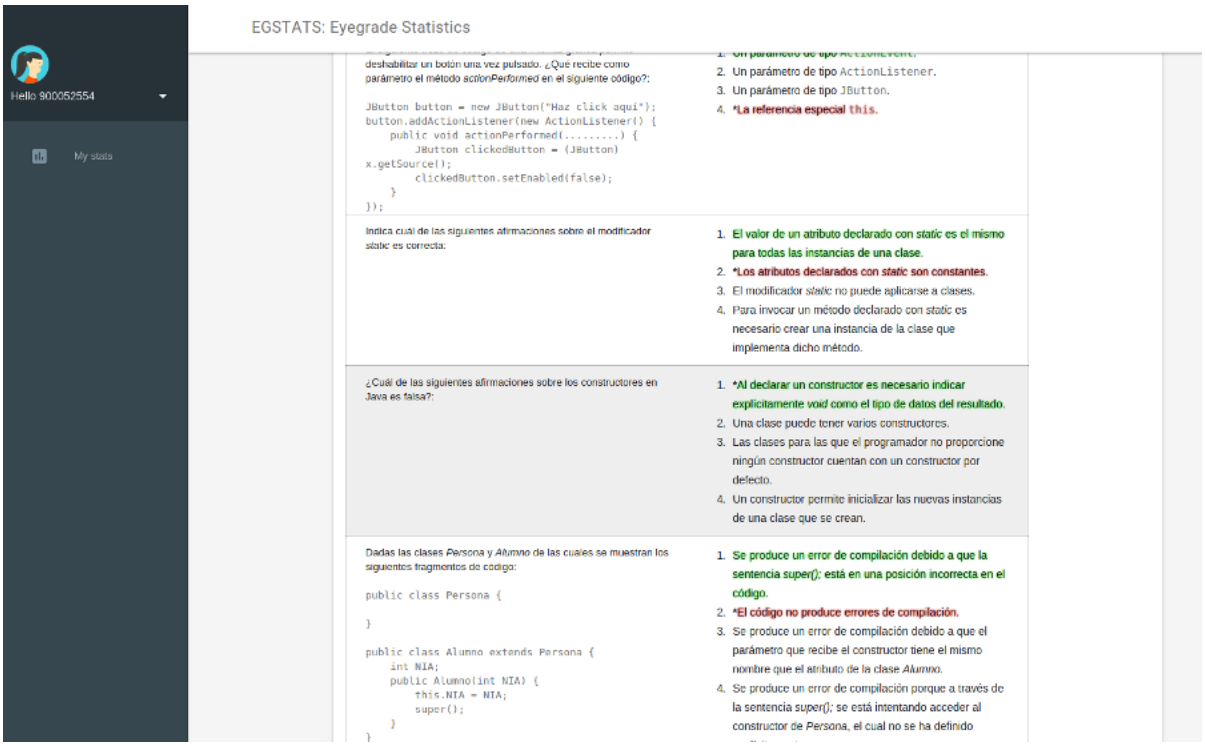

Figura B.4: Examen corregido

La vista de preguntas acertadas, fallidas o sin respuesta es análoga, con la salvedad de que se muestra únicamente el tipo de respuesta elegido, pero de todos los exámenes disponibles.

B.3.2. Consultas con perfil de personal

Los profesores tienen acceso a una funcionalidad mucho mayor, representada por un menú izquierdo mucho más amplio, como se puede ver en la figura B.3.2.

Cada profesor podrá ver los datos correspondientes a los exámenes que él mismo haya subido, así como a aquellos a los que se le haya concedido acceso explícitamente (mediante la opción de gestionar examen explicada en el apartado B.5).

La primera pantalla que se encontrará será la siguiente:

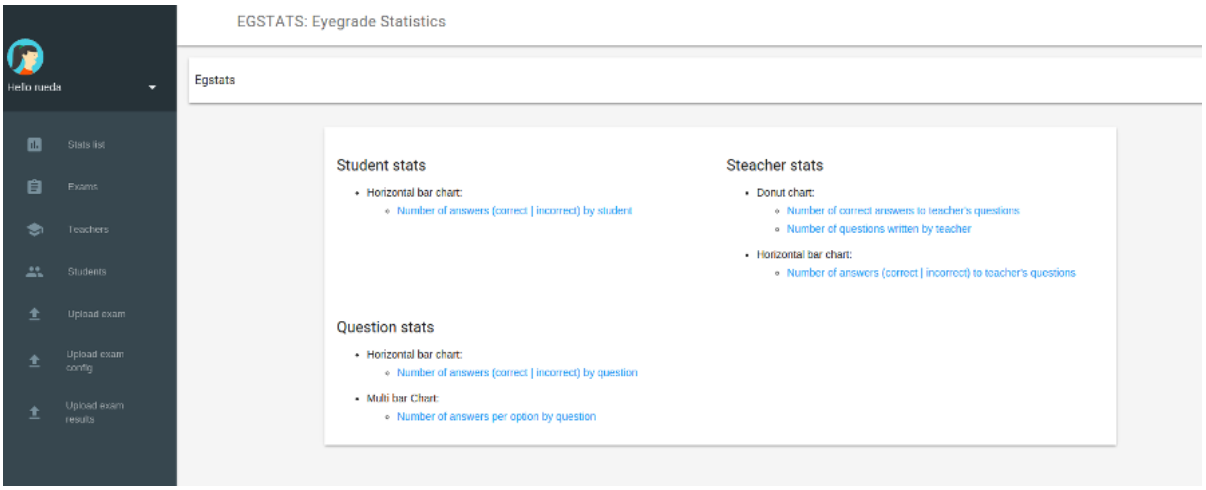

Figura B.5: Pantalla inicial personal

Desde el menú tendrá acceso a las principales funcionalidades de la aplicación, mientras que pinchando en los enlaces del área central podrá visualizar los principales tipos de estadísticas disponibles.

En cualquier momento se podrá volver a esta pantalla pinchando en "*Stats list*" en el menú, o bien en "*Egstats*" en la barra de navegación.

También se podrá hacer uso de la barra de navegación superior para acceder a pantallas anteriores, así como a alguna relacionada con la actual.

En los apartados siguientes se describen algunas de las principales pantallas de estadísticas, así como la forma de interactuar con ellas.

Aciertos y errores por alumno

Pinchando en "*Number of answers correct|incorrect by student*" se llegará a esta pantalla, que representa el número de respuestas acertadas y fallidas por cada estudiante.

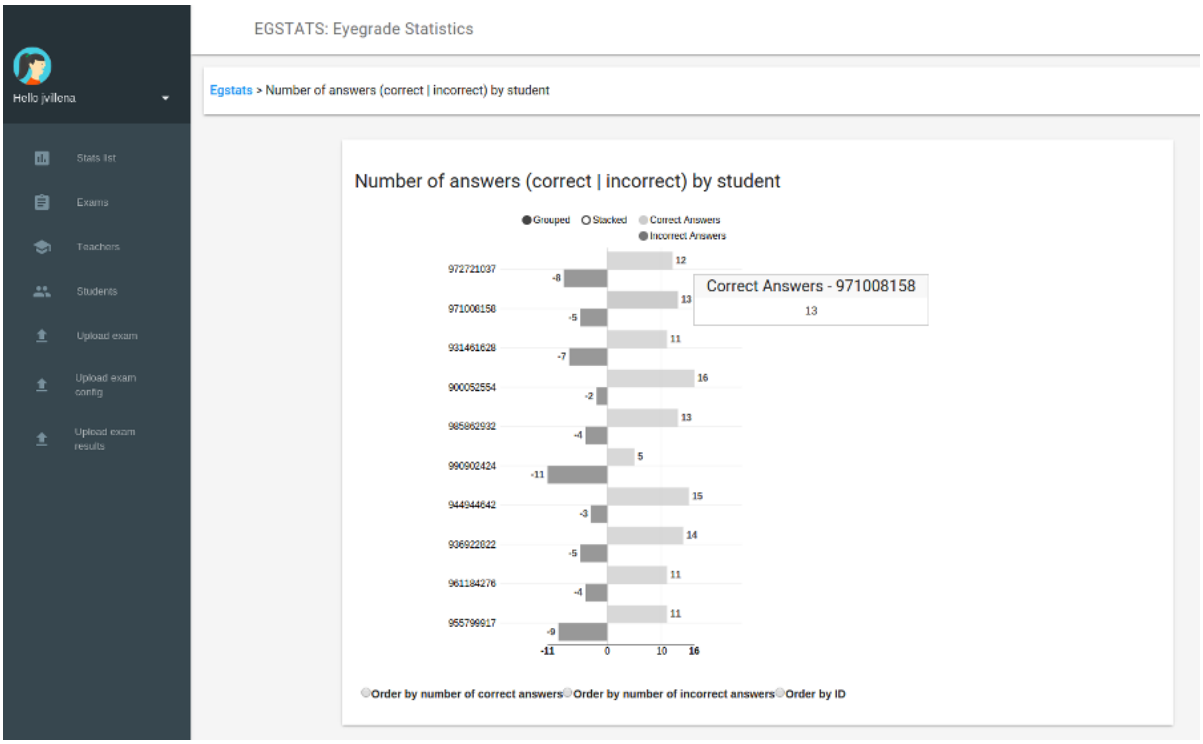

Figura B.6: Número de aciertos y errores por alumno

Pasando el ratón sobre cada barra se visualizarán los datos numéricos asociados, y si se pincha en ella se accederá al texto de las opciones acertadas/erróneas marcadas por el alumno en cuestión (apareciendo destacadas las opciones correctas, y los errores cometidos por el estudiante, donde así hubiera sido).

La gráfica se puede ordenar por tres posibles criterios:

- Por número de respuestas acertadas.
- Por número de respuestas fallidas.
- Por identificador de alumno.

Además se puede:

- Hacer que las dos barras de cada alumno aparezcan enfrentadas (en lugar de contrapeadas).
- Ocultar las barras correspondientes a respuestas acertadas/fallidas.

Ordenando por número de respuestas acertadas, y pinchando en la barra de respuestas equivocadas de los mejor situados se podría ver en qué fallaron los que mejor calificación sacaron.

Aciertos y errores por pregunta

Accesible a través del enlace "*Number of answers correct|incorrect by question*", esta pantalla representará mediante barras horizontales el número de respuestas correctas e incorrectas marcadas para cada pregunta de los exámenes a los que el usuario tenga acceso.

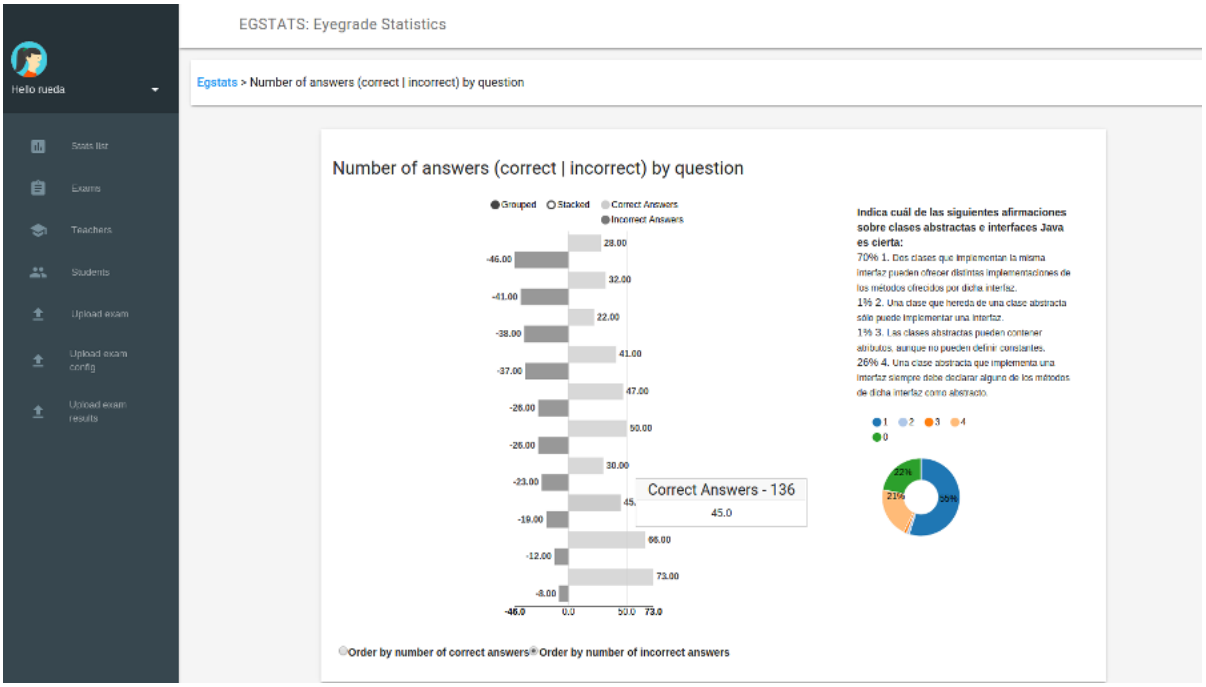

Figura B.7: Número de aciertos y errores por pregunta

Al pasar el ratón sobre cada barra aparecerá el dato numérico, y al pinchar sobre ella aparecerá a la derecha el texto de la pregunta, las diferentes opciones, y la gráfica con los porcentajes correspondientes al número de veces que cada opción fue marcada.

Esta gráfica se puede ordenar por dos criterios:

- Número de respuestas correctas.
- Número de respuestas incorrectas.

Al igual que la gráfica de aciertos y errores por alumno, permite ocultar las barras correspondientes a aciertos o errores, o bien situarlas enfrentadas, en lugar de contrapeadas.

Número de respuestas por opción y pregunta

Pinchando en "*Number of answers per option by question*" se mostrará una gráfica de barras apiladas en la que se representa el número de veces que se ha marcado cada una de las opciones de cada pregunta, así como el número de veces que se ha dejado en blanco.

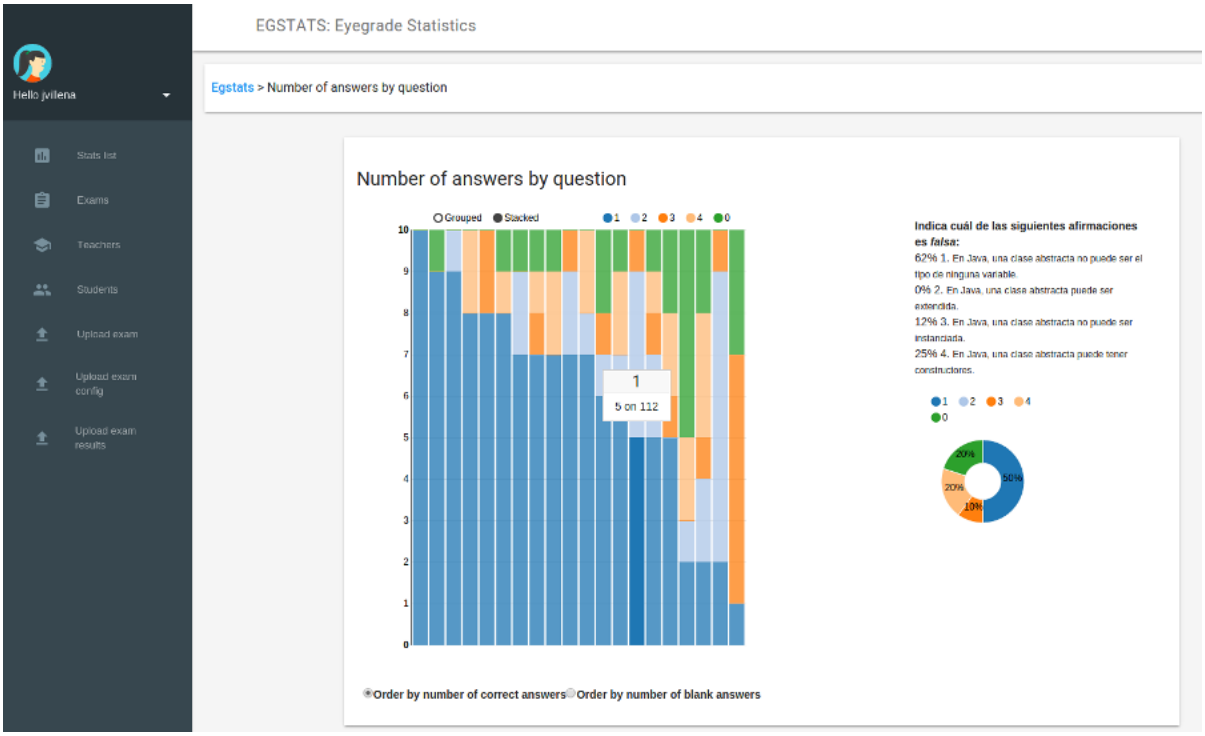

Figura B.8: Número de respuestas por opción y pregunta

Al pasar el ratón sobre cada barra, aparecerá el número de veces que fue seleccionada esa opción (o el número de veces que se dejó en blanco, si se pasa el ratón por el trozo de barra correspondiente a esa alternativa).

Si además se pincha en la barra, a la derecha aparecerá el texto de la pregunta con sus posibles respuestas, y una gráfica con el porcentaje de veces que se optó por cada uno de los valores.

La gráfica se puede ordenar según dos criterios:

- Por número de respuestas correctas.
- Por número de veces que la pregunta se dejó en blanco.

Empleando cualquiera de los dos criterios y seleccionando las preguntas que queden en las

posiciones extremas, se puede ver fácilmente qué cuestiones son respondidas correctamente con más frecuencia, cuáles dan lugar a más equívocos y cuáles plantean más dudas (hasta el punto de dejarla en blanco).

Número de preguntas por profesor

Al pinchar en el enlace "*Number of questions written by teacher*" se llegará a la pantalla que muestra la gráfica que representa el número de preguntas escritas por cada profesor (de aquellos a los que el usuario tenga acceso).

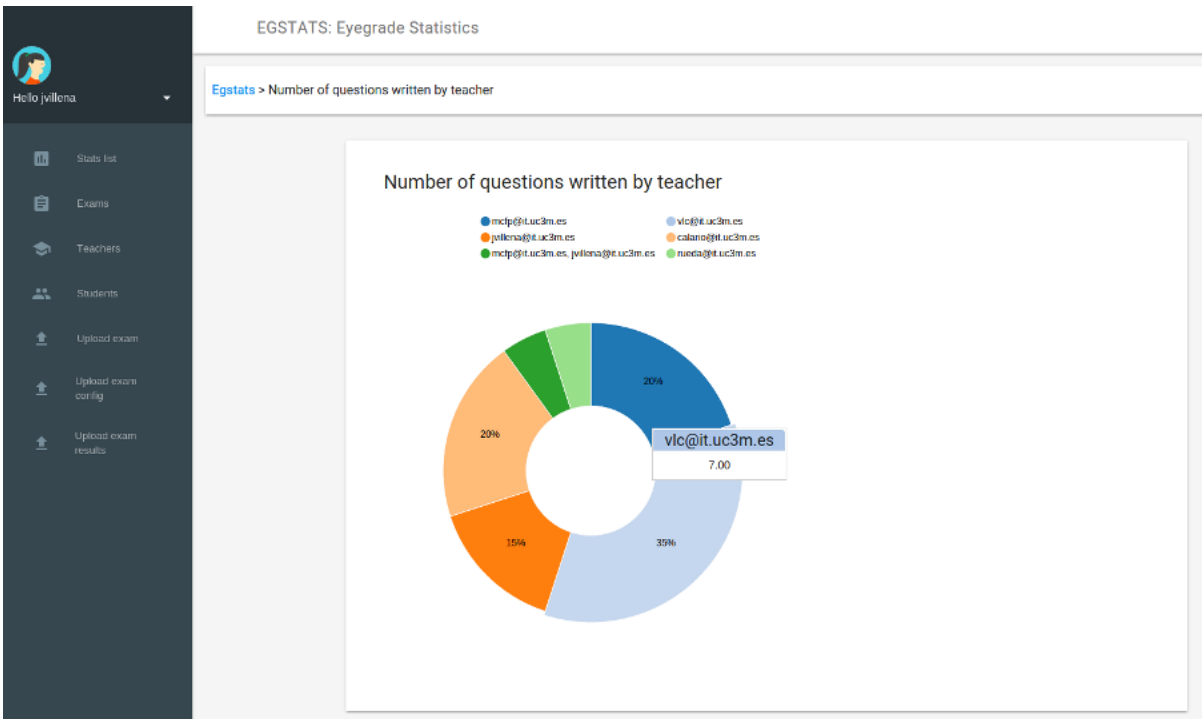

Figura B.9: Número de preguntas por profesor

Al pinchar en cada porción se podrá ver el texto de las preguntas, junto con las posibles respuestas, destacando en verde la correcta.

En la gráfica se pueden ocultar los segmentos, y los porcentajes se actualizan.

Exámenes

Pinchando en el menú de la izquierda sobre la opción "*Exams*" se accederá a la pantalla que permitirá elegir uno de los exámenes a los que el usuario tenga acceso.

Al hacerlo se llegará a la vista resumen del examen:

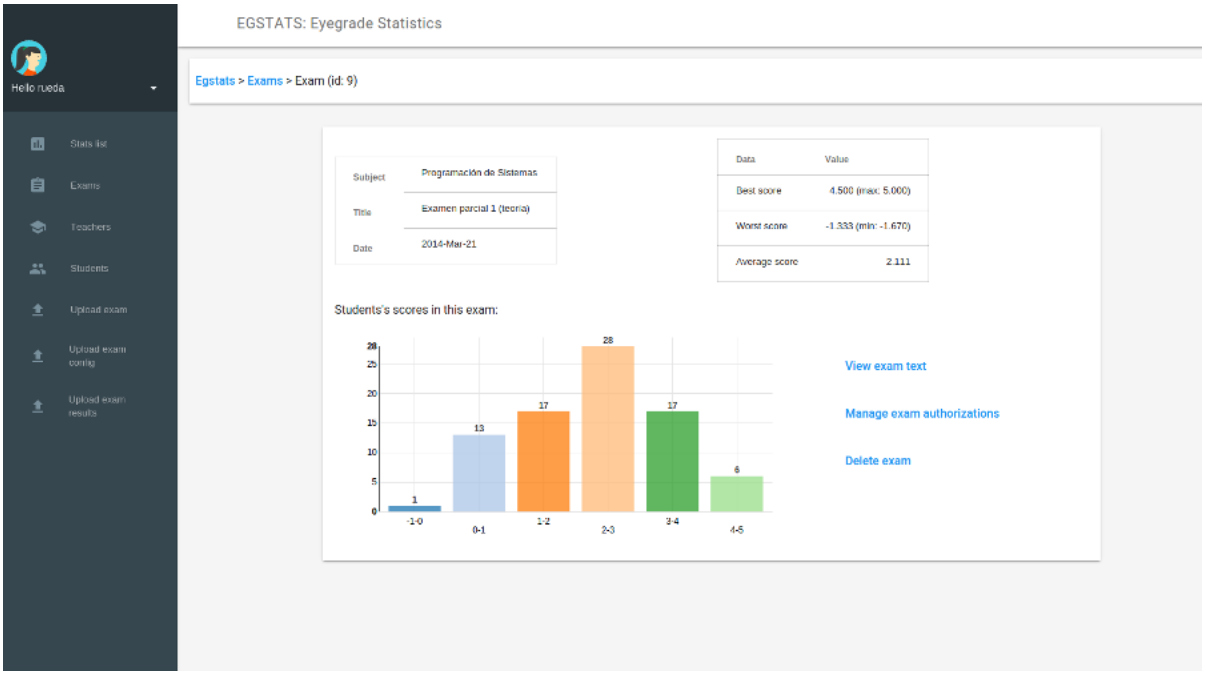

Figura B.10: Pantalla resumen del examen

Donde destaca la gráfica con la distribución de notas (número de alumnos que han obtenido cada rango de notas). Además se pueden observar los datos identificativos del examen (asignatura, título del examen, fecha), la mejor nota obtenida, la menor y la media.

Por último aparecen varios enlaces que permitirán:

Ver el texto del examen Presenta las preguntas, junto a las opciones posibles, destacando en verde la correcta.

Gestionar las autorizaciones para el examen Permite añadir o eliminar profesores autorizados a visualizar los datos relacionados con este examen. Esta opción únicamente le aparece al docente que ha subido este examen a la aplicación.

Borrar examen Permite borrar el examen y todos los datos asociados de la aplicación. Esta

opción sólo aparece cuando el usuario es el mismo que subió el examen.

Profesores

Al seleccionar "*Teachers*" en el menú de la izquierda, se podrá elegir a un profesor, llegando a la pantalla que muestra las estadísticas básicas sobre el mismo.

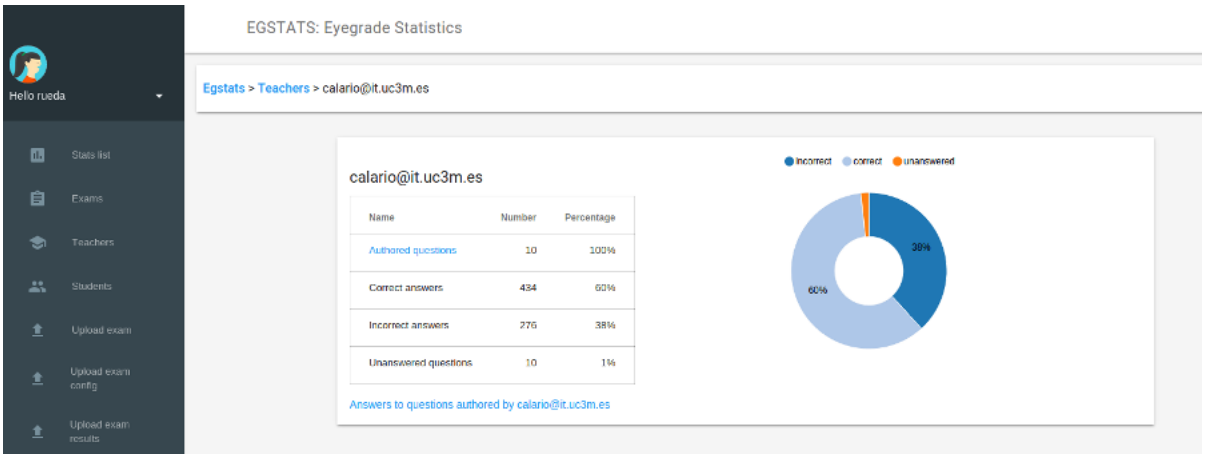

Figura B.11: Pantalla resumen del profesor

Destaca la gráfica que representa los porcentajes de aciertos, errores y respuestas en blanco a las preguntas redactadas por el profesor.

Junto a ella, una tabla presenta los datos numéricos, así como un enlace a la página en la que se muestran todas las preguntas redactadas, y debajo, un enlace a la gráfica de número de respuestas por opción y pregunta (como la vista en el apartado B.3.2, pero limitada a las preguntas redactadas por el docente cuyos datos se están viendo).

Estudiantes

Pinchando en "*Students*" en el menú de la izquierda se podrá elegir un estudiante en concreto para ver sus estadísticas y respuestas, del mismo modo que lo visto en el apartado B.3.1.

Únicamente serán accesibles los datos de alumnos a cuyos exámenes se tenga acceso.

B.4. Subir un examen (sólo perfil de profesor)

El proceso de subir un nuevo examen a la aplicación consta de tres pasos consecutivos, uno para cada uno de los tres tipos de archivo en que basa Eyegrade su funcionamiento.

En los siguientes apartados se verán en orden.

B.4.1. Subir las preguntas del examen (sólo perfil de profesor)

Pinchando en "*Upload exam*" se podrá elegir el fichero que contenga los datos de preguntas y respuestas del examen que se desee subir.

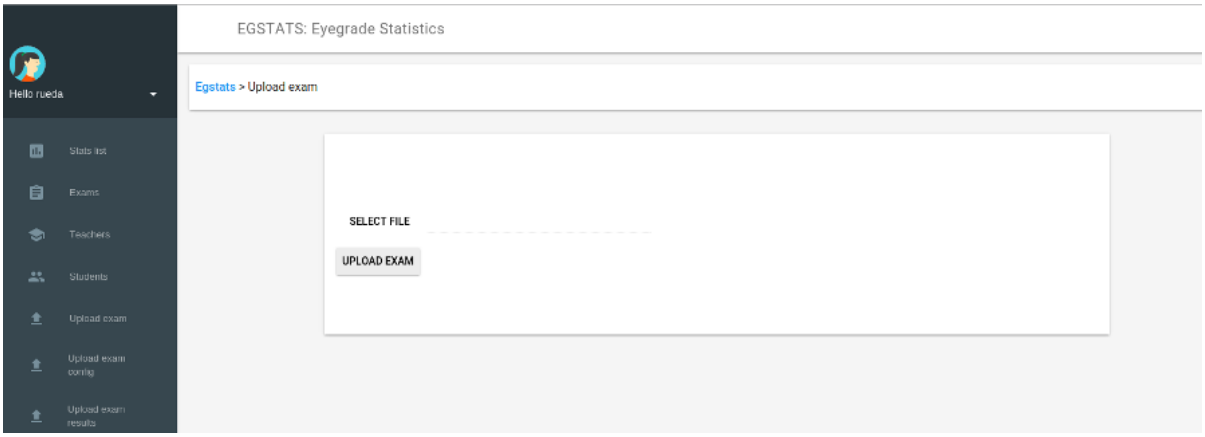

Figura B.12: Subir las preguntas de un examen

El fichero deberá tener extensión .xml, ya que de lo contrario no se permitirá continuar.

Una vez subido, la aplicación comprueba que no se haya subido con anterioridad (basándose en asignatura, título y fecha del examen). En caso de ser un examen repetido, no incorporará los datos, para evitar duplicidades.

B.4.2. Subir la configuración del examen (sólo perfil de profesor)

Al seleccionar "*Upload exam config*" se podrá elegir el fichero que contenga los datos de configuración del examen. También habrá que elegir a qué examen corresponden en el desplegable inferior, en el que sólo aparecen aquellos cuya configuración no ha sido aún añadida.

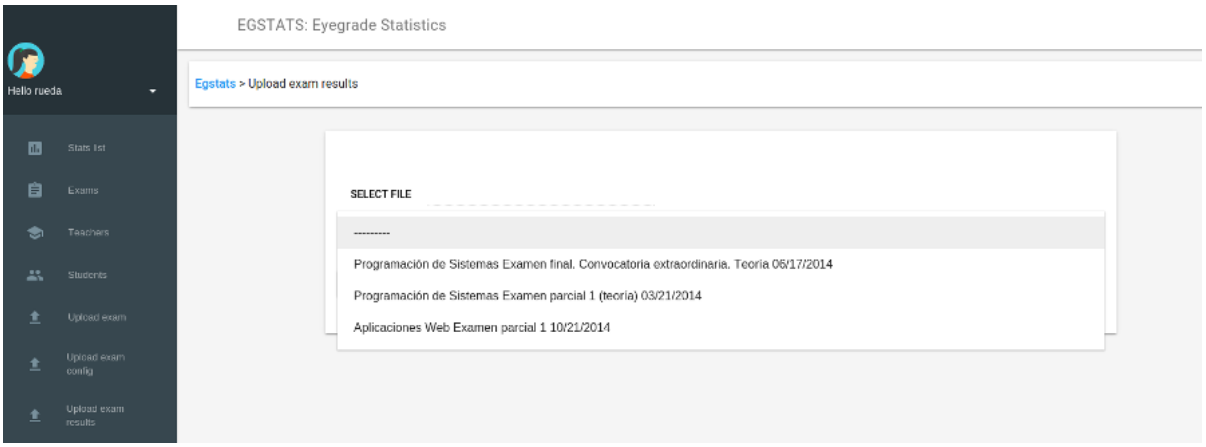

Figura B.13: Subir la configuración de un examen

Si el fichero no tiene extensión .eye, no se permitirá continuar.

B.4.3. Subir respuestas de un examen (sólo perfil de profesor)

Tras seleccionar "*Upload exam results*" se podrá elegir el fichero que contenga los datos de las respuestas de los alumnos al examen. También habrá que elegir a qué examen corresponden en el desplegable inferior, en el que sólo aparecen aquellos cuya configuración se haya establecido previamente.

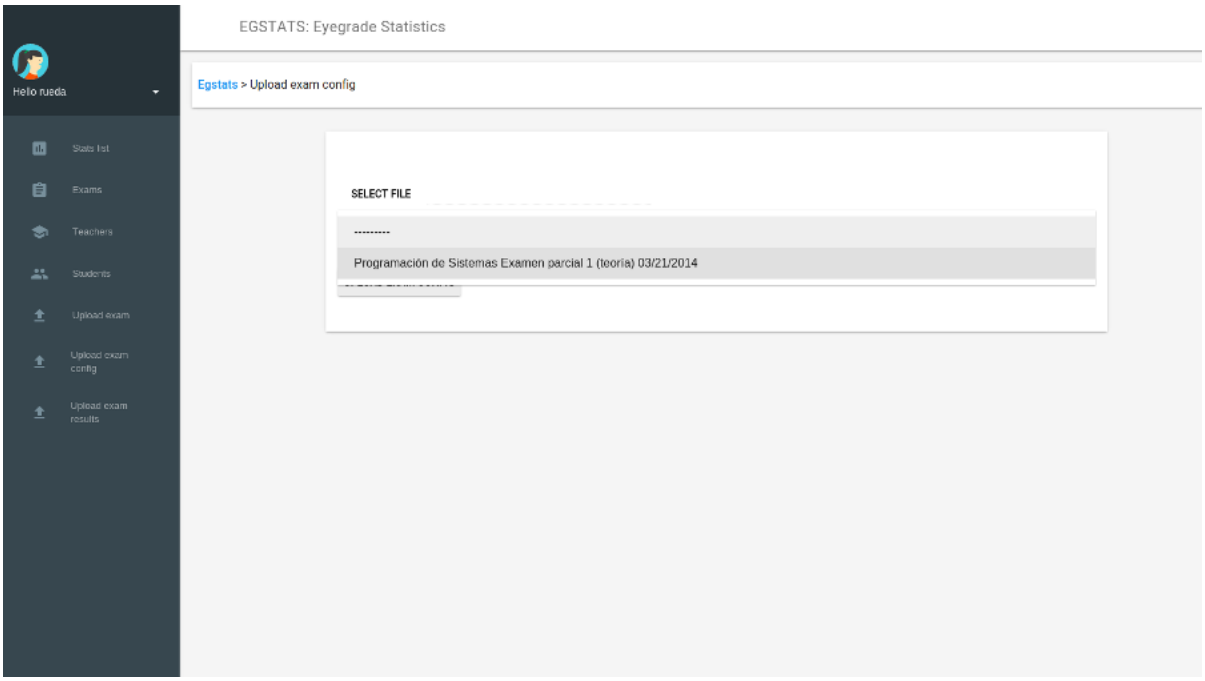

Figura B.14: Subir respuestas de un examen

Sólamente se podrán subir ficheros con extensión .csv.

Al terminar este tercer paso, la aplicación dispondrá de todos los datos necesarios para mostrar estadísticas sobre este examen.

B.5. Gestionar exámenes (sólo perfil de profesor)

Desde la vista resumen de examen vista en el apartado B.3.2, el docente que lo subió encontrará un enlace con el texto "*Manage exam authorizations*". Pinchando en él podrá gestionar qué profesores podrán acceder a los datos de este examen en Egstats.

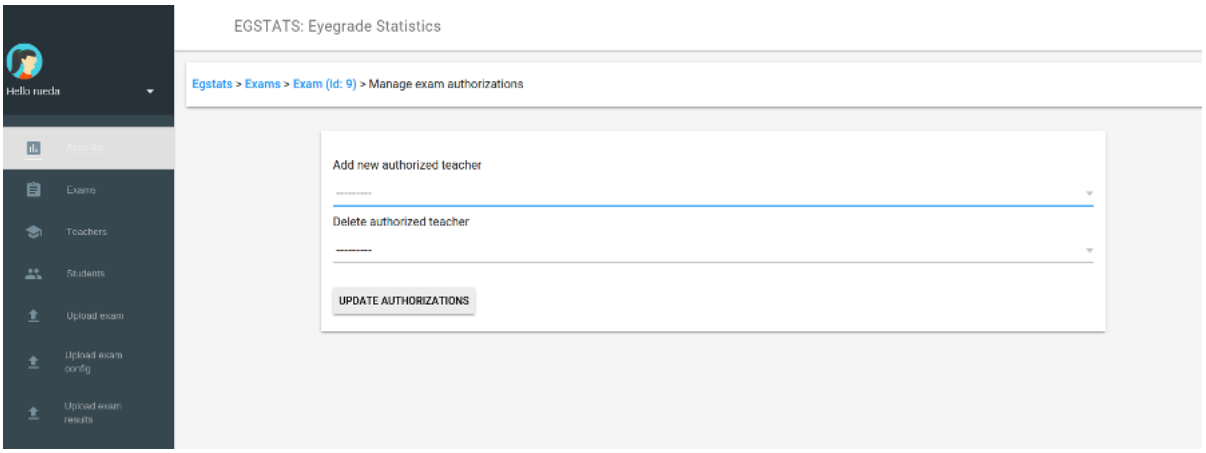

Figura B.15: Gestionar autorizados de un examen

En el desplegable habilitado para añadir autorizaciones estarán disponibles los profesores usuarios de Egstats que no estén entre los autorizados a visualizar este examen.

En el desplegable dedicado al borrado de autorizados, aparecerán únicamente aquellos profesores que ahora tengan acceso.

B.6. Borrar exámenes (sólo perfil de profesor)

También desde la vista resumen de examen vista en el apartado B.3.2, el profesor que lo añadió a Egstats podrá ver un enlace con el texto "*Delete exam*". Este enlace le permitirá borrar el examen completo, incluyendo preguntas, respuestas, opciones marcadas por los alumnos y las vinculaciones que cualquiera de estos datos tuviera con otros elementos de Egstats.

Al pinchar, aparecerá una pantalla de confirmación, cuyo fin es evitar borrados accidentales.

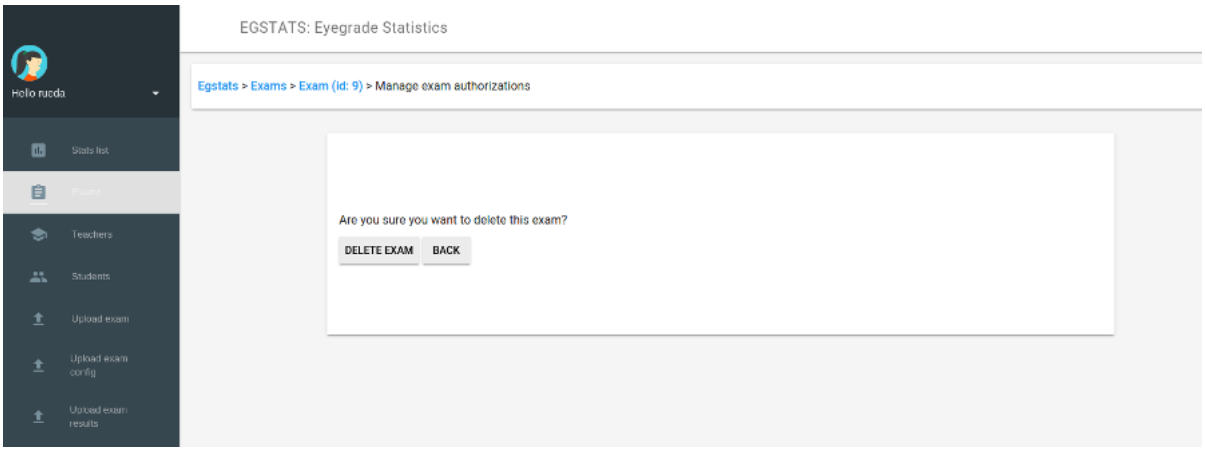

Figura B.16: Borrar examen

Bibliografía

[1] Beehive. *The Book of Zope.* No Starch Press, 2001.

[2] H. Bergsten. *JavaServer Pages, 3rd Edition.* O'Reilly Media, 2003.

[3] H. Bergsten. *JavaServer Faces.* O'Reilly Media, 2004.

[4] E. S. Boese. *An Introduction to Programming with Java Applets, 3rd edition.* Jones & Bartlett Learning, 2009.

[5] T. Christiansen, B. d Foy, L. Wall, and J. Orwant. *Programming Perl, 4th Edition.* O'Reilly Media, 2012.

[6] K. Diment and M. Trout. *The Definitive Guide to Catalyst Writing Extensible, Scalable and Maintainable Perl-Based Web Applications.* Apress, 2009.

[7] J. A. Fisteus, A. Pardo, and N. F. García. Grading multiple choice exams with low-cost and portable computer-vision techniques. *Journal of Science Education and Technology*, 22:560–571, 2013.

[8] D. Flanagan and Y. Matsumoto. *The Ruby Programming Language.* O'Reilly Media, 2008.

[9] A. Freeman. *Pro ASP.NET MVC 5.* Adam Freeman, 2013.

[10] D. Golding. *Beginning CakePHP: From Novice to Professional.* Apress, 2008.

[11] D. Greenfeld and A. Roy. *Two Scoops of Django: Best Practices For Django 1.8.* Two Scoops Press, 2015.

[12] M. Grinberg. *Flask Web Development: Developing Web Applications with Python.* O'Reilly Media, 2014.

[13] A. Holovaty. *The Definitive Guide to Django: Web Development Done Right.* Apress, 2009.

[14] A. Lewis. *Rails Crash Course.* No Starch Press, 20014.

[15] M. Lutz. *Learning Python, 5th Edition.* O'Reilly Media, 2013.

[16] C. McDonough. *The Pyramid Web Application Development Framework.* Agendaless Consulting, 2011.

[17] S. Murray. *Interactive Data Visualization for the Web.* O'Reilly Media, 2013.

[18] D. Newton. *Apache Struts 2 Web Application Development.* Packt Publishing, 2009.

[19] P. Niemeyer and D. Leuck. *Learning Java.* O'Reilly Media, 2013.

[20] K. Tatroe, P. MacIntyre, and R. Lerdorf. *Programming PHP, Third Edition.* O'Reilly Media, 2013.

[21] S. Teller. *Data Visualization with D3.js.* Packt Publishing, 2013.

[22] C. Walls. *Spring in Action (3rd Edition).* Manning Publications, 20011.

[23] C. Wilson. *RaphaelJS Graphics and Visualization on the Web.* O'Reilly Media, 2013.

[24] F. Zaninotto. *The Definitive Guide to symfony.* Apress, 2007.