



**ESCUELA POLITÉCNICA SUPERIOR
UNIVERSIDAD CARLOS III DE MADRID**

**INGENIERÍA DE TELECOMUNICACIÓN
SISTEMAS Y REDES DE TELECOMUNICACIONES**

PROYECTO FINAL DE CARRERA

**DISEÑO Y DESARROLLO DE UNA HERRAMIENTA
PARA LA GESTIÓN Y PUBLICACIÓN
ELECTRÓNICA DE CALIFICACIONES DE LA
EVALUACIÓN CONTINUA**

**Autor: Sherezade Fraile Paniagua
Tutora: Iria Estévez Ayres**

Julio de 2014

Resumen

En el Proyecto Fin de Carrera titulado *Diseño y desarrollo de una herramienta para la gestión y publicación electrónica de calificaciones de la evaluación continua* se describe el proceso de análisis, diseño, desarrollo y evaluación de una aplicación para facilitar la introducción de datos (en este caso notas), por parte de los profesores, dada la introducción del plan Bolonia y su disparidad en la casuística.

Con este propósito se ha implementado una arquitectura Cliente-Servidor a tres niveles compuesta por diferentes tecnologías entre las que se pueden destacar: Apache-Tomcat como Servidor Web, AJAX como técnica de desarrollo web, JSON como lenguaje de comunicación y MySQL como sistema de gestión de base de datos.

La característica principal, que diferencia a nuestro sistema de otros con el mismo cometido, reside en el hecho de que nuestra aplicación, además de permitir realizar las mismas acciones que estos, incorpora nuevas opciones derivadas de posibilitar al servidor la recepción de archivos de distintos clientes que se analizarán para ser insertados en la base de datos.

Las funciones principales que se llevan a cabo en este sistema son la conexión del dispositivo al servidor web, el almacenamiento de información en el servidor web y la visualización de dicha información en un navegador web.

Agradecimientos

Este proyecto no habría sido posible sin la inestimable colaboración de mis padres. Gracias.

No puedo obviar en este epígrafe a mi tutora Iria, por haber aportado la idea, sugerencias, opiniones, comentarios y sobre todo por su paciencia.

Agradezco igualmente su apoyo, aguante y dedicación a todos los compañeros de la carrera.

Por supuesto agradecer y brindar este proyecto a mis amigos por aguantarme e intentar entenderme (vosotros sabéis quienes sois).

Por último dedicar el proyecto a todos los que de alguna manera lo han apoyado y me han ayudado a conseguirlo: profesores, compañeros, camareros. Todos sois en parte responsables del éxito o el fracaso de esta difícil empresa.

Por todo lo que me habéis dado a lo largo de estos años, muchas gracias. Sin vosotros no lo hubiera logrado.

Contenidos

Resumen	III
Acrónimos	XIII
1. Introducción	1
1.1. Contexto	1
1.2. Objetivos	1
1.3. Contenido	2
2. Estado del arte	3
2.1. Servidor	3
2.1.1. Sistema Gestor de Base de Datos	4
2.1.1.1. MySQL	4
2.1.1.2. PostgreSQL	5
2.1.1.3. Oracle	6
2.1.1.4. SQL Server	7
2.1.1.5. DB2	7
2.1.1.6. Comparación	8
2.1.1.7. Herramienta visual de diseño de bases de datos	9
2.1.2. Software Intermedio o Middleware	10
2.1.2.1. Servidor de Aplicaciones	10
2.1.2.2. Servidor Web	10
2.2. Cliente	12
2.2.1. HTML/XHTML	12
2.2.2. JavaScript	13
2.2.3. Flash y Shockwave	13
2.2.4. AJAX	13
2.2.5. Comparación	14
2.3. Comunicación Cliente Servidor	14
2.3.1. XML	15
2.3.2. JSON	15

2.3.3.	YAML	15
2.3.4.	Comparación	15
2.4.	Conclusiones	16
3.	Selección de las tecnología a usar en el proyecto	17
3.1.	Sistema 1: LAMP (Linux, Apache, MySQL y PHP)	17
3.1.1.	Diseño Arquitectura	17
3.1.2.	Instalación	18
3.1.3.	Creación del proyecto	19
3.1.3.1.	Modelo de datos	20
3.1.3.2.	Servidor	20
3.1.3.3.	Cliente	21
3.1.3.4.	Comunicación	22
3.1.4.	Sistema Completo	23
3.1.5.	Esquema Completo	25
3.1.6.	Mejoras	26
3.2.	Sistema 2: LAMP (Linux, Apache, Middleware (JSP) y PostgreSQL)	27
3.2.1.	Diseño Arquitectura	27
3.2.2.	Instalación	28
3.2.3.	Creación del proyecto	29
3.2.3.1.	Modelo de datos	29
3.2.3.2.	Servidor	29
3.2.3.3.	Cliente	31
3.2.3.4.	Comunicación	31
3.2.4.	Sistema Completo	32
3.2.5.	Esquema Completo	32
3.2.6.	Mejoras	32
3.3.	Comparación	33
4.	Descripción del proyecto	35
4.1.	Casos de usos	35
4.2.	Requisitos	36
4.2.1.	Acciones Permitidas	36
4.2.2.	Acciones NO Permitidas	37
4.3.	Diseño	38
4.4.	Resumen	39
5.	Implementación	41
5.1.	Tecnologías	41
5.2.	Modelo de datos	41
5.3.	Cliente	44
5.3.1.	Sistema de acceso	45
5.3.2.	Sistema de recuperación de contraseñas	47
5.3.3.	Sistema de inserción de profesores	48
5.3.4.	Cambio de contraseña	49
5.3.5.	Funcionalidades Profesor	50
5.3.5.1.	Desconectar	51

5.3.5.2.	Settings	51
5.3.5.3.	Incorporar nuevo grupo de alumnos	51
5.3.5.4.	Gestionar mi grupos de alumnos	52
5.3.5.5.	Borrar Grupo/Asignatura	54
5.3.6.	Funcionalidades alumno	54
5.4.	Servidor	56
5.4.1.	Sistema de acceso	57
5.4.2.	Sistema de recuperación de contraseñas	58
5.4.3.	Sistema de inserción de profesores	59
5.4.4.	Cambio de contraseña	60
5.4.5.	Funcionalidades Profesor	61
5.4.5.1.	Sistema verificación	62
5.4.5.2.	Settings	63
5.4.5.3.	Incorporar nuevo grupo de alumnos	63
5.4.5.4.	Gestionar mis grupos de alumnos	69
5.4.5.5.	Borrar Grupo/Asignatura	83
5.4.6.	Funcionalidades alumno	84
5.4.6.1.	Sistema verificación	84
5.4.6.2.	Sistema alumno	84
5.5.	Archivos	85
6.	Validación	91
6.1.	Resultados obtenidos	91
6.1.1.	Profesor	94
6.1.2.	Cliente	101
6.2.	Conclusiones	102
7.	Conclusiones y Trabajo futuro	105
7.1.	Conclusiones	105
7.2.	Trabajo futuro	105
	Bibliografía	107
A.	Presupuesto del proyecto	109

Indice de tablas

2.1. Comparación SGBDR 1	8
2.2. Comparación SGBDR 2	9
2.3. Herramientas para los SGBDR	10
2.4. Comparación de Servidores Web	11
2.5. Servidores Web & SGBDR	12
2.6. Comparacion Tecnologías Cliente	14
2.7. Comparación Lenguajes Comunicación	15
3.1. Comparacion Sistemas Estudio Práctico	33
5.1. Proceso para la comprobación del acceso en el Servidor	58
5.2. Proceso para el envío de la contraseña en el Servidor	58
5.3. Proceso para la inserción de profesores en el Servidor	60
5.4. Proceso para el cambio de contraseña en el Servidor	61
5.5. Proceso para la verificación en el Servidor	62
5.6. Proceso para la obtención de las asignaturas asociadas a un profesor en el Servidor	64
5.7. Proceso para la creación de una asignatura en el Servidor	64
5.8. Proceso para la obtención de los grupos de una asignatura en el Servidor	66
5.9. Proceso para la creación de un grupo de alumnos en el Servidor	67
5.10. Proceso para la obtención de las notas de un alumno en el Servidor	71
5.11. Proceso para la modificar la nota de un examen sin apartados en el Servidor	72
5.12. Proceso para la introducir un examen en el Servidor	75
5.13. Proceso para obtener la nota de un examen en el Servidor	77
5.14. Proceso para la obtención de los distintos parametros de la evaluación en el Servidor	81
5.15. Proceso para la personalización de la evaluación en el Servidor	82
5.16. Proceso para descargar una copia de seguridad del Servidor	83
5.17. Proceso para la borrar un o varios grupos en el Servidor	84
A.1. Fases del Proyecto	109
A.2. Costes de material	109

A.3. Presupuesto 110

Lista de Figuras

2.1. Arquitectura a 3 niveles	3
2.2. Tecnologías AJAX	14
3.1. Esquema Arquitectura Sistema1	18
3.2. Estructura Tabla SGBDR MySQL	20
3.3. Doctrine	20
3.4. Página Inicio Estudio	23
3.5. Página Buscar Estudio	24
3.6. Página Add Estudio	25
3.7. Esquema Arquitectura completa Sistema1	26
3.8. Esquema Arquitectura Sistema2	27
3.9. Conexion mediante JDBC	30
3.10. Esquema Arquitectura completa Sistema2	32
4.1. Subdivisión Base de datos	38
4.2. Lectura información	38
4.3. Inserción información	39
4.4. Modificar información	39
4.5. Cliente-Servidor	40
5.1. Esquema Modelo de datos	42
5.2. Sistema acceso	46
5.3. Flujo métodos página acceso	47
5.4. Flujo métodos página guardar	49
5.5. Flujo métodos página cambiarPassword	50
5.6. Funciones profesor	50
5.7. Funciones alumno	54
5.8. Flujo métodos página menú	56
5.9. Flujo <i>servlet</i> comprobación acceso del Servidor	57
5.10. Flujo <i>servlet</i> guardarArchivo(profesores) del Servidor	59
5.11. Flujo <i>servlet</i> cambiarContraseña del Servidor	61

5.12. Flujo <i>servlet</i> comprobarTipo del Servidor	62
5.13. Flujo <i>servlet</i> crearAsignatura del Servidor	65
5.14. Flujo <i>servlet</i> guardarArchivoGrupo del Servidor	68
5.15. Flujo <i>servlet</i> notasAlum	70
5.16. Flujo <i>servlet</i> Cambiar Nota Examen sin apartados del Servidor	73
5.17. Flujo <i>servlet</i> Evaluación del Servidor	80
5.18. Archivo Profesores	86
5.19. Archivo Grupo Alumno	86
5.20. Archivo Planificación Asignatura	87
5.21. Archivo Notas examen sin apartados	88
5.22. Archivo Notas examen con apartados	88
5.23. Archivo fórmula evaluación simple personalizada asignatura	89
5.24. Archivo fórmula evaluación condicionada personaliza asignatura	89
5.25. Archivo download	89
6.1. Página Acceso	92
6.2. Página recuperacion	92
6.3. Página guardar	93
6.4. Página cambiarPassword	93
6.5. Pagina inicio	94
6.6. Vista Settings	94
6.7. Vista Incorporar nuevo grupo de Alumnos	95
6.8. Vista Crear una nueva asignatura	95
6.9. Visualización grupos existentes de una asignatura	96
6.10. Vista Crear nuevo grupo	96
6.11. Visualización gestionar grupos	97
6.12. Visualización alumnos del grupo	97
6.13. Visualización notas alumno	98
6.14. Vista Evaluación	99
6.15. Vista Planificación	99
6.16. Visualización notas examen	100
6.17. Vista Eliminar Grupo	101
6.18. Página menu - Listado de asignaturas	101
6.19. Página menu - Información asignatura	102

Acrónimos

A | B | C | D | F | H | I | J | M | O | P | R | S | T | X | Y

A

AJAX

(Asynchronous JavaScript And XML) JavaScript Asíncrono y XML. 12

API

(Application Programming Interface) Interfaz de Programación de Aplicaciones. 4, 6

ASP

Active Server Pages. 11

B

BD

Base de Datos. 9

C

CGI

(Common Gateway Interface) Interfaz de Entrada Común. 11

CPU

(Central Processing Unit) Unidad Central de Procesamiento. 7

CSS

(Cascading Style Sheets) Hojas de Estilo en Cascada. 13

D

DOM

(Document Object Mode) Modelo en Objetos para la Representación de Documentos. 13

DTD

(Document Type Definition) Definición de Tipo de Documento. 15

F**FTP**

(File Transfer Protocol) Protocolo de Transferencia de Archivos. 3, 11

H**HTML**

(HyperText Markup Language) Lenguaje de Marcas de Hipertexto. 10, 12, 13

HTTP

(Hypertext Transfer Protocol) Protocolo de Transferencia de Hipertexto. 10

I**IIS**

Internet Information Server. 11

J**J2EE**

Java 2 Enterprise Edition. 10, 11

JDBC

Java DataBase Conector. 29

JDK

(Java Development Kit) Conjunto de Desarrollo de Java. 33

JSON

(JavaScript Object Notation) Notación de Objetos de JavaScript. 13

JSP

JavaServer Pages. 11

M**MVC**

Modelo Vista Controlador. 18

O**ORM**

(Object-Relational Mapping) Mapeo Objeto-Relacional. 20

P**PHP**

Hypertext Preprocessor. 4

R**RAM**

(Random Access Memory) Memoria de Acceso Aleatorio. 5

S**SGBD**

Sistema de Gestión de Bases de Datos. 4

SGBDR

Sistema de Gestión de Bases de Datos Relacional. 4, 9

SGML

(Standard Generalized Markup Language) Estándar de Lenguaje de Marcado Generalizado. 12

SQL

(Structured Query Language) Lenguaje de Consulta Estructurado. 4

T**TOAD**

(Tool for Oracle Application Developers) Herramienta de Oracle para Desarrolladores de Aplicaciones. 9

TORA

(Toolkit for Oracle) Conjunto de Herramientas para Oracle. 10

X**XHTML**

(eXtensible Hypertext Markup Language) Lenguaje de Etiquetado Hipertextual Extensible. 12

XML

(eXtensible Markup Language) Lenguaje de Marcas Extensible. 10, 12

XPATH

(XML Path Language) Definición de Tipo de Documento. 15

XSL

(eXtensible Stylesheet Language) Lenguaje Extensible de Estilos. 13

Y**YAML**

Yet Another Multicolumn Layout. 15

Este capítulo introductorio iniciará al lector en los objetivos que se pretenden conseguir con este proyecto y dará una descripción de la estructura de este documento, aportándose una breve descripción de cada capítulo.

1.1. Contexto

Con la llegada del plan Bolonia, la evaluación de las distintas asignaturas, sobre las que un profesor puede impartir docencia, se ha vuelto más compleja. En las aplicaciones que se emplean actualmente, aunque se ha tenido en cuenta este hecho, su diseño no cubre de un modo eficiente estas nuevas necesidades.

Esta aplicación se ha diseñado para proporcionar alternativas que solventen dicha problemática.

1.2. Objetivos

Por lo anteriormente comentado, en este sistema una de las premisas que se persiguen es la de permitir la gestión de las docencia de las asignaturas de una manera más cómoda y fácil, dotando para ello al sistema de una mayor flexibilidad que la existente en los sistemas actuales.

Aunque, de manera resumida, la lista de objetivos de este proyecto es:

- Realizar un estudio teórico de las posibles tecnologías a emplear en nuestro proyecto, que junto con uno práctico, nos permita elegir la que mejor se adapte para implementar la solución del problema en cuestión.
- Análisis, diseño y desarrollo de las funcionalidades del sistema.
- Realización de las pruebas necesarias para comprobar la funcionalidad del sistema.
- Efectuar dicha aplicación de manera que pueda ser entendida y mejorada por terceras partes.

- Llevar a cabo el sistema de modo que resulte económico, accesible desde cualquier sitio y portable a cualquier plataforma (lograremos esto ya que se trata de una aplicación accesible vía web).

NOTA: En nuestro caso no será necesaria una fase de mantenimiento puesto que con la defensa del trabajo termina nuestra relación con el mismo.

Puesto que se trabajará con una arquitectura Cliente-Servidor, todo ello será estudiado desde dos perspectivas:

- La parte del Cliente.
- La parte del Servidor.

1.3. Contenido

Para entender mejor las circunstancias, implementación, estructura, complejidad y posibilidades de nuestra aplicación, este documento se divide en varios capítulos.

El capítulo de *Introducción* nos acerca las motivaciones que han dado lugar a la realización del presente proyecto, así como los objetivos que se han marcado al comienzo del mismo.

Por su parte, el *Estado del arte*, revisa las distintas tecnologías que pueden utilizarse en el desarrollo del sistema a implementar.

Posteriormente, en el tercer capítulo, esta información será reforzada con un estudio práctico que nos permita seleccionar las tecnologías más adecuadas para la realización el proyecto.

En el cuarto capítulo, se detallan las labores de diseño, concretamente, requisitos y funcionalidad del sistema.

En el capítulo *Implementación* nos adentraremos en el desarrollo del sistema, donde se explica cómo se han logrado realizar las funcionalidades que el programa requiere.

Una vez efectuado el desarrollo del sistema, comprobaremos que el mismo funciona adecuadamente y realiza las labores que se han descrito en los diferentes requisitos, revisando cada función de la aplicación en el capítulo *Validación*. Así evaluaremos los resultados del proyecto, y, además, estableceremos el posible camino por recorrer de nuestra aplicación.

En el apéndice A, *Presupuesto del proyecto*, se hace un análisis de los costes relacionados con las distintas fases, del desarrollo en función del tiempo y de los recursos invertidos en cada una de ellas.

En este capítulo haremos un estudio comparativo de las distintas tecnologías que se pueden utilizar para el desarrollo de una arquitectura Cliente-Servidor.

Concretamente, realizaremos el estudio de las tecnologías que componen la arquitectura Cliente-Servidor organizada como se muestra en la figura 2.1.

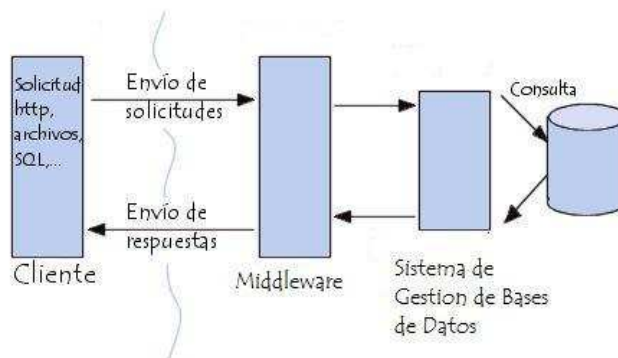


Figura 2.1: Arquitectura a 3 niveles

Se trata de un modelo de aplicación distribuida (donde se descentralizan los procesos y los recursos) en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Un cliente realiza peticiones a otro programa, el servidor, que le da respuesta.

2.1. Servidor

Existen diferentes tipos de servidores compatibles con esta arquitectura, tales como Servidores Web, Servidores FTP, entre otros. En nuestro caso, para una mayor flexibilidad, seguridad y rendimiento, usaremos aquel que esta compuesto por una arquitectura de tres niveles, tal y como se mostraba en la figura 2.1.

En ella puede apreciarse que será necesaria la utilización de un **Servidor de Aplicaciones/Web** (también denominado Software Intermedio o *Middleware*), cuya tarea es proporcionar los recursos solicitados, y el **Sistema Gestor de Base de Datos**, que proporciona al servidor de aplicaciones los datos que requiere.

2.1.1. Sistema Gestor de Base de Datos

El propósito general de los Sistemas de Gestión de Bases de Datos (SGBD) es el de manejar de manera clara, sencilla y ordenada un conjunto de datos que posteriormente se convertirán en información relevante para una organización. Estos proveen facilidades para la manipulación de grandes volúmenes de datos, además de abstracción de la información, independencia, consistencia, seguridad e integridad de los datos.

Aunque se pueden encontrar distintos tipos de SGBD, según el modelo de datos en el que esté basado, nosotros nos centraremos en el relacional (SGBDR). En él, se representa la base de datos como una colección de tablas, donde todos los datos tienen relación entre ellos. Este tipo suele utilizar SQL (Structured Query Language) como lenguaje de consultas de alto nivel, frente a las bases de datos No-SQL o no relacionales.

Existen gran variedad de SGBDR, pero en este documento vamos a analizar los más usados, entre los cuales se encuentran MySQL, PostgreSQL, Microsoft SQL server, Oracle y DB2.

2.1.1.1. MySQL

Se trata de Sistema de Gestión de Bases de datos relacional (SGBDR), que por lo tanto utiliza SQL, multihilo (que permite soportar una gran carga de forma muy eficiente) y multiusuario [MyS].

Este tipo puede ejecutar desde acciones tan básicas como insertar y borrar registros, actualizar información o hacer consultas simples, hasta realizar tareas tan complejas como la aplicación requiera.

Características

- Aprovecha la potencia de sistemas multiprocesador (multiproceso), gracias a su implementación multihilo.
- Soporta gran cantidad de tipos de datos para las columnas.
- Gran portabilidad entre sistemas.
- Gestión de usuarios y passwords flexible, manteniendo un muy buen nivel de seguridad en los datos (fuerte protección de datos).
- Infinidad de librerías y otras herramientas que permiten su uso a través de gran cantidad de lenguajes de programación, ya dispone de API's en gran cantidad de lenguajes (C, C++, Java, PHP, etc).

- Gran rapidez y facilidad de uso.
- Fácil instalación y configuración.
- Se puede descargar su código fuente (esto ha favorecido muy positivamente en su desarrollo y continuas actualizaciones).
- Utilización gratuita.
- Cada base de datos cuenta con tres archivos (uno de estructura, uno de datos y uno de índice) y soporta hasta treinta y dos índices por tabla, con registros de longitud fija y variable.
- Robustez (aunque se cuelgue, no suele perder información ni corromper los datos).
- Alto rendimiento.
- Agrupación de transacciones.
- Replicación segura.
- Múltiples motores de almacenamiento.
- Planificación de sventos.
- Conectividad segura.
- Búsqueda e indagación de datos.
- No requiere de mucha memoria RAM.

2.1.1.2. PostgreSQL

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional [Pos].

Características

- Implementación del estándar SQL92/SQL99.
- Soporta distintos tipos de datos y permite la creación de tipos propios.
- Incorpora una estructura de datos *array*.
- Incorpora funciones de diversa índole y permite la declaración de funciones propias, así como la definición de disparadores.
- Soporta el uso de índices, reglas y vistas.
- Incluye herencia entre tablas (objeto-relacional).
- Permite la gestión de diferentes usuarios, como también los permisos asignados a cada uno de ellos.

- Es altamente confiable, en cuanto a estabilidad se refiere.
- Su administración se basa en usuarios y privilegios.
- Control de concurrencia multiversión, que mejora sensiblemente las operaciones de bloqueo y transacciones en sistemas multiusuario.
- Dispone de API's en gran cantidad de lenguajes (C, C++, Java, PHP, etc).
- Conexión estable.
- Integridad transaccional.

2.1.1.3. Oracle

Oracle es un sistema de gestión de base de datos objeto-relacional, que utiliza un lenguaje procedural (PL/SQL), donde los datos son tratados como unidad. Entre sus funciones más importantes está hacer uso de los recursos del sistema informático en todas las arquitecturas de hardware, para garantizar su aprovechamiento al máximo en ambientes cargados de información, lo que combina rendimiento y escalabilidad [Ora].

Características

- Estabilidad.
- Alto Rendimiento en transacciones.
- Multiplataforma.
- Soporta alto volumen de datos.
- Optimización del modelado de datos.
- Intuitivo.
- Posee un registro de cambios.
- Difícil instalación.
- Se puede acceder a la instancia Oracle sin necesidad de abrir la base de datos.
- Permite el uso de particiones para la mejora de la eficiencia de replicación.
- Precio elevado (licencia privada).
- Sistemas de alta disponibilidad.
- Gestión de la seguridad.
- Autogestión de la integridad de los datos.
- Replicación de entornos.
- Escalable (soporta gran volumen de datos).

2.1.1.4. SQL Server

Microsoft SQL Server es un sistema para la gestión de bases de datos producido por Microsoft basado en el modelo relacional. Sus lenguajes para consultas son T-SQL y ANSI SQL [SQL].

Características

- Soporte de transacciones.
- Robustez (menor tiempo de recuperación y realización de copia de seguridad de manera sencilla).
- Seguridad alta (permite administrar permisos a todo).
- Soporta alto volumen de datos.
- Replicación.
- Permite administrar información de otros servidores de datos.
- Soporta procedimientos almacenados.
- Alto consumo de memoria RAM (instalación y utilización SW).
- Menor necesidad de limpieza de las memorias intermedias durante el procesamiento de las transacciones (mejor utilización de la CPU, y No-Consumición de recursos, debido a que los recursos son las propias versiones generadas de los registros).
- Permite la declaración de funciones propias.
- Rápido.
- La cancelación y aceptación de las transacciones son inmediatas.
- Tiempo de respuesta no esperada.
- Soporte de tipos de datos incompleto.
- Potente entorno gráfico.
- Escalable.

2.1.1.5. DB2

Sistema de gestión de base de datos relacional multiplataforma, especialmente diseñada para ambientes distribuidos, permitiendo que los usuarios locales compartan información con los recursos centrales que integra XML de manera nativa (propiedad de IBM) [DB2].

Características

- Automatización.
- Multiplataforma.
- Escalabilidad sencilla.
- Alta disponibilidad.
- Memoria que optimiza el rendimiento.
- Robustez.
- Facilidad de instalación y uso.
- Universalidad.

2.1.1.6. Comparación

A continuación, se muestran las tablas 2.1 y 2.2 con la comparación de las características más importantes. En ellas se denotará con un ✓ que posee esa característica, y con ✓✓, que destaca por encima de las demás.

	Velocidad	Robustez	Alto Rendimiento	Limitación Tamaño Registros	Control Acceso	Bajo Consumo Recursos
MySQL	✓✓	✓✓	✓			✓
PostgreSQL	✓	✓		✓	✓	
Oracle	✓	✓✓	✓✓		✓	
SQL Server	✓	✓		✓	✓	✓✓
DB2	✓	✓✓	✓	✓	✓✓	

Tabla 2.1: Comparación SGBDR 1

Finalmente, podemos concluir que el SGBDR más potente en el mercado a día de hoy es Oracle, pero, por no ser un sistema gratuito y tener un alto costo, tendremos que descartarlo. Lo mismo ocurre con el siguiente sistema más potente, el SQL Server, aunque en este caso, incluiremos como causa de exclusión el hecho de que sólo sea compatible

	Multiplataforma	Seguridad	Escalable	Soporta Transacciones	Soporta definición de funciones
MySQL	✓✓		✓	✓	
PostgreSQL	✓	✓	✓	✓	✓
Oracle	✓✓	✓✓	✓✓	✓	✓✓
SQL Server		✓	✓	✓	✓
DB2	✓	✓	✓	✓✓	✓

Tabla 2.2: Comparación SGBDR 2

con Windows. A la misma altura estaría, si no fuera porque no posee un bajo consumo de recursos, DB2, que, al igual que los dos anteriores, posee un alto costo, por lo que también será descartado.

Nos encontramos entonces con que nuestra elección se acotará a los 2 sistemas más populares MySQL y PostgreSQL. Generalmente, MySQL será usado para sistemas en los que la velocidad y el número de accesos concurrentes sea algo primordial, y donde la seguridad no sea muy importante y pueda bastar con hacer backups periódicos, que se restaurarán tras una caída del servidor. En cambio, para sistemas más serios en los que la consistencia de la BD sea fundamental (BD con información realmente importante, bancos, etc.) PostgreSQL es una mejor opción, pese a su mayor lentitud.

2.1.1.7. Herramienta visual de diseño de bases de datos

Cada uno de los sistemas anteriormente analizados dispone de una herramienta visual de diseño de bases de datos que nos facilitará el uso de éstas, integrando el desarrollo de software, la administración de bases de datos y su diseño, y la creación y mantenimiento del sistema de bases de datos propio.

Con esta herramienta, se nos ofrece una interfaz intuitiva que permite realizar rápida y fácilmente tareas con la base de datos.

Para cada sistema existe un amplio conjunto de herramientas con este fin. Nosotros recogeremos una selección en la tabla 2.3.

En este caso, el uso de cualquiera de los sistemas para un SGBDR dado será igual de válido. Cabe destacar TOAD (Tool for Oracle Application Developers), ya que es el único que soporta los distintos SGBDR. No tiene soporte para ningún entorno ajeno a Microsoft,

SGBDR	Herramientas
MySQL	MySQL Workbench, TOAD, DreamCoder, ...
PostgreSQL	pgaccess, TOAD, DreamCoder, ...
Oracle	SQL Developer, TOAD, DreamCoder, ...
SQL Server	Management Studio Express (SSMSE), TOAD, ...
DB2	DB2 Query Patroller, TOAD, ...

Tabla 2.3: Herramientas para los SGBDR

pero existe una variante de código abierto llamada **TORA** (Toolkit for Oracle) para esos casos.

2.1.2. Software Intermedio o Middleware

En esta sección, primero vamos a evaluar las distintas opciones que se nos presentan: Servidor de Aplicaciones y Servidor Web.

2.1.2.1. Servidor de Aplicaciones

Se trata de aquellos servidores que proporcionan la lógica de negocio sobre la que construir aplicaciones (Servlet). Las principales ventajas del uso de la tecnología de servidores de aplicación son la centralización y la disminución de la complejidad en el desarrollo de aplicaciones. Además, brindan soporte a una gran variedad de estándares, tales como HTML, XML, etc., que les permiten su funcionamiento en ambientes web y la conexión a una gran variedad de fuentes de datos, sistemas y dispositivos.

El término “servidor de aplicaciones” usualmente hace referencia a un servidor de aplicaciones J2EE, aunque, también ha sido aplicado a otros productos no-J2EE. Entre los servidores de aplicación Java EE más conocidos se encuentran WebLogic de Oracle y WebSphere de IBM, JBoss AS de JBoss, entre otros.

2.1.2.2. Servidor Web

Un Servidor Web, es, esencialmente, un programa que escucha en un puerto a la espera de conexiones. Cuando se detecta una, el servidor espera a recibir una petición en formato adecuado desde la aplicación cliente (navegador, gestor de download, etc.). Tanto la petición, como la respuesta, se encapsulan siguiendo el protocolo HTTP.

Hoy en día existen multitud de productos que funcionan como servidores Web. Entre los más conocidos encontramos Apache, Microsoft IIS,...

Puesto que casi todos los servidores Web actuales son también servidores de aplicaciones, ya que incluyen alguna tecnología que permite crear aplicaciones con contenido dinámico (CGI, PHP, JSP, etc.), utilizaremos este tipo de servidor para nuestro diseño.

2.1.2.2.1. Apache

Apache-Tomcat es un Servidor Web, con soporte para *servlets* y JSP escrito en Java/J2EE (parte Tomcat), autónomo en entornos con alto nivel de tráfico y alta disponibilidad, multiplataforma, escalable, seguro y que soporta distintos lenguajes de programación (PHP, Perl, Ruby, Python etc.) [Apa].

2.1.2.2.2. Microsoft IIS

Internet Information Server (IIS) es el Servidor Web (y FTP) estrella de Microsoft ambientado en ASP.

Los principales problemas son la falta de una distribución para Unix (debido a que no sólo habría que portar IIS, sino el conjunto completo de tecnologías Microsoft con las que se integra), los conocidos problemas de seguridad y su precio [IIS].

2.1.2.2.3. Weblogic

Weblogic es un servidor de aplicaciones J2EE (basado en Servlets y JSP) y un Servidor Web de Oracle, aunque puede usar otras bases de datos (DB2, SQL server, MySQL...), multiplataforma, con interoperabilidad con .NET (dominio de nivel superior genérico utilizado en el Sistema de Nombres de Dominio de Internet) y seguro, con implementación cuidada y eficiencia en características avanzadas. Puede colaborar con los servidores Web más utilizados (Apache, IIS, iPlanet, etc.).

La principal desventaja es su elevado precio [Web].

2.1.2.2.4. Comparación

Completaremos este análisis, como en los casos anteriores, comparando las propiedades más importantes.

	Multiplataforma	Gratuito	Seguro	Java
Apache	✓	✓	✓	✓
IIS				
Weblogic	✓		✓	✓

Tabla 2.4: Comparación de Servidores Web

Para complementar este análisis, analizaremos también la compatibilidad de los distintos servidores con los SGBDR que seleccionamos en el capítulo 2 como posible elección, en la tabla que se expone a continuación. En ella, encontramos ✓* para indicar que dicha base de datos es soportada, pero no nativamente.

	MySQL	PostgreSQL
Apache	✓	✓
ISS	✓	✓*
Weblogic	✓	✓*

Tabla 2.5: Servidores Web & SGBDR

Como podemos ver, aunque Weblogic posee una mejor implementación para las mismas propiedades que Apache-Tomcat, dado su alto coste, tendrá que ser descartado pasando a ser este último la mejor opción a usar en nuestro caso.

2.2. Cliente

En este capítulo analizaremos las tecnologías correspondientes a la parte del Cliente, ya que este es totalmente independiente del Servidor.

El Cliente está compuesto por una serie de estructuras programáticas que le permiten desplegar y ejecutar documentos e instrucciones que son cargados en él. Este tipo de instrucciones pueden variar desde documentos estáticos (**HTML**), contenido dinámico (**JavaScript**), interactivas (**AJAX**), contenido multimedia (**Flash y Shockwave**), etc.

Estas tecnologías permiten que los procesos se ejecuten del lado del cliente, con el fin de acelerar las conexiones y no sobrecargar los servidores.

2.2.1. HTML/XHTML

HTML (Hypertext Markup Language) es un lenguaje estructurado de representación utilizado para la creación de páginas web.

Hoy en día, la última versión de HTML ha evolucionado a un nuevo estándar denominado XHTML (eXtensible Hypertext Markup Language). Esta especificación se encuentra basada en el lenguaje de marcación XML (eXtensible Markup Language), que representa un lenguaje más simplificado.

Aunque XHTML representa un cambio de raíces, de SGML a XML, son sólo unas pequeñas diferencias las que deben ser contempladas para diseñar un documento XHTML

[W3].

2.2.2. JavaScript

Si bien es cierto que HTML / XHTML no es capaz ejecutar cierto tipo de lógica, ésta es sólo una de las razones por las que surgieron *Scripting Languages* como JavaScript.

Es un lenguaje de programación bastante sencillo y pensado para hacer las cosas con rapidez, a veces con ligereza (van escritos dentro de una página en HTML, es decir, formando parte del propio código HTML de dicha página). Incluso las personas que no tengan una experiencia previa en la programación podrán aprender este lenguaje con facilidad y utilizarlo en toda su potencia con sólo un poco de práctica. Por ello, y por una gran compatibilidad, es el lenguaje de programación del lado del Cliente más utilizado [Java].

NOTA: No debemos confundir Java y JavaScript, ya que no están relacionados y tienen semánticas y propósitos diferentes.

2.2.3. Flash y Shockwave

Utilizar Flash y Shockwave presenta dos grandes diferencias con respecto a utilizar HTML / XHTML y JavaScript: el elevado costo y la complejidad del diseño, puesto que se requiere llevar a cabo una serie de pasos intermedios entre la creación del código y su colocación en un documento.

Es una tecnología muy utilizada pero en extinción, ya que se prevee que desaparezca con la llegada de HTML5 (nuevo estándar de HTML, que, sin versión final, ya empieza a ser utilizada por páginas como facebook).

Pero además, Flash también posee facilidades para interactuar directamente con el navegador (el Cliente) a través de JavaScript o Formas HTML / XHTML [Ado].

2.2.4. AJAX

AJAX (Asynchronous JavaScript and XML) es una suma de técnicas de desarrollo web que permite que las aplicaciones creadas se ejecuten en el navegador y se comuniquen de forma asíncrona con el servidor, de forma transparente al usuario. Con esto se consigue modificar la página sin necesidad de recargarla, y se aumenta la interactividad. Las tecnologías que usa AJAX son:

- XHTML y CSS, para crear una presentación basada en estándares.
- DOM, para la interacción y manipulación dinámica de la presentación.
- XML, XSL y JSON, para el intercambio y manipulación de la información.
- XMLHttpRequest, para el intercambio y la manipulación de información.
- JavaScript, para unir todas las tecnologías.

Además, AJAX posee compatibilidad con tecnologías como, por ejemplo, Flash [AJA].

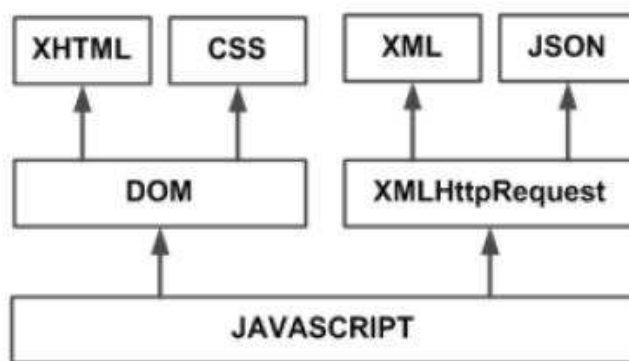


Figura 2.2: Tecnologías AJAX

2.2.5. Comparación

En este caso, aunque dicha comparación se ha ido haciendo durante la propia exposición, como soporte incorporamos la tabla 2.6 .

	Estructurado & simple	Dinámico	Maneja Objetos	Seguro	Multimedia	Interactivo
HTML	✓	✓				
JavaScript	✓	✓	✓	✓		
Flash					✓	
AJAX	✓	✓	✓	✓		✓

Tabla 2.6: Comparacion Tecnologías Cliente

Tras este análisis, podemos decir que la mejor tecnología a usar, en nuestro, caso es AJAX.

2.3. Comunicación Cliente Servidor

Cuando queremos transmitir datos entre dos máquinas, almacenar la configuración o guardar datos para un uso posterior, tenemos que decidir el formato del archivo que estamos usando.

En este caso, nos centraremos en los más simples: **XML**, **JSON**, y **YML**.

2.3.1. XML

XML (eXtensible Markup Language) es un metalenguaje que fue diseñado básicamente para estructurar, almacenar e intercambiar datos entre diferentes aplicaciones.

Posee una gran variedad de herramientas asociadas (CSS, DTD, XPATH, Schema,...), es muy potente, y se utiliza en sistemas de comunicación complejos y cuando el orden de los datos es relevante [W3].

2.3.2. JSON

JSON, acrónimo de JavaScript Object Notation, es un formato ligero para el intercambio de datos, un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML. Puede ser muy compacto y eficiente si se usa de manera correcta.

Resulta muy útil, sencillo, intuitivo, y es fácil procesarlo automáticamente.

Existen interpretes de JSON para la mayoría de lenguajes de programación [JSO].

2.3.3. YAML

Similar a XML en características, legible (pensado para ser leído por el ser humano más que por la máquina) y orientado a serialización de objetos en flujos de datos.

YAML es un superconjunto de JSON que trata de superar algunas de las limitaciones de éste, aunque es significativamente más complejo, con portabilidad entre lenguajes de programación [Yam].

2.3.4. Comparación

Con la tabla 2.7 en se realiza la comparación.

	Simple	legible	Ordenado	Ofrece Herramientas de Desarrollo
XML	✓	✓	✓	✓✓
JSON	✓✓	✓✓	✓✓	✓
YAML		✓	✓	✓

Tabla 2.7: Comparación Lenguajes Comunicación

Los tres son lenguajes adaptativos y óptimos, pero, aunque XML goza de mayor soporte y ofrece muchas más herramientas de desarrollo que JSON, JSON es algo más lineal,

simple, ordenado y legible que XML (facilitando las pruebas). Por tanto, elegiremos JSON para nuestro proyecto.

2.4. Conclusiones

Después del análisis teórico realizado, podemos concluir que para el sistema estudiado la combinación óptima, será aquella que se implemente con un Apache-Tomcat como Servidor Web, AJAX como técnica de desarrollo web, JSON como lenguaje de comunicación y MySQL o PostgreSQL como sistema de gestión de base de datos (dependiendo de las características de nuestro sistema).

En el siguiente capítulo, se realizará un estudio práctico para fortalecer nuestro análisis y ajustar a nuestro proyecto el SGBD a emplear.

Selección de las tecnología a usar en el proyecto

Para una adecuada selección de las tecnologías a emplear, se ha reforzado el análisis anterior realizando un estudio desde un punto de vista práctico, en el cual, se han implementado dos sistemas, donde se combinan varias de las tecnologías estudiadas.

Aunque explícitamente no se haya mencionado, primero debemos elegir el Sistema Operativo sobre el que montaremos nuestro Cliente-Servidor. En nuestro proyecto elegimos Linux frente a otras opciones más usadas, ya que posee una mayor seguridad, estabilidad y portabilidad. Sobre él, realizaremos las distintas pruebas.

Llevaremos a cabo 2 sistemas, cada uno de ellos para cada unas de las SGBDRs elegidas como posibilidad (PostgreSQL y MySQL). Para ambos, las tecnologías empleadas en el Cliente y en la Comunicación Cliente-Servidor permanecerán fijas, ya que durante el análisis se llegó a una solución concluyente (JSON y AJAX).

Aunque se estudió previamente, los tipos de tecnologías a emplear en el Servidor (llegando finalmente a la conclusión del uso de Apache-Tomcat), en ningún momento se concretó, dentro de las muchas posibilidades que existen, el lenguaje que se emplearía para realizar la aplicación servidora correspondiente. Entre los más populares se encuentran PHP y *Servlets*, ambos con un mismo propósito, pero que dependiendo de nuestro sistema, serán más o menos adecuados. En este caso, en vez de realizar un estudio teórico de ambos lenguajes, procederemos a un estudio práctico, por lo que, para cada uno de los casos que aquí se detallan, usaremos un lenguaje distinto.

3.1. Sistema 1: LAMP (Linux, Apache, MySQL y PHP)

Como base Linux de nuestro sistema, usaremos un Ubuntu Server, con un MATE (entorno de escritorio bifurcado de GNOME 2) para el entorno gráfico.

3.1.1. Diseño Arquitectura

Para este caso, la arquitectura que vamos a implementar es la del diagrama que se muestra en la figura .

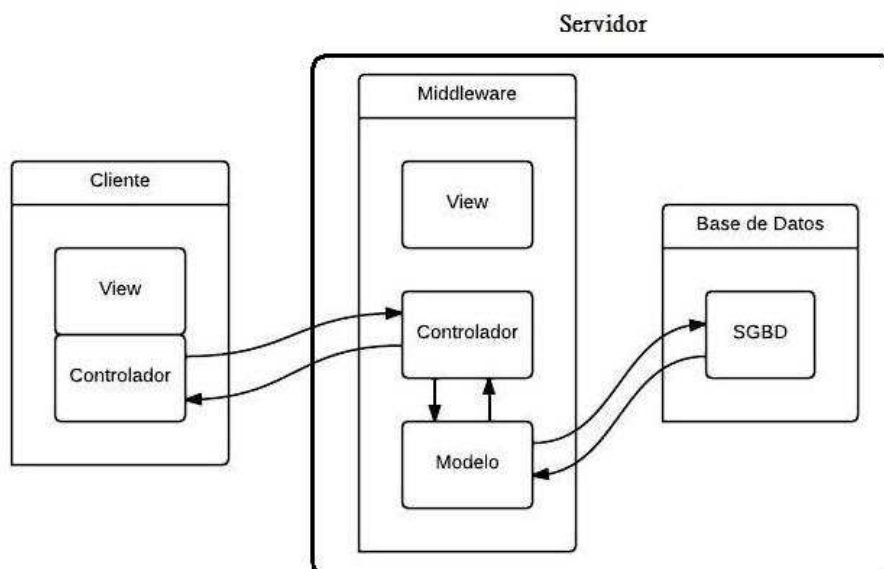


Figura 3.1: Esquema Arquitectura Sistema 1

Cliente:

- AJAX

Middleware:

- MVC (Modelo Vista Controlador) con Symfony, PHP 5
- Doctrine, como capa de abstracción con la base de datos.

Base de datos:

- MySQL

Comunicación entre sistemas:

- REST/HTTP-JSON

3.1.2. Instalación

Actualizar el sistema

```
$ sudo aptitude update
```

Instalar Apache2

```
$ sudo apt-get install apache2
```

Instalar MySQL

```
$ sudo apt-get install mysql-server
```

Instalar PHP

```
$ sudo apt-get install php5 libapache2-mod-php5 php5-mysql
```

Reiniciaremos Apache2 para aplicar los cambios.

```
$ sudo /etc/init.d/apache2 restart
```

Es aconsejable instalar los siguientes módulos de PHP5 (JSON en particular).

```
$ apt-get install php5-mysql php5-curl php5-gd php5-idn php-pear php5-  
imagick php5-imap php5-mcrypt php5-memcache php5-mhash php5-ming php5-  
ps php5-pspell php5-recode php5-snmp php5-sqlite php5-tidy php5-xmlrpc  
php5-xsl php5-json  
$ sudo /etc/init.d/apache2 restart
```

Instalar PHPMyAdmin

Esta herramienta visual será la elegida para la comprobación del estado de la base de datos, tras los distintos cambios realizados durante las pruebas.

```
$ sudo apt-get install phpmyadmin
```

Instalar Symfony

Este programa es un *Framework* (marco de trabajo) que empleamos para la creación de la aplicación servidora en PHP. Aunque nos proporciona un MVC que podría satisfacer en cierto modo nuestro sistema de tres capas completo, no emplearemos las tres capas, siendo para nosotros la capa Vista inexistente.

Para instalar, hay que seguir los pasos indicados en [Sym], creando durante el proceso el proyecto (*pruebaPFC*) y la aplicación (*prueba*), y configurando el servidor Apache para que se dirija a la aplicación adecuada.

NOTA: Existen más PHP framework que podrían usarse igualmente, teniendo en cuenta lo anteriormente mencionado.

NOTA2: La instalación de un Tomcat no será necesaria, al no emplearse en este caso servlets.

3.1.3. Creación del proyecto

En esta sección se explicarán y justificarán las distintas partes de las que consta nuestro sistema de prueba.

3.1.3.1. Modelo de datos

Tras instalar los distintos componentes en nuestro sistema Operativo, en este primer caso, empezaremos creando un modelo de datos, que tendrá dos tablas con varios campos, los cuales estarán relacionados tal y como se muestra a continuación.

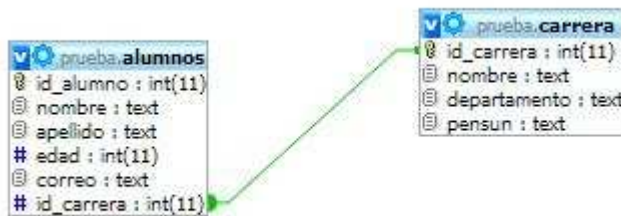


Figura 3.2: Estructura Tabla SGBDR MySQL

3.1.3.2. Servidor

Symfony es un *framework* orientado a objetos, por lo que manipularemos objetos, en lugar de escribir sentencias SQL para recuperar registros de la base de datos. Para ello, la información de BD relacional debe ser asignada a un modelo de objetos. Esto se puede hacer con una herramienta ORM (Object-Relational Mapping) que Symfony tiene incluido (Doctrine).

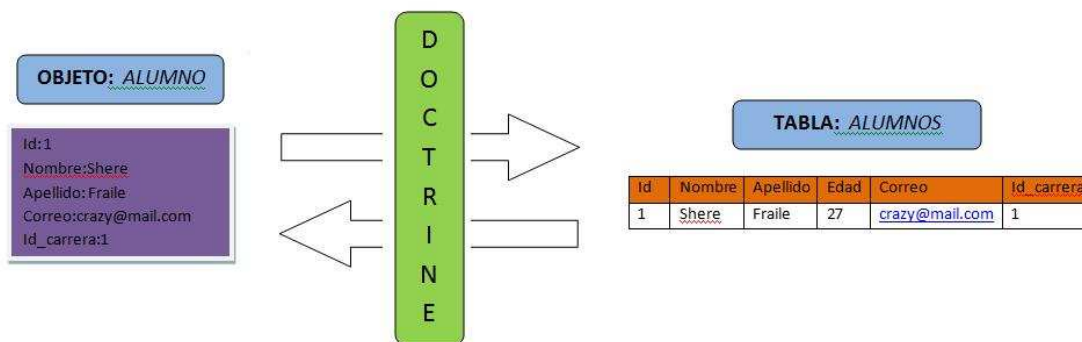


Figura 3.3: Doctrine

El ORM necesita una descripción de las tablas y sus relaciones para crear las clases relacionadas. Existen distintas formas de hacer esto, pero, en este caso, vamos a crear el archivo de esquema a mano.

Con PHPMyAdmin creamos la base de datos, pero no incluimos nada dentro, ya que, ejecutando los comandos que se muestran a continuación, mapearemos los objetos PHP con la base de datos.

```
$ php symfony propel:build-schema
$ php symfony propel:build-sql
$ php symfony propel:insert-sql
$ php symfony propel:build-form
$ php symfony propel:build-filters
```

Una vez finalizado el mapeo, crearemos un modulo para cada una de las tablas (*Modulo Alumnos* y *Modulo Carreras*) y ya dispondremos de una aplicación servidor. Symfony habrá creado un directorio en el que se encuentra toda la información, aunque será necesario realizar cambios sobre él para conseguir nuestro objetivo. Concretamente, se trata de modificar las acciones a realizar de la aplicación, incluyendo en ellos la codificación y decodificación de los mensajes a intercambiar (funciones `json_encode` y `json_decode`). Estos cambios se realizarán en la capa controlador del sistema, que se compone de diferentes funciones a modo *webservice*, que devolverá la información en formato JSON y la recibirá como POST para añadir nuevos elementos a GET HTTP cuando se realicen consultas.

Antes de explicar las modificaciones a realizar con PHPMyAdmin, crearemos un elemento de la *Tabla Carrera* (que también podría hacerse mediante archivo). De este modo, nuestro sistema se basará exclusivamente en alumnos de una carrera universitaria, que interactúan con las modificaciones anteriormente mencionadas.

La modificación a realizar consistirá en editar el archivo que se encuentra en `/apps/-prueba/modules/alumnos/actions/action.class.php`, donde *prueba* es el nombre de nuestra aplicación y *alumnos*, el nombre del modulo/tabla, en el que vamos a realizar las modificaciones.

Dicho archivo se compone de cinco métodos principales, correspondientes a las modificaciones anteriormente comentadas:

- **executeIndex** : método que devuelve los parámetros al completo de la base de datos.
- **executeShow** : método que realizará una búsqueda para un nombre determinado, devolviendo todos sus atributos.
- **executeNew** : método que creará un nuevo objeto/registro en la base de datos, que tendrá en cuenta si el este ya existe en la misma.
- **executeEdit** : método que realizará la modificación de un objeto de la base de datos, siempre y cuando exista.
- **executeDelete** : método que elimina un objeto/registro de la base de datos.

3.1.3.3. Cliente

El cliente empleado será una página web creada con AJAX donde el HTML corresponderá a la parte vista y el código JavaScript, a la parte del controlador. Tendrá cinco acciones sobre la base de datos: consulta, búsqueda, borrado, inserción y modificación.

Aunque en este sistema la parte JavaScript va dentro del propio código HTML, podría ir en un archivo diferente. Normalmente para una mayor optimización, en un proyecto

de mayor importancia se encuentran separados. A continuación, mostraremos las distintas funciones implementadas.

- **crear** : función que crea dinámicamente el formulario a rellenar para introducir un nuevo alumno en la base de datos. Al pulsar *Submit*, llamaremos a la función **addAlumno**.
- **addAlumno** : función que hace una petición a **executeNew** y nos muestra la respuesta (ok o ko).
- **showCustomer** : función que hace una petición a **executeShow** y nos muestra la respuesta (nos muestra el alumno, si existe, o un mensaje, si no existe).
- **consulta** : función que hace una petición a **executeIndex** y nos muestra la respuesta (nos muestra todos los alumnos de la base de datos en un tabla o nos indica que no hay alumnos con un mensaje).

3.1.3.4. Comunicación

Por motivos de seguridad, los navegadores no permiten realizar llamadas AJAX a dominios diferentes. Puesto que nuestro Cliente y nuestro Servidor son dos aplicaciones distintas, y por tanto, se encuentran en dominios distintos, deberemos realizar las modificaciones correspondientes para la que la comunicación pueda llevarse a cabo.

Para ello, debemos habilitar el módulo de cabeceras de Apache.

```
$ sudo a2enmod headers
```

Podría ocurrir que no se encuentre dicho módulo, en cuyo caso se puede solucionar con un *update* o configurando manualmente las *header* de Apache (configuración avanzada de Apache, headers [con]).

A continuación vamos al fichero que hemos creado con Symfony para añadir nuestro Virtualhost, incluyendo la etiqueta <Virtualhost >.

```
Header set Access-Control-Allow-Origin *
```

Para finalizar, reseteamos Apache y ya tendremos la configuración AJAX **cross-site domain**.

Finalmente vamos a tratar el formato del que constarán nuestras respuestas. Para cualquier petición de las deseadas podemos tener dos tipos de respuestas: éxito o error en la acción. Por ello, nuestra respuesta constara de una primera cabecera **tipo**, que será 0 en el caso en el que se produzca un error, y 1 en el caso de que la operación se haya realizado con éxito en la base de datos. El tipo de contenido de cada una de las respuestas será diferente, pero en ambos casos irá dentro del campo **list**: en el caso del tipo 0, este campo irá únicamente acompañado de un mensaje de texto en el que se describirá el problema; mientras que en el tipo 1, el campo contendrá uno o más alumnos (*array* de *arrays*), para el caso de búsquedas o consultas, o un mensaje de confirmación, para el caso de edición, inserción o

borrado. Un ejemplo de tipo 1, con un único alumno (resultado de llamar nuestro a método show, creado en Symfony, con el nombre concreto de Shere), sería:

```
{ "tipo":1,"list":[{"id":"1","nombre":"Shere","edad":"27","email":"sherenei@gmail.com"}]}
```

Un ejemplo de tipo 1, con más de un alumno (resultado de llamar nuestro método Index, creado en Symfony), sería:

```
{ "tipo":1,"list":[{"id":"1","nombre":"Shere","edad":"27","correo":"sherenei@gmail.com"}, {"id":"2","nombre":"Ovi","edad":"29","correo":"ovi@gmail.com"}]}
```

Un ejemplo de tipo 0:

```
{ "tipo":0,"list":"El alumno seleccionado, no existe"}
```

3.1.4. Sistema Completo

Finalmente, el sistema completo funciona del modo que se describe a continuación.

Al entrar en la sistema a través de la dirección *localhost/buscar.html*, se muestra en el navegador la página que vemos en la figura 3.4.

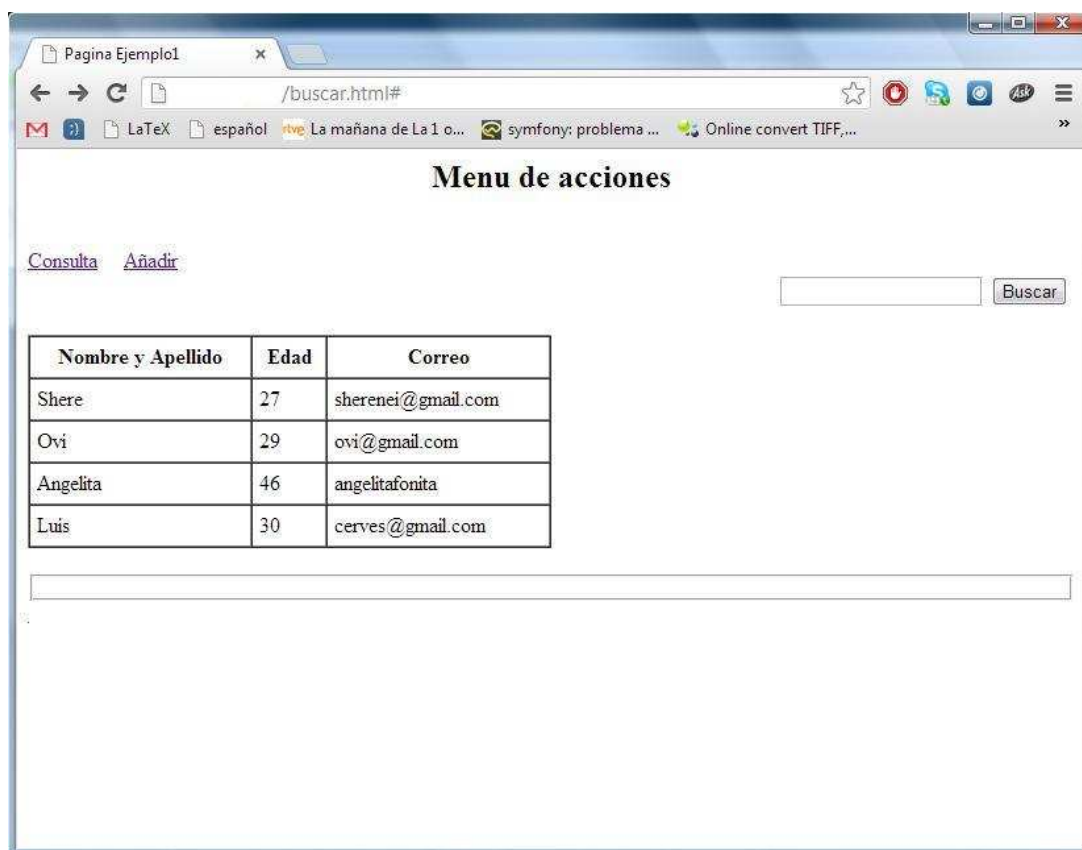


Figura 3.4: Página Inicio Estudio

La página consta de tres secciones importantes. Un primer div (elemento HTML que se emplea a nivel de bloque como contenedor de otros elementos) donde se encuentran

las tres opciones implementadas del menú: consulta, añadir y buscar (de borrar y editar hablaremos más adelante). Debajo de este div, encontramos otro, donde se mostrarán las distintas respuestas a las peticiones (en forma de una tabla o de un mensaje). Y un div inferior, que sólo tendrá algún contenido en el caso de querer introducir un nuevo alumno.

Inicialmente se muestra el resultado de realizar una consulta, una tabla con todos los elementos en la base de datos, tal y como se ve en la figura 3.4.

Tanto desde la visualización de la página actual, como desde cualquiera de las visualizaciones que vamos a comentar, nos desplazaremos de unas a otras simplemente utilizando el menú.

Aunque actualmente la base de datos tiene un número muy bajo de alumnos, en un caso real, podría ser demasiado grande para su consulta. Si quisiéramos buscar los datos de un alumno en concreto, sería muy difícil, por ello se implementa la opción buscar. Al introducir el nombre del alumno a buscar y pulsar sobre el botón buscar (con un intro sólo no vale), nuestra página se actualiza, pero no completamente (gracias al uso de AJAX), sólo el div que muestra la tabla, para pasar a mostrar la respuesta recibida, como se ve en la siguiente figura 3.5.



Figura 3.5: Página Buscar Estudio

En caso de error, la imagen variaría, cambiando la tabla por un mensaje de error.

Por último, en caso de querer introducir un nuevo alumno, al pulsar sobre el enlace, actualizaremos el div que se encuentra en la parte inferior de la página. En él, nos mostrará un formulario a rellenar, para introducir los datos del alumnos (véase figura 3.6).

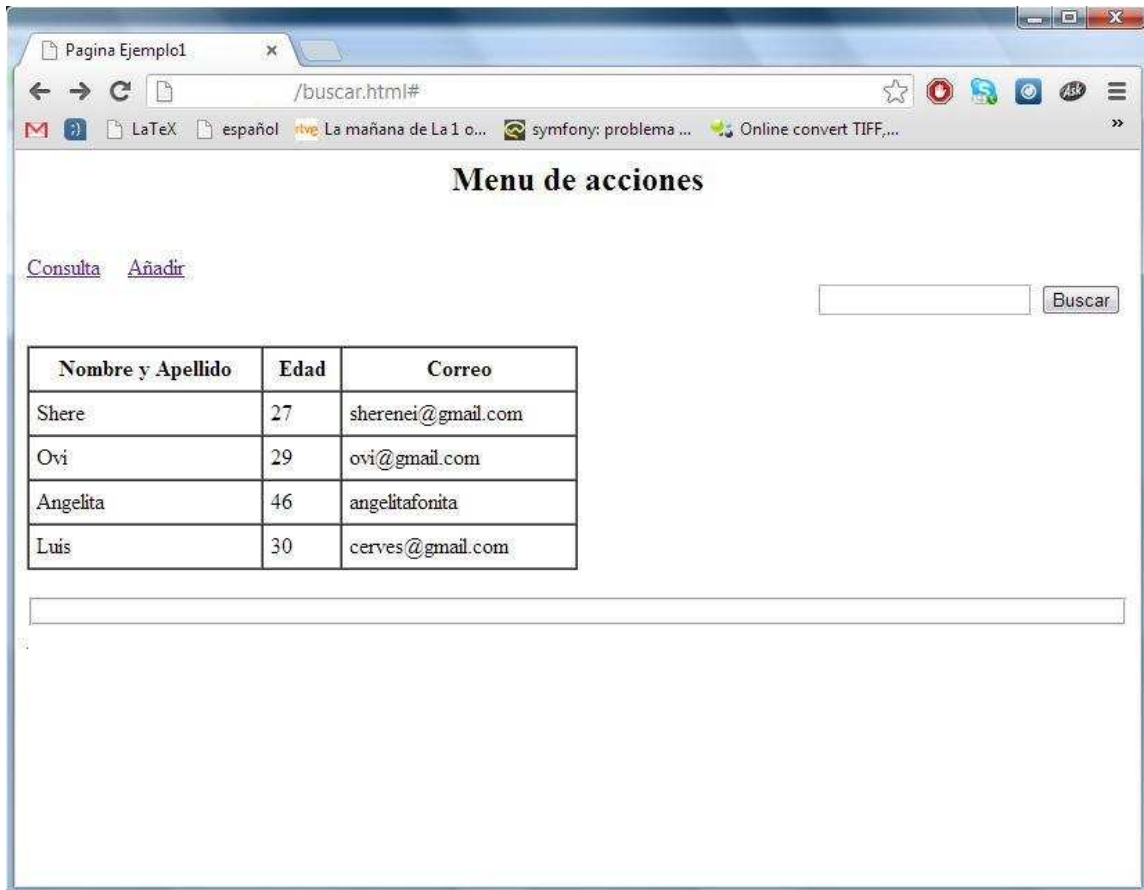


Figura 3.6: Página Add Estudio

Una vez cumplimentado el formulario, los dos divs inferiores (aquellos que no contienen el menú), cambiarán. En el caso del inferior de ambos simplemente pasara a estar vacío. Mientras que en el superior se mostrará el mensaje de error (ya existe) o éxito en la operación.

3.1.5. Esquema Completo

Para finalizar, resumiremos todo lo anterior en un diagrama completo de nuestro sistema, especificando el nombre de los archivos y métodos más importantes, y cómo éstos interactúan entre ellos para llevar a cabo las funciones especificadas anteriormente.

El diagrama es el correspondiente a la figura 3.7, que se muestra a continuación.

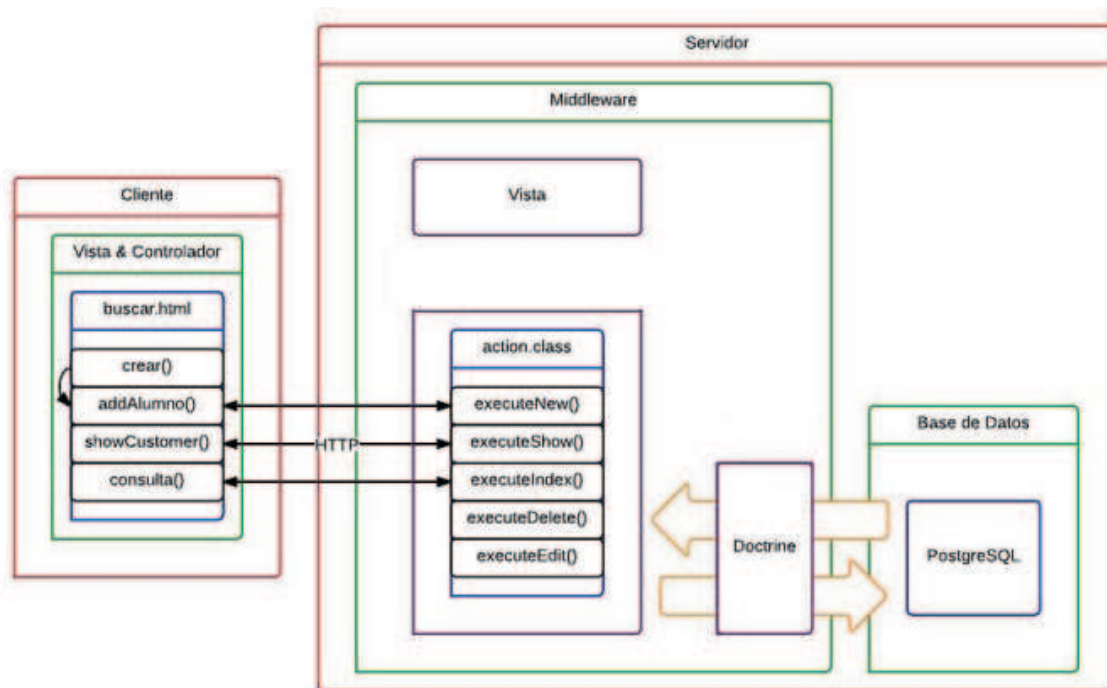


Figura 3.7: Esquema Arquitectura completa Sistema I

3.1.6. Mejoras

Dado que inicialmente se habla de cinco funciones a comprobar, pero solo tres están implementadas en JavaScript, faltando Editar y Borrar, lo primero sería terminar esta parte. Quizás el hecho de que no dispongamos del acceso mediante la página web, nos haga pensar que el trabajo no sólo debe realizarse en JavaScript, sino también en PHP, pero esa parte está concluida. Para verificar este hecho, bastará con realizar la llamada a `localhost:8080/alumnos/Delete?nombre=Shere` para borrar o a `localhost:8080/alumnos/Edit?nombre=Edad&apellido=Apellido&edad=27&correo=correo` para editar.

Aunque la parte de `actions.class` ha sido terminada, también podría mejorarse. Por ejemplo, la búsqueda se ha programado para una única coincidencia, es decir, en caso de que encontráramos dos usuarios con el mismo nombre, se cogería sólo el primero. En el caso de editar y borrar, también se ha planteado la búsqueda por nombre, que podría llevarnos a equívoco, mientras que si ésta fuera por `id`, no cabría posibilidad de error. Esto fue elegido así para una mayor facilidad a la hora de realizar las distintas pruebas, ya que los `ids` se crean consecutivamente, pero sin tener en cuenta los eliminados. Para que éstos sean suplidos, deberán realizarse algunos cambios.

A pesar de todo ello, el sistema implementado cumple las funciones de aprendizaje y análisis deseadas.

3.2. Sistema 2: LAMP (Linux, Apache, Middleware (JSP) y PostgreSQL)

Emplearemos la misma base que en el caso anterior para el sistema operativo y el entorno gráfico.

3.2.1. Diseño Arquitectura

Para este caso, la arquitectura que vamos a implementar es la del diagrama de la figura 3.8 que se muestra a continuación.

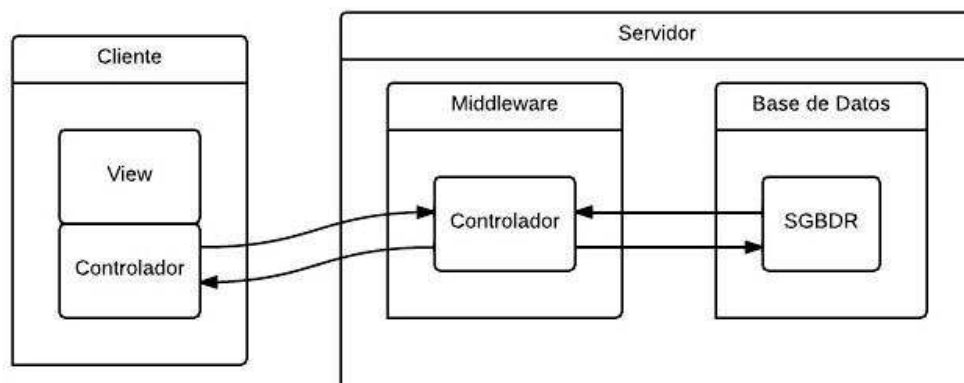


Figura 3.8: Esquema Arquitectura Sistema2

Cliente:

- AJAX

Middleware:

- *Servlet* de Java

Base de datos:

- PostgreSQL

Comunicación entre sistemas:

- REST/HTTP-JSON

3.2.2. Instalación

Para llevar a cabo la instalación, actualizaremos el sistema e instalaremos Apache2 y PHP5 (debemos cambiar *php5-mysql* por *php5-pgsql*), como en el caso anterior.

Instalar PostgreSQL

```
$ sudo apt-get install postgresql
```

Instalar Tomcat

En este caso instalaremos un Apache-Tomcat de última versión, ya que Eclipse sólo nos permite enlazar con versiones superiores a la 3.2.

Deberemos ir a la página [Tom] y descargar en la sección Core el archivo *.tar.gz, que descomprimos y reubicamos mediante comando.

```
$ tar xvzf Apache-tomcat-7.1.4.tar.gz
$ sudo mkdir /usr/share/tomcat7
$ sudo mv Apache-tomcat-7.1.4/ /usr/share/tomcat7
```

Instalar PHPPgAdmin

PHPPgAdmin en PostgreSQL (por este programa, se instaló previamente PHP).

```
$ sudo apt-get install phpPgAdmin
```

Instalar JDK

Los *servlet* son una clase Java, por lo que es necesaria la instalación de J2EE.

```
$ sudo add-apt-repository ppa:webupd8team/java
$ sudo apt-get update
$ sudo apt-get install oracle-java7-installer
```

Una vez instalado, deberemos crear la variable `JAVA_HOME` (`JAVA_HOME=/usr/lib/jvm/java-7-oracle`) en `/etc/environment`.

Instalar Eclipse

Este programa es necesario, ya que nos facilitará el trabajo a la hora de hacer conexiones *servlet-Tomcat* y *servlet-PostgreSQL*.

```
$ sudo apt-get install eclipse
```

Si intentamos ejecutarlo directamente, nos dará un problema, que se resolverá con el siguiente comando:

```
$ sudo ln -s /usr/lib/jni/libswt-* ~/.swt/lib/linux/x86/
```

Por defecto Eclipse no lleva instaladas las opciones web que nosotros vamos a usar, por lo que será actualizar el Eclipse desde el propio programa, seleccionando la opción JEE, servicio web.

NOTA: Debemos arrancar Tomcat7 para que nuestro sistema pueda funcionar (no solo Apache2):

```
$ sudo /usr/share/tomcat7/bin/startup.sh
```

3.2.3. Creación del proyecto

En esta sección se expondrán las características de las distintas partes de este sistema de prueba.

3.2.3.1. Modelo de datos

Emplearemos el mismo modelo de datos que en el Sistema1 (ver figura 3.2).

3.2.3.2. Servidor

En este sistema, como ya avanzamos, crearemos un *servlet*, que extenderá la funcionalidad de los *servlet* propios del Apache-Tomcat. Se tratará de un programa que, tras recibir la petición del cliente, actuará en consecuencia, invocando otras clases.

Comenzaremos creando un *Dinamic Web Project* en Eclipse (que asociaremos a nuestro Apache instalado en *usr/local/* y en el cual debemos activar la generación del web.xml Deployment Descriptor). En este proyecto encontraremos dos carpetas importantes: **Java Resources:src**, donde guardaremos el código fuente de las clases Java empaquetadas en *packages*, y **WebContent**, que emplearíamos para crear las vistas (HTML..), pero que es una aplicación aparte como ya indicamos, por lo que no será usada.

Con PHPPgAdmin crearemos la base de datos a emplear.

Hay que tener en cuenta que Eclipse no es un *framework*, por lo que necesitará de un **JDBC (Java DataBase Conector)**, que conectara nuestra clase Java con PostgreSQL (en este caso se tratará con la base de datos mediante lengua SQL). Pero el empleo de Eclipse nos beneficia ya que conseguimos una aplicación independiente de la plataforma y capaz de cambiar de un SGBD a otro.

Esta conexión consistirá simplemente en descargarse dicho conector *.jar de internet, y configurarlo en nuestro proyecto Eclipse (*Built path-...-Add external jar*).

Tendremos que tener en cuenta que este tipo de conectores llevan a cabo la conexión del modo que se representa en la figura 3.9.

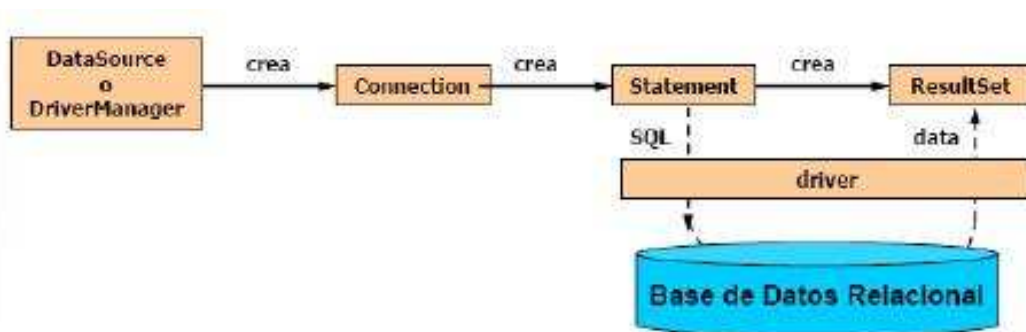


Figura 3.9: Conexión mediante JDBC

La conexión con la base de datos se hará mediante una clase Java llamada *conexion*.

Una vez configurado lo anterior, ya podremos empezar a crear nuestro *servlet* y las clases Java de las que se proveerá. Aunque lo más óptimo sería contar con un solo *servlet*, que conste de los métodos *doPOST* y *doGET*, en nuestro caso, vamos a crear un *servlet* para cada una de las acciones a desempeñar en la base de datos:

- **Listar.java** : devuelve los parámetros al completo de la base de datos.
- **Buscar.java** : método que realizará una búsqueda para un nombre determinado, pudiendo dar más de un resultado, devolviendo todos los atributos.
- **Guardar.java** : método que creará un nuevo registro en la base de datos, que tendrá en cuenta si el objeto ya existe en la base de datos.
- **Editar.java** : método que realizará la modificación de un registro de la base de datos, siempre y cuando exista.
- **Borrar.java** : método que elimina un registro de la base de datos.

Cuando nuestro *servlet* sea invocado, tanto si se trata de una petición GET o POST, la acción a realizar será la misma, por lo que ambos métodos serán redirigidos a otro método de la misma clase, en el cual se realiza la acción correspondiente.

Para que nuestra aplicación pueda estar disponible, el primer paso es publicar el proyecto. Para ello, deberemos ir a sección *server* de Eclipse y añadir el Tomcat7, instalado anteriormente, el cual también añadiremos nuestro proyecto. A continuación, deberemos editar el archivo descriptor de implementación **web.xml** (en WebContent - WEB-INF - lib) para determinar la forma en las que las URL se asignan a los *servlets*, del modo que se muestra a continuación como ejemplo (Listar=nombre del *servlet*, estudio=nombre del proyecto y *servlets* el paquete en el que se encuentra Listar).

```

1 <servlet>
2   <servlet-name>Listar</servlet-name>
3   <servlet-class>estudio.servlet.Listar</servlet-class>

```



```
4     </servlet>
5     <servlet-mapping>
6         <servlet-name>Listar</servlet-name>
7         <url-pattern>/Listar</url-pattern>
8     </servlet-mapping>
9 </web-app>
```

3.2.3.3. Cliente

El cliente, ya que es independiente del servidor y, por tanto, de la base de datos, será el mismo para ambos, cambiando la dirección de la aplicación a llamar, ya que será distinta:

- **addAlumno** : función que hace una petición a **Guardar** y nos muestra la respuesta (ok o ko).
- **showCustomer** : función que hace una petición a **Buscar** y nos muestra la respuesta (información del alumno o mensaje de error).
- **consulta** : función que hace una petición a **Listar** y nos muestra la respuesta (nos muestra todos los alumnos de la base de datos en un tabla o nos indica que no hay alumnos con un mensaje).

Hemos de destacar el hecho de que, en este sistema, se ha corregido el proceso que se realizaba para buscar, mostrando todos los elementos existentes en una búsqueda, y no solo el primero.

3.2.3.4. Comunicación

Aunque en este caso también deberemos cambiar la configuración de las cabeceras, como durante la instalación no se ha realizado ningún Virtualhost, se modificará directamente el archivo *default* de Apache (*/etc/Apache2/sites-available/default*), del mismo modo que en el sistema anterior (introduciendo *Header set...*).

Otro cambio a tener en cuenta para este sistema es la necesidad de un interprete que convierta nuestro objeto Java a un String JSON. Existen ya muchas librerías para esto, pero dada su sencilla utilización, emplearemos **google-gson**, que añadiremos a nuestro proyecto Eclipse del mismo modo que el JDBC (*Built path...-Add external jar [GSO]*).

Para conseguir el mensaje deseado en JSON y por tratarse de Java, hemos tenido que crear dos clases auxiliares, la clase Alumno y la clase Mensaje.

Dado que en Java no podemos utilizar un mismo nombre para variables de distinto tipo, los mensajes, aunque mantendrán la misma estructura, con un primer campo **tipo** (0 ó 1) y un segundo campo, que en este caso tendrá un nombre variable asociado al tipo (tipo 0: mensaje y tipo 1: list o mensaje), pero que tendrá la misma función que en el Sistema1.

Un ejemplo de trama tipo 1:

```
{ "tipo":1,"list":[{"nombre":"Shere","apellido":"Fraile","edad":28,"correo":"crazy@gmail.com","id_carrera":1},{ "nombre":"Luis","apellido":"Martin","edad":30,"correo":"cerves","id_carrera":1}]}
```

Otro ejemplo de trama tipo 1:

```
{ "tipo":1,"mensaje":"El alumno ha sido correctamente introducido"}
```

3.2.4. Sistema Completo

En líneas generales, el sistema se comporta como en el ejemplo anterior. Más concretamente, para el usuario no habrá diferencias.

Básicamente, la diferencia reside en los métodos a llamar cuando se requiere una acción desde el navegador, que ya comentamos anteriormente.

3.2.5. Esquema Completo

Aunque el comportamiento es en cierto modo idéntico, el esquema difiere tal y como se muestra en la figura 3.10.

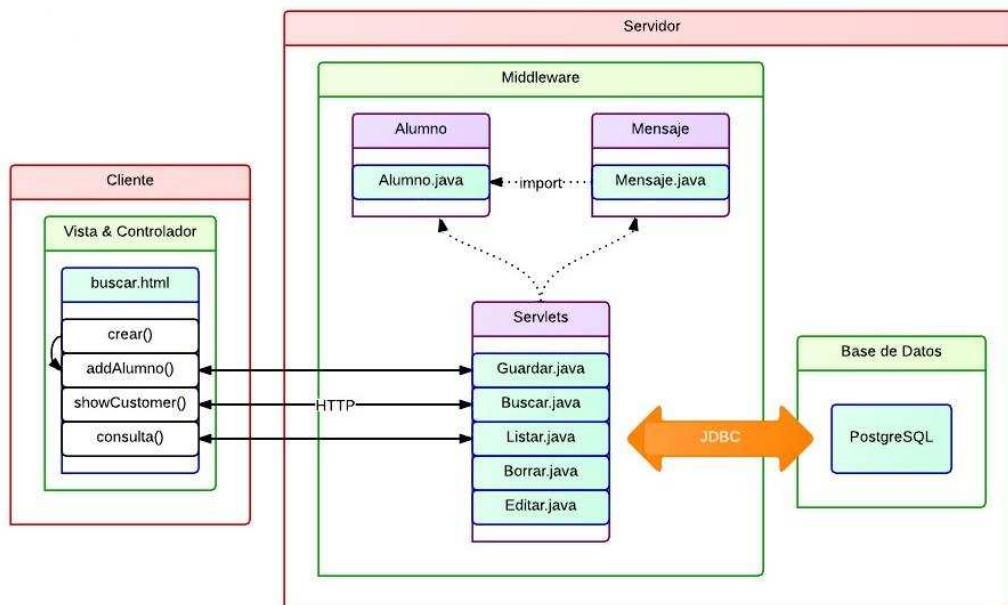


Figura 3.10: Esquema Arquitectura completa Sistema2

3.2.6. Mejoras

Como en el caso anterior, desde la parte del cliente sólo podemos verificar el correcto comportamiento de tres de las cinco acciones, aunque Borrar y Editar están implementadas y pueden ser verificadas mediante la llamada `localhost:8080/estudio/Borrar?nombre=Shere` y `local-`

host:8080/estudio/Editar?nombre=Edad&apellido=Apellido&edad=27&correo=correo.
Por ello, como en el Sistema1, se debería acabar esta parte.

En este sistema, al igual que en el anterior, las acciones de Editar y Borrar, se ha planteado de modo que pueden llevarnos a equivoco, puesto que la búsqueda se realiza por nombre en lugar de por *id*.

Aunque en el otro caso no fue comentado, la aplicación de un estilo CSS sería necesario en ambos sistemas.

A pesar de todo ello, como en el sistema anterior, cumple las funciones de aprendizaje y análisis deseadas.

3.3. Comparación

Realizaremos una comparación de ambos sistemas para determinar cuál emplearemos finalmente. En ella se pondrá una valoración personal basada en las experiencias adquiridas durante las pruebas, por lo que se tratará de una comparación subjetiva, que podría haber sido distinta, ya que como en ocasiones se ha mencionado, la aplicación de las tecnologías depende del caso concreto a implementar.

	Sistema1	Sistema2	Comentarios
Instalación	✓	✓✓	En el Sistema1, la dificultad ha residido en la configuración de algunos aspectos de Symfony, mientras que en el Ejemplo2, la máxima dificultad se ha encontrado en buscar el paquete (Java Development Kit) Conjunto de Desarrollo de Java (JDK), eliminado por Oracle del repositorio.
Desarrollo	✓	✓✓	En este caso, ha resultado más fácil el Sistema2, pero esto puede ser debido al conocimiento previo de Java y al hecho de haber realizado con anterioridad el Sistema1. Además, el empleo de Eclipse ha facilitado la depuración del código.

Tabla 3.1: Comparacion Sistemas Estudio Práctico

Después de todo este análisis, no hemos podido obtener ninguna conclusión sobre si sería mejor el empleo de un SGBDR MySQL frente a PostgreSQL, por lo que para nuestra aplicación podrá usarse indistintamente. Eso sí, según nuestra experiencia, el mejor sistema

sería el basado en *servlet*, ya que posee todo el potencial de Java (como por ejemplo la seguridad) y permite integrar fácilmente sistemas ya desarrollados.

Descripción del proyecto

En este capítulo nos centraremos en los requisitos y funciones a implementar en el proyecto.

Antes de meternos de lleno con el desarrollo es una buena práctica estudiar cuáles son los requisitos que tenemos, para adaptar de esta forma lo máximo posible nuestra aplicación a lo que el usuario espera encontrarse.

4.1. Casos de usos

Los casos de uso proporcionan los escenarios mediante los cuales definimos como interactuará el sistema con los diferentes usuarios, logrando de esa forma un objetivo específico.

Para comenzar, identificaremos los distintos actores que vamos a encontrar interviniendo en nuestro software:

- El administrador podrá comprobar la funcionalidad del sistema y además, modificar los contenidos que ofrece la aplicación. Es el encargado de gestionar posible errores e incorporar o modificar funcionalidades. Solamente él puede introducir a los profesores en el sistema (aquel con permisos de lectura y escritura incondicional).
- El profesor gestionará sus grupos, es decir, podrá crear/modificar/eliminar sus propios grupos de alumnos (permisos de lectura y escritura condicionada).
- El alumno sólo podrá visualizar los datos (permisos únicos de lectura, aunque podrá cambiar su contraseña).

Como consecuencia de estas características se crearán dos vistas: una para el uso exclusivo del profesor, donde poder insertar, modificar y visualizar información; y otra a la que sólo tendrá acceso el alumno.

4.2. Requisitos

En este apartado explicaremos las funciones que debe cumplir la aplicación de manera más específica.

4.2.1. Acciones Permitidas

Generales

1. Acceso sólo a aquellos usuarios registrados.
2. Al administrador, la inserción en bloque de profesores (requisito específico).
3. Visualizar los datos almacenados modificables del profesor (nombre, apellido y correo electrónico). Debe permitirlo en ambas vistas, aunque en cada una se realizará de un modo diferente.
4. La modificación de la contraseña al usuario.
5. Posibilitar la desconexión del usuario en cualquier momento.
6. La gestión por varios profesores de un mismo grupo de alumnos de una asignatura.
7. Recuperar la contraseña mediante el envío de un correo electrónico.
8. Desconexión automática en caso de 30 minutos de inactividad.

Profesor

9. Modificación de sus datos visibles (nombre, apellido y correo electrónico).
10. La creación de nuevas asignaturas.
11. La creación de nuevos grupos de alumnos dentro de una asignatura.
12. Visualización de los grupos de alumnos sobre los que tiene capacidad de gestión.
13. Visualizar todos los alumnos que componen un grupo de una asignatura.
14. Incorporar o eliminar alumnos de una asignatura.
15. El visionado de las notas de todos los exámenes de un determinado alumno para una asignatura.
16. Introducir las notas en bloque de un determinado examen.
17. Visualizar la nota de un examen de todos los alumnos que corresponden al grupo desde que se hace la petición.
18. Modificar las notas de los exámenes de los alumnos.

19. Si el examen a visualizar contiene apartados, deberá mostrar la información correspondiente a dichos apartados (nombre, puntuación máxima, materia y web) y permitir su modificación. Requisito válido únicamente cuando se visualiza la nota del examen en grupo.
20. Personalizar el cálculo de la nota de la evaluación.
21. Visualizar y modificar la planificación de la asignatura (cantidad de exámenes, nombre, tipología, puntuación máxima...).
22. Realizar una copia de seguridad con información almacenada en la aplicación para un grupo de una asignatura, para el momento actual, mediante la descarga de un archivo *csv*. Es decir, una copia de seguridad con la información correspondiente a la notas de los exámenes y de la evaluación de cada alumno del grupo.
23. Visionado para todo el grupo de la nota de evaluación instantánea (la nota que tendría si el alumno continuará como hasta ahora) y final.
24. Visionar el método de evaluación.
25. Eliminar grupos/asignaturas que sólo el gestiona.

Alumno

26. Visualizar las distintas asignaturas en las que se encuentra.
27. Visualizar para cada asignatura la información de contacto correspondiente a los profesores que gestionan dicha asignatura (nombre, apellidos y correo electrónico).
28. Visualizar la información de los distintos exámenes de la asignatura (nombre, tipología..., incluyendo si es necesario la de los apartados) con la nota obtenida por el alumno.
29. Visualizar la nota de la evaluación instantánea y final para cada asignatura.
30. Visualizar la posición en el ranking de la asignatura según la evaluación.

4.2.2. Acciones NO Permitidas**Generales**

31. Acceso a los alumnos, a la vista de los profesores, ni a la inversa.
32. Acceso a las páginas que contengan el menú de opciones, hasta que el usuario no haya cambiado su contraseña inicial.
33. A un alumno pertenecer a varios grupos de una misma asignatura.

4.3. Diseño

Dado que emplearemos una arquitectura Cliente-Servidor basada en el patrón Modelo-Vista-Controlador, diseñar nuestro proyecto principalmente consistirá en la creación de una base de datos que, con este propósito y el de cumplir con las necesidades del sistema, dé soporte a las funcionalidades de un modo adecuado. En nuestro caso, se tratará de la creación de un conjunto de tablas donde almacenar la información, que se categorizará en dos grupos: uno con la información correspondiente al profesor y otro, con la del alumno, tal y como se muestra en la figura 4.5.

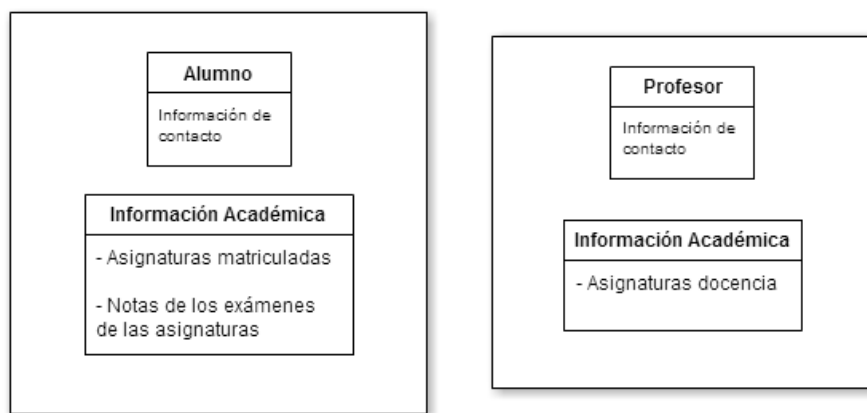


Figura 4.1: Subdivisión Base de datos

De este modo, nuestros casos de uso, en función de la acción a realizar en la base de datos, se pueden resumir tal y como se muestra en las figuras 4.2, 4.3 y 4.4.



Figura 4.2: Lectura información



Figura 4.3: Inserción información

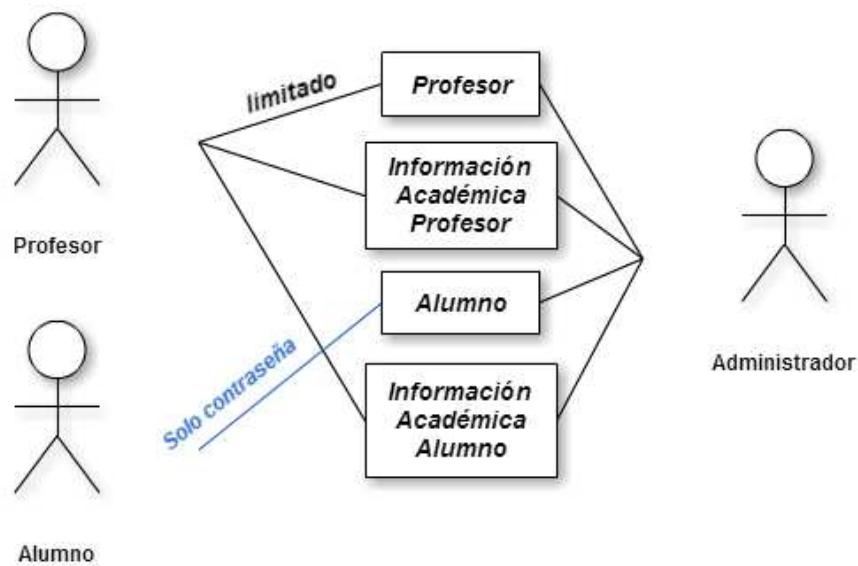


Figura 4.4: Modificar información

4.4. Resumen

Nuestro proyecto es una aplicación, en la cual los usuarios (profesores o alumnos), desde la parte Cliente de la aplicación (una página web), realizarán una petición para un determinado servicio (función) al Servidor, que lo realizará y devolverá los resultados como respuesta.

Dicho funcionamiento del sistema podría resumirse con el diagrama de la figura 4.5.

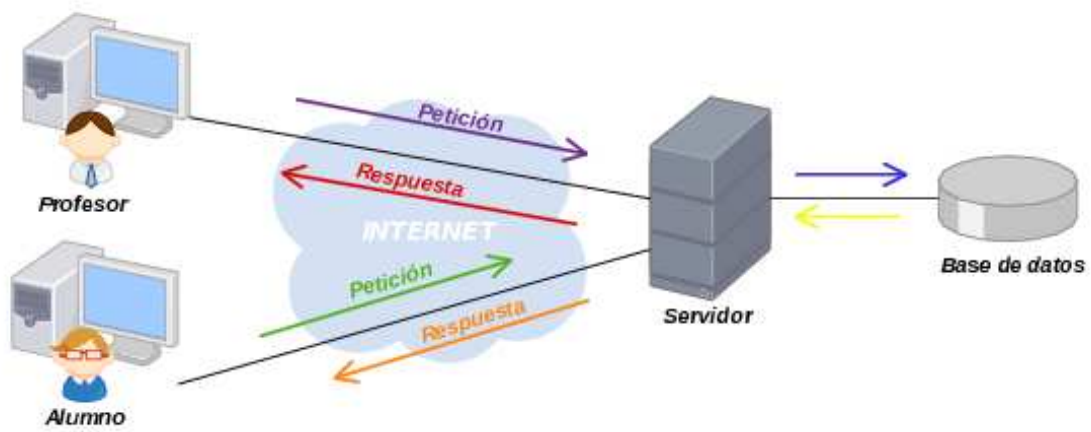


Figura 4.5: Cliente-Servidor

En el siguiente capítulo se especifican y detallan todos los pasos que se han realizado para conseguir que la aplicación sea totalmente funcional.

A lo largo de este capítulo mostraremos los pasos que hemos seguido para la realización del software. Para ello explicaremos las decisiones de diseño que se han tomado.

5.1. Tecnologías

Los lenguajes de programación y herramientas usadas para el desarrollo han sido elegidas, teniendo en cuenta el estudio realizado en el Capítulo 3, concretamente de las experiencias obtenidas tras los dos ejemplos.

Tras el análisis, obtuvimos que el mejor sistema en la parte del Servidor para nuestro propósito era el que consistía en uno creado por la conjunción de un sistema operativo **Linux**, un software para Servidor Web **Apache-Tomcat**, un lenguaje de programación **Java-Servlet** (para ampliar las capacidades de respuesta del servidor) y, donde la única duda no despejada residía, en el gestor de base de datos a usar MySQL o PostgreSQL. En nuestro caso le hemos dado mayor importancia a la capacidad y la rapidez, frente a otras características, por lo que usaremos **MySQL**.

En la parte del Cliente y la Comunicación Cliente-Servidor la solución fue concluyente sólo con el estudio: **JSON** y **AJAX**.

Como entorno de desarrollo para la programación, aunque se disponían de otras alternativas, la herramienta más recomendable ha sido el IDE Eclipse por su popularidad, facilidad de uso y compatibilidad.

5.2. Modelo de datos

Una parte muy importante es el almacenamiento de la información en el servidor, para futuras consultas o modificaciones, labor realizada por la base de datos implementada. Con el modelo de datos básicamente describimos la estructura lógica de los datos, es decir, determinamos de qué modo organizamos, almacenamos y manipulamos los datos. Por ello que se trata del pilar fundamental del sistema.

Conocidos los usuarios, necesidades y requerimientos del sistema, se dispondrá de una sola base de datos general cuyo diseño es el modelo que se ilustra en la figura 5.1.

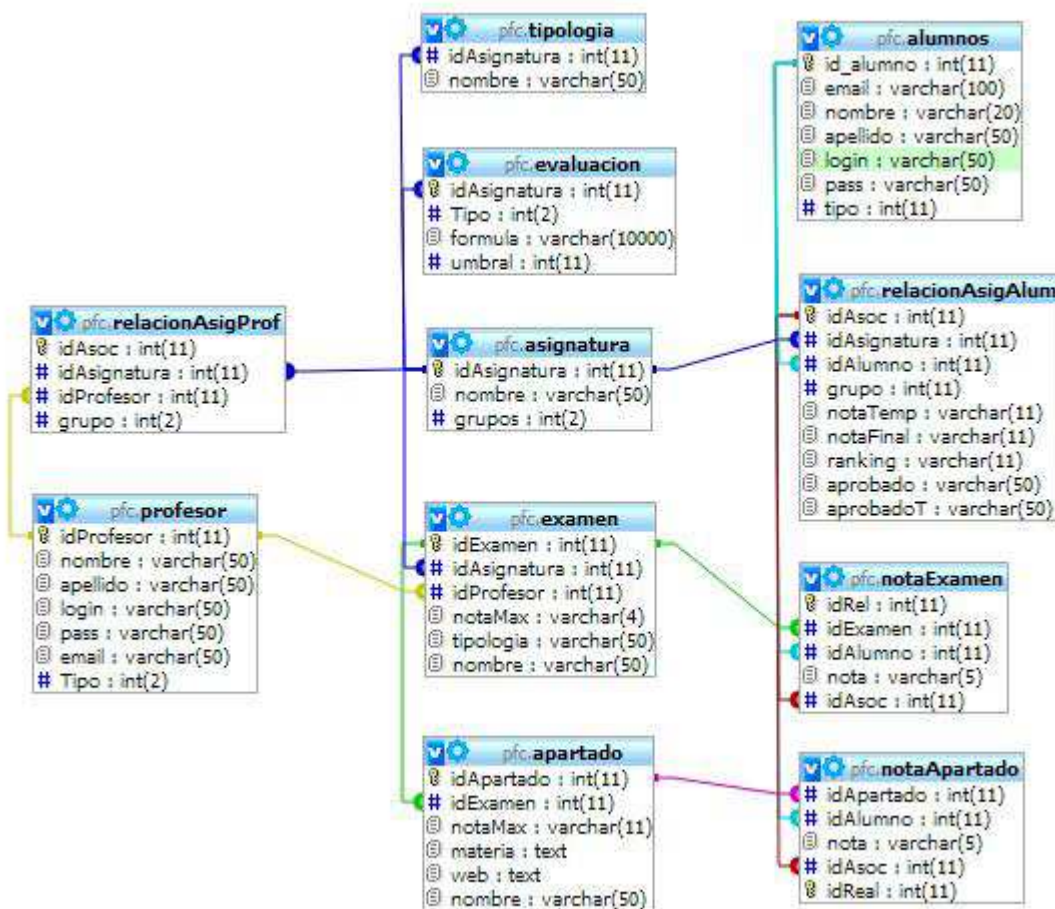


Figura 5.1: Esquema Modelo de datos

En el proyecto actual existen 11 tablas diferentes, como puede apreciarse en la 5.1, con relaciones entre ellas.

A continuación realizaremos un análisis detallado de las distintas tablas, justificando además su utilización desde el punto de vista de los casos de uso.

Comenzaremos hablando de las tablas de los distintos usuarios de nuestro sistema. Aunque hay tres tipos de usuarios, encontramos sólo las tablas de alumnos (**Tabla alumnos**) y profesores (**Tabla profesor**), debido a que el administrador del sistema será tratado como un caso particular de los profesores (el administrador será el primer elemento de la tabla profesores).

Tabla profesor: en esta tabla se almacenan todos los profesores que tienen acceso al sistema. Como elemento inequívoco ante búsquedas tendremos el *login*, que será igual al *email*, y que junto con el *pass* nos dará acceso al sistema. Los parámetros modificables

serán *nombre*, *apellido* y *email* (pero el login no cambiará). La clave principal es el *id-Profesor*, también inequívoca, y será el elemento que se relacionará con otras tablas. El parámetro *Tipo* sirve para identificar si un profesor ha cambiado, o no, su contraseña inicial.

Tabla alumnos: en ella se almacena la información personal correspondientes a los alumnos que tienen acceso al sistema, ya que pertenecen o han pertenecido a una o varias asignaturas. El parámetro *Tipo* sirve, como en el caso del profesor, para identificar si el alumno ha cambiado, o no, su contraseña inicial. Los elementos de *login* y *pass* nos darán acceso al sistema (en este caso el elemento *login* se corresponderá con el login de Aula Global, es decir, con un conjunto de nueve números). La clave es el *id_alumno*, que es el elemento que nos relaciona con otras tablas.

Para justificar la relación y existencia del resto de las tablas, continuaremos la explicación desde el punto de vista de las acciones que el usuario sigue. En este caso, el usuario principal es el profesor: el profesor entra en el sistema vacío y la primera acción a realizar es crear una asignatura.

Con todo esto, la siguiente tabla sobre la que hablaremos es la **Tabla asignatura**. Su labor es guardar todas las asignaturas que se han creado. El parámetro *grupos* es un contador de la cantidad de grupos que hay creados sobre esa asignatura. Hasta que no se cree un grupo, la asignatura tendrá 0 grupos. Si, por ejemplo, su valor fuese el 2 no significaría que los grupos existentes son el 1 y el 2, podrían ser el 2 y el 3, puesto que el grupo 1 podría haber sido borrado (si se creara un grupo nuevo sería el 4, pero en *grupos* pondría 3). El elemento inequívoco ante búsquedas es el *nombre*, ya que, basándonos en las asignaturas realizadas en la Universidad Carlos III, éstas poseen un nombre compuesto por una sucesión de números y letras que no dan posibilidad de error. La clave es el *idAsignatura*, que es el elemento que nos relaciona con otras tablas.

Pero crear una asignatura no sólo consiste en introducir los parámetros anteriores, puesto que una asignatura también posee una planificación, es decir, un conjunto de exámenes ya planificados.

La información correspondiente a estos exámenes *nombre*, *tipología*, *notaMax* (nota máxima) es almacenada en **Tabla examen**. Esta tabla posee varios elementos relacionales: *idAsignatura*, para designar a qué asignatura pertenece este examen; *idProfesor*, que nos indique qué profesor ha puesto este examen (aunque podrán ver la información todos aquellos profesor que tengan la asignatura); e *idExamen* que es la clave y la relación a otras tablas que aún no hemos explicado. De esta tabla debemos destacar algo que a simple vista pasa desapercibido: la tipología que corresponda con cada examen debe existir antes en la **Tabla tipología**, cuya única función es almacenar los distintos valores que podrá contener el elemento *tipologia* de un examen.

A su vez, un examen puede contener apartados. De ser así, las características *nombre*, *notaMax*, *materia*, *web* de dichos apartados serán introducidos en la **Tabla apartados**. El elemento *idExamen* nos relaciona con la tabla anterior, indicando así qué apartados corresponden a un examen. La clave es *idApartado*, que además es el nuevo elemento

relacional creado.

El último paso para que una asignatura tenga toda su información completa consiste en tener en cuenta que toda asignatura posee un método de evaluación, y es ahí donde reside la importancia de la **Tabla evaluacion**. El elemento clave es aquel que nos indica a qué asignatura corresponde la evaluación: *idAsignatura* (ya que sólo podemos encontrar una evaluación por asignatura). El parámetro *Tipo* nos indica de qué tipo de evaluación se trata: por defecto (media de todos los exámenes, indistintamente de la tipología) o personalizada, en cuyo caso, los elementos *umbral* y *formula* tendrán cabida, ya que nos determinarán qué fórmula debemos seguir para calcular la nota y si eso se traduce en un aprobado.

Una vez creada la asignatura, se debe crear una relación donde se indique que un determinado profesor (*idProfesor*) tiene acceso a un determinado grupo (*grupo*) de una asignatura (*idAsignatura*). Para reflejar esta asociación encontramos la **Tabla relacionAsigProf**, donde el elemento clave *idAsoc* ha sido creado para la correcta actualización de la tabla.

NOTA: Un grupo de una asignatura podrá estar asociada a varios profesores.

Llegados a este punto, en el cual se ha finalizado la creación de una asignatura, el siguiente paso a seguir (de nuevo suponiendo el sistema vacío) es crear un grupo para esta asignatura, es decir, se introducirán los alumnos en la base de datos (**Tabla alumnos**), junto con una relación, donde indiquemos que un determinado alumno (*idalumno*) pertenece a un grupo (*grupo*) de una asignatura (*idAsignatura*). Los parámetros *notaTemp-aprobadoT* y *notaTemp-aprobadoF* son los que se corresponden con la nota de evaluación, tanto en progresión como final, que tendrá el alumno en esa asignatura, y son los que determinarán la posición en el ranking (*ranking*) de la asignatura. El elemento clave *idAsoc* es el que vincula esta relación en otras tablas.

Finalmente, las tablas **Tabla notaExamen** y **Tabla notaApartado** son aquellas en las que, como su propio nombre indica, se insertan las notas (*nota*) de los exámenes/apartados (*idExamen* o *idApartado*, único elemento diferente) del examen de cada alumno (*idalumno+idAsoc*, que referencia la relación asignatura-alumno explicada anteriormente). De estas tablas cabe destacar el hecho de que podría chocarnos que *nota* sea una variable de texto en vez de numérica. Esto es debido a que cuando un alumno no se presenta a un examen, su nota se establece como NP, que no sería aceptado por una variable numérica. Como en otras tablas, el elemento clave *idReal* ha sido creado para la correcta modificación de la tabla.

5.3. Cliente

En este apartado se explicarán las funcionalidades asociadas al cliente, sin entrar en el servidor.

Se trata de un conjunto de páginas HTML, todas con la misma estructura, donde la diferencia reside en el archivo JavaScript al que llamamos, tal y como se muestra en el siguiente ejemplo (*menu.html*).

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta httpequiv="ContentType" content="text/html; charset=ISO
   -8859-1">
5 <title>Inicio</title>
6 <script type="text/JavaScript" src="/js/funcionMenu.js">
7 </script>
8 </head>
9 <body>
10 <form >
11 <fieldset id="fiel" style="border:0">
12 <script type="text/JavaScript">consulta()</script>
13 </fieldset>
14 </form>
15 </body>
16 </html>
```

Básicamente, se trata de realizar peticiones al servidor y analizar la respuesta que éste nos envía.

*NOTA: en todo las funciones, salvo en las que se indique explícitamente, el sistema elegido para el control de sesiones ha sido el análisis de cookies. Es decir, si existe una cookie **id=**, hay un usuario autenticado.*

5.3.1. Sistema de acceso

Dado que se trata de un sistema de acceso restringido, para comenzar, necesitaremos autenticarnos en el sistema.

Para ello, se ha creado la página HTML denominada *acceso.html*.

Inicialmente, la página realiza una comprobación para verificar si ya existe una sesión iniciada. En caso afirmativo, se informará al usuario de ello, dándole además la opción de *Desconectar* o *Ir a menu* (dependiendo del usuario nos redirigirá a su página principal). Si no se ha encontrado una sesión, se mostrará un formulario para introducir la información de acceso que enviaremos al servidor.

Dependiendo de la respuesta recibida del servidor, tendremos distintas acciones:

- Usuario no Autorizado

En la misma página se nos mostrará un aviso de que dicho usuario no existe en nuestro sistema.

- Administrador

En caso de ser el administrador el que se autentifique en el sistema, se le redirigirá automáticamente a la página *guardar.html*. Esto es debido a que esta página

ha sido creada para uso exclusivo del administrador, ya que el resto de funciones pueden ser desempeñadas por el administrador sin necesidad de iniciar sesión en el sistema (PhpMyAdmin...).

- Profesor o alumno Autorizado pero con Contraseña Inicial

Uno de los requisitos del sistema consiste en que, antes de poder emplear el sistema normalmente, debe ser cambiada la contraseña que asigna el sistema automáticamente. Por ello, el usuario será redirigido a la página *cambiarPassword.html*.

- Profesor Autorizado

El usuario será redirigido a la página *inicio.html*.

- Alumno Autorizado

El usuario será redirigido a la página *menu.html*.

Para una mejor comprensión del proceso se añade el esquema de la figura 5.2.

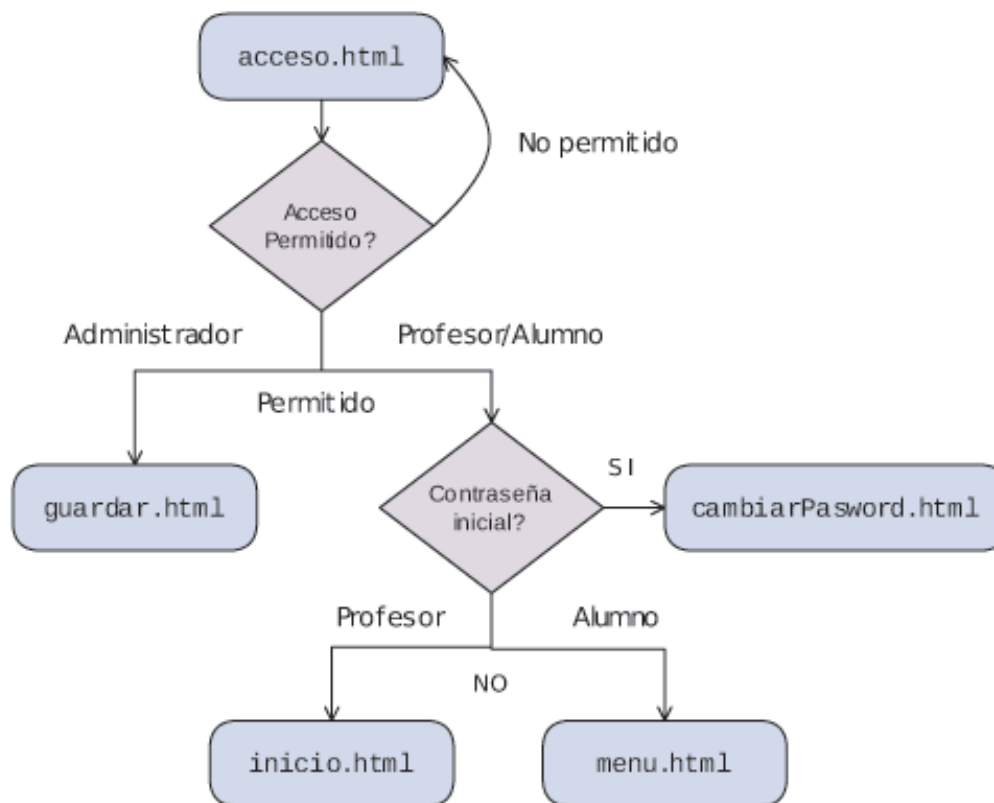


Figura 5.2: Sistema acceso

Si no se posee o no se recuerda la contraseña para acceder al sistema, en la parte inferior de la página se encuentra un enlace al sistema de recuperación de contraseñas.

Para finalizar el análisis, describiremos los distintos métodos que componen el archivo JavaScript, que implementa las acciones que hacen posible el comportamiento anteriormente comentado (*funcionAcceso.js*).

- **conectado**: función que comprueba si existe una sesión activa en el sistema.
- **crear**: función que crea dinámicamente el formulario a rellenar para introducir los datos de acceso. Al pulsar *Acceder*, llamaremos a la función **comprueba**.
- **comprueba**: función que verifica que el formulario no tenga ningún campo vacío, en cuyo caso invocará al método **consulta**.
- **consulta**: función que hace una petición a *comprobaciónAcceso* y nos muestra la respuesta (acceso denegado o redirección).
- **desconectar**: función que finaliza la sesión del usuario.

Como apoyo a esta descripción se incluye la figura 5.3.

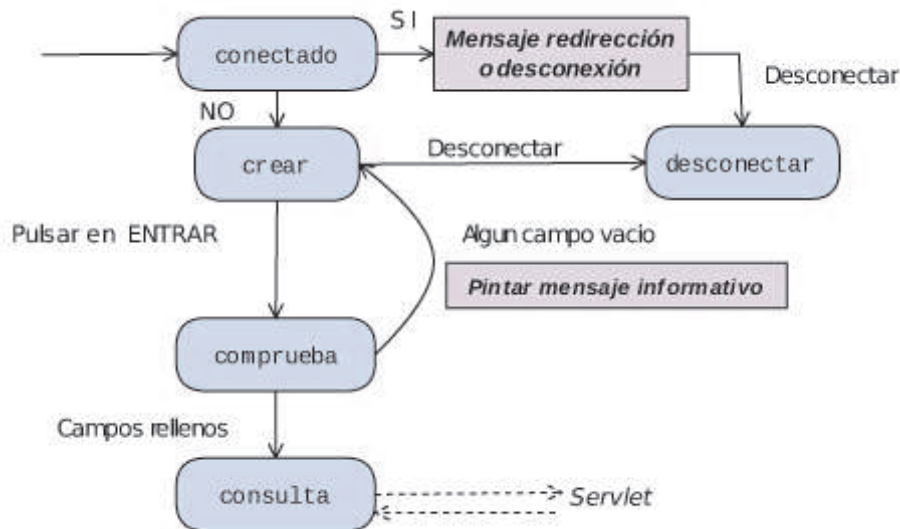


Figura 5.3: Flujo métodos página acceso

5.3.2. Sistema de recuperación de contraseñas

Para la adquisición de la contraseña, ya sea porque no se haya accedido nunca al sistema y no se sepa, o por el hecho de que se haya olvidado, se ha creado un sistema que enviará la contraseña por correo electrónico.

Este sistema se implementa en la página *recuperacion.html*. En ella se pide al usuario que introduzca el correo electrónico de la universidad, ya que este elemento puede ser comparado con datos almacenados, para el caso del profesor, con el login, y, para el caso del alumno, con la información adicional *email*.

Será a este correo electrónico, al que se envíe la información.

Para realizar esta función, el archivo *funcionRecuperar.js* se compone de los métodos que se describen a continuación:

- **consulta**: función que crea dinámicamente el formulario a rellenar para introducir los datos de acceso. Al pulsar *Enviar*, llamaremos a la función **compruebaE**.
- **compruebaE**: función que verifica que el formulario no tenga ningún campo vacío, en cuyo caso invocará al método **enviarMail**.
- **enviarMail**: función que hace una petición a *enviarEmail* y nos muestra la respuesta (éxito o fracaso de la acción).

5.3.3. Sistema de inserción de profesores

En nuestra aplicación el administrador tendrá dos modos de insertar profesores en el sistema: la manual, es decir, uno a uno en la base de datos, o la automática, mediante la página diseñada para este fin *guardar.html*.

Como se ha indicado anteriormente, a esta página sólo tendrá acceso el administrador (al resto no se permite el acceso, mostrando un mensaje de rechazo). Su única función es, mediante un formulario, dado un archivo *csv*, introducir los datos de los profesores en el sistema de forma masiva.

Dicho archivo *csv* deberá poseer un formato explícito, ya que si no se producirían errores al cargar los datos. Sobre los distintos tipos de *csv* y su formato, hablaremos en la sección 5.5 (Archivos).

Por defecto, el sistema ha sido diseñado para que al introducir los datos del profesor, el correo electrónico sea el login de acceso, de modo que evitaremos incompatibilidades o errores por el uso del mismo login. La contraseña inicial será generada aleatoriamente, incorporando números, letras y símbolos.

Con esta finalidad, el archivo *funcionGuardar.js* se compone de los métodos que se exponen a continuación:

- **conectado**: función que comprueba que el usuario que intenta acceder a la página es el administrador. Si él es administrador se invoca a **crear**, si no a **noConectado**
- **crear**: función que crea dinámicamente el formulario a rellenar para introducir los datos de acceso. Al pulsar *Subir*, llamaremos a la función **compS**.
- **compS**: función que comprueba que el formulario no tenga ningún campo vacío, en cuyo caso, invocará al método **consulta**. En caso contrario, llamará al metodo **crear**.
- **consulta**: función que hace una petición a *GuardarArchivo* y nos muestra la respuesta (éxito o fracaso de la acción).
- **noConectado**: función que muestra un mensaje por pantalla para indicar que no se tiene acceso a la página.

Como complemento a esta descripción se incluye la siguiente ilustración (figura 5.4).

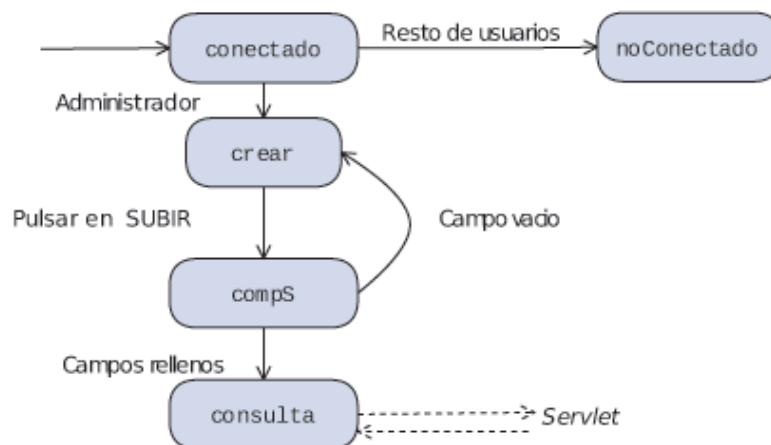


Figura 5.4: Flujo métodos página guardar

5.3.4. Cambio de contraseña

Se ha creado una página *HTML* específica con este fin: *cambiarPassword.html*. En ella, mediante un formulario, se pide información necesaria del usuario para poder proceder a cambiar la contraseña. Dado que se requiere la contraseña antigua, no hace falta estar autenticado en el sistema para poder cambiar la contraseña. Cuando no se está autenticado y la operación se realiza con éxito, la opción de volver nos guiará hasta *acceso.html*, mientras que si se está autenticado, nos retornará a *inicio.html*, ya que seguramente vengamos de la opción de *Cambiar Contraseña* que nos ofrece el menú.

Durante la operación de cambio de contraseña en el servidor también se cambia la información para indicar que la acción que permite pasar al menú ha sido realizada.

Con esta finalidad, se han creado los métodos que componen el archivo *funcionGuardar.js*, que se exponen a continuación:

- **crear:** función que crea dinámicamente el formulario a rellenar para introducir los datos de acceso. Al pulsar *Submit*, llamaremos a la función **compruebaE**.
- **compruebaE:** función que comprueba que en el formulario, la nueva contraseña introducida, tenga la longitud y la combinación de caracteres adecuados, y que, además, no exista ningún campo vacío, en cuyo caso se invocará al método **consulta**.
- **consulta:** función que hace una petición a *cambiarPass* y nos muestra la respuesta (éxito o fracaso de la acción).
- **compruebaId:** subfunción que nos indica si el usuario que accede a la página ha iniciado sesión antes de acceder a ella, para que en caso de pulsar *Submit*, seamos redirigidos a la página correcta.

Como complemento a esta descripción se incluye la siguiente ilustración (figura 5.5).

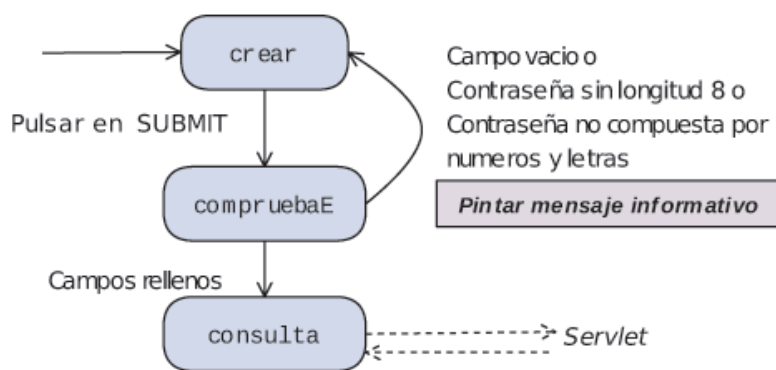


Figura 5.5: Flujo métodos página cambiarPassword

NOTA: La nueva contraseña deberá contener ocho caracteres, con números y letras.

5.3.5. Funcionalidades Profesor

Todas las acciones que pueden ser realizadas por un profesor en el sistema se encuentran en la página *inicio.html* (página principal para los profesores).

Por su propia definición, sólo los profesores podrán acceder a ella. Si se intenta acceder sin encontrarse autenticado en el sistema, el usuario será redirigido hacia la página de acceso (*acceso.html*). Si se intenta acceder directamente a esta página antes de haber cambiado la contraseña, se impedirá el acceso y se mostrará un mensaje de error, indicándonos que debemos cambiar la contraseña primero.

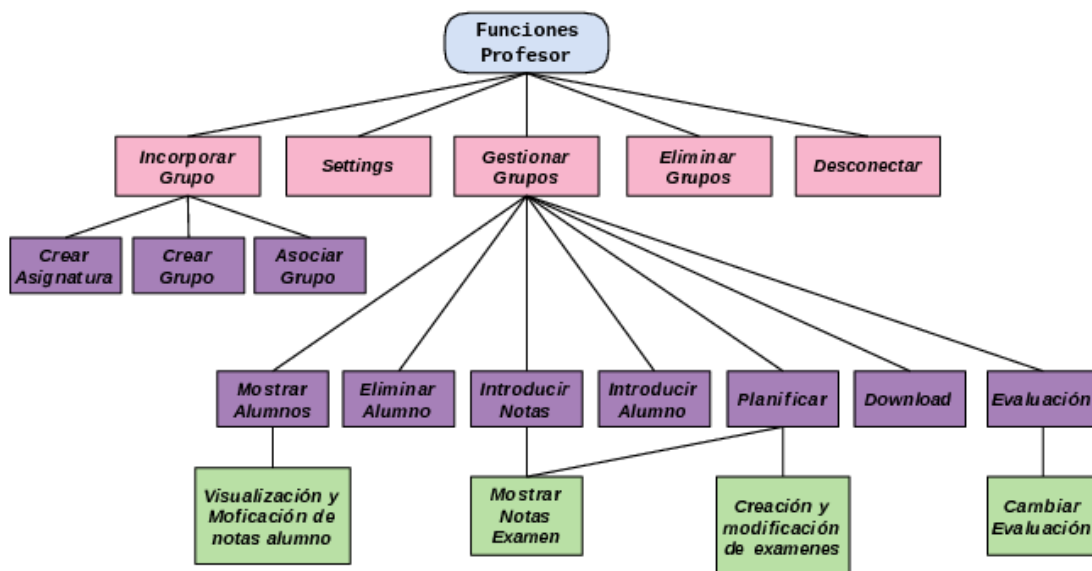


Figura 5.6: Funciones profesor

La web consta de una primera parte compuesta por un menú con las cinco funciones principales: *Setting*, *Incorporar nuevo grupo de alumnos*, *Gestionar grupos*, *Borrar Gru-*

po/Asignatura y *Desconectar*; y una segunda parte donde se desarrollara el resultado de clicar en alguna de la opciones anteriores (ver figura 5.6).

*NOTA: En este caso, debido al alto tamaño de métodos que componen el archivo **funcionInicio.js**, no se entrará en la misma categoría de detalle que la vista hasta ahora, ya que, salvo alguna excepción, básicamente se trata de una sucesión de peticiones en cadena.*

NOTA2: La página posee un sistema propio de verificación de acceso, para permitir únicamente el acceso a los profesores, que será obviado en esta sección por tratarse de un proceso oculto al usuario final.

5.3.5.1. Desconectar

Como se ha dicho anteriormente, el control de sesiones, se hará mediante la incorporación de una *cookie*. Ésta tendrá un tiempo de vida de 30 minutos, tras los cuales expirará automáticamente.

Si para el uso del sistema se requiere más de ese tiempo, el usuario se vería obligado a iniciar sesión de nuevo. Para evitar esta incomodidad, en cada acción que se realice, esa *cookie* es actualizada, sobrescribiéndola, de modo que desde el instante de actualización disponemos de 30 minutos más de uso.

De todos modos, si en cualquier momento el usuario quiere abandonar la sesión, simplemente, pulsando esta opción, borrará la *cookie*, de modo que, para nuestro sistema, el usuario estará desconectado.

*NOTA: Esta funcionalidad se corresponde con el método **desconectar** anteriormente comentado.*

5.3.5.2. Settings

El propósito de esta opción consiste en permitir al profesor modificar la información que existe sobre él y mantenerla actualizada, puesto que estos datos son mostrados al alumno como información de la asignatura.

Básicamente, al activar esta opción, se nos muestra un formulario editable con la información modificable del profesor que existe en el servidor (nombre, apellidos y correo electrónico).

En este formulario existe también una opción para cambiar la contraseña. Si pulsamos en esta opción seremos redirigidos a la página creada para el cambio de contraseñas (*cambiarPassword.html*) anteriormente comentada.

5.3.5.3. Incorporar nuevo grupo de alumnos

Cuando un profesor entra por primera vez en el sistema no posee ninguna asignatura/-grupo de alumnos, debiendo ser él quien incorpore dicha información. Para este propósito, ha sido creada esta parte del menú.

Al entrar en esta opción, se muestra la opción de crear una nueva asignatura y, si ya se ha creado con anterioridad alguna asignatura, la lista con las asignaturas que el profesor tiene asociada, contenga o no ya grupos creados.

Para crear una nueva asignatura, mediante un formulario, se nos pedirá el nombre y la información de la planificación correspondiente, la cual será heredada por todos los grupos, al igual que la evaluación. Pero esta opción no sólo crea, sino que antes realiza un función de búsqueda para verificar que la asignatura no haya sido creada, en cuyo caso, descartará la planificación y procederá a la asignación, y finalizará avisando de dicho suceso (también se comprueba que no la tenga asignada ya). En caso de no darse las situaciones anteriores, la asignatura será creada, pero no contendrá ningún grupo.

Para crear un nuevo grupo, debemos clicar sobre el nombre de la asignatura en cuestión. Una nueva vista surgirá en ella, donde se dá al usuario la oportunidad de crear un nuevo grupo o la posibilidad, en caso de contener ya grupos, de agregar aquellos que no tenga ya asociados.

Crear un nuevo grupo consistirá simplemente en introducir un archivo *csv* con los alumnos pertenecientes (se comprueba duplicidad). El número del grupo vendrá designado por el del último grupo existente para dicha asignatura más uno.

5.3.5.4. Gestionar mi grupos de alumnos

En esta opción se nos muestra el listado de grupos (puede haber más de uno por materia), en los que el profesor imparte docencia.

Hasta que una materia creada, no esté provista de un grupo de alumnos, no aparecerá en esta vista, aunque sí aparece en la vista anteriormente comentada.

Al pulsar sobre uno de los grupos, una nueva vista provista de un submenú aparece. Las funciones básicas de las que dispone y que se aplicarán estrictamente a este grupo son: *Mostrar alumnos*, *Incorporar nuevo alumno*, *Eliminar alumno*, *Introducir notas*, *Planificación*, *Evaluación* y *Download*.

Tanto *Incorporar nuevo alumno* como *Eliminar alumno* son funciones que responden a la necesidad que posee un profesor de cambios en un grupo, ya que los alumnos en la prematriculación son distintos a los que se encuentran en la matrícula definitiva.

Inicialmente tendremos una vista de los alumnos que componen el grupo, el mismo resultado que obtendríamos al pulsar sobre *Mostrar alumnos*.

5.3.5.4.1. Mostrar alumnos

Aparte de mostrarnos los alumnos, tal y como ya hemos mencionado, se da la posibilidad para cada alumno de visualizar y modificar las notas de los exámenes de la asignatura.

5.3.5.4.2. Eliminar alumno

La eliminación de un alumno del grupo se realizará simplemente introduciendo el NIA del alumno, que ya de por sí está definido como elemento único e inequívoco.

5.3.5.4.3. Incorporar nuevo alumno

Básicamente se trata de, mediante un formulario, introducir la información necesaria del alumno para incorporarle al grupo (como en el caso del archivo, se introduce toda la información relativa a éste, ya que será necesaria, en caso de que el alumno aún no esté en la base de datos).

5.3.5.4.4. Introducir notas

Se crea un formulario para la inserción de un archivo *csv* (ver sección 5.5), donde se encuentran las notas de los alumnos de un determinado examen.

Una vez introducidas, se visualizará el grupo con las notas del examen que acabamos de introducir. En esta vista, no se permitirá la edición, siendo necesario ir a la opción de *Mostrar alumnos* para ello. Si el examen contiene apartados, las características generales de éstos sí podrán ser modificadas, mientras que la nota que corresponde al alumno no.

5.3.5.4.5. Evaluación

Inicialmente, por defecto, la evaluación de la asignatura es la nota media de todos los exámenes realizados. En esta vista, se permite la personalización de la evaluación mediante un archivo *csv* (ver sección 5.5) y, además, se representa la nota temporal y final de la asignatura, en función de lo anteriormente seleccionado, para cada uno de los alumnos del grupo.

Al hablar de planificación, se ha hablado de exámenes con tipologías. Esto responde a la necesidad de dar relevancia a ciertos exámenes frente a otros a la hora de calcular la evaluación, y es en la personalización de la evaluación donde esta dinámica coge fuerza. Esencialmente, cuando un profesor se decante por esta opción, estará matizando qué tipologías va a usar para su evaluación y en qué porcentaje, es decir, se indica la relevancia de dicha tipología.

NOTA: En el formulario para el cambio de evaluación se pide la introducción del umbral. Para la incorporación de este campo nos hemos basado en el conocimiento de que en el plan Bolonia no siempre es la media, ya que existen casos en los que una asignatura puede llegar a sumar un total de 110 puntos, pero se aprueba a partir de 50.

5.3.5.4.6. Download

Como sistema para la descarga de una copia de seguridad, en él se implanta esta opción que descarga un fichero con toda la información correspondiente al grupo. Es decir, los alumnos del grupo con cada una de las notas de los exámenes y de la evaluación (los apartados no serán introducidos, solo la nota final).

5.3.5.4.7. Planificación

En esta parte de la aplicación se muestra, gráficamente, la información contenida sobre la planificación de la asignatura, es decir, número, tipología... de los exámenes de la asignatura, posibilitando, además, la modificación de dicha información y la visualización de las notas para un determinado examen (la misma vista que se muestra tras insertar las notas de un examen). Se posibilitará también la creación de nuevos elementos de la tipología.

5.3.5.5. Borrar Grupo/Asignatura

La funcionalidad que aquí se desarrolla es la de permitir al profesor eliminar grupos que haya creado o que tenga asignados, ya sea porque al crearlos hubo algún error o porque se asoció incorrectamente a una asignatura ya creada.

Para realizar dicha función, se muestra un menú multiselección con todos los grupos asociados al profesor (grupo = 0 incluido), permitiendo de este modo borrar más de una asociación a la vez.

Cabe destacar el hecho, de que se puede caer en la equivocación de pensar que cuando un profesor borre un grupo, está borrando el grupo de la base de datos. Esto sólo ocurrirá cuando el profesor que borre el grupo, sea el único asociado a esa asignatura, si no sólo se borrará la asociación asignatura-profesor.

5.3.6. Funcionalidades alumno

Las funcionalidades del alumno implementadas en *menu.html* son mucho más simples que las del profesor (vease figura 5.7).

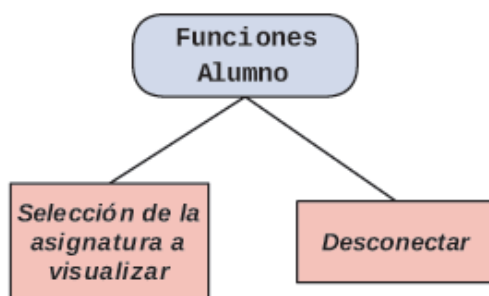


Figura 5.7: Funciones alumno

Básicamente, en esta página se trata de mostrar las asignaturas en las que el alumno está matriculado (siempre que el profesor use este sistema), particularizando para cada una, al clicar sobre ella, la información sobre los profesores que imparten la asignatura, la nota de la asignatura y su evaluación, y el ranking del alumno con respecto al resto. Permitiendo además, en cualquier momento, abandonar el sistema mediante la opción de desconexión (esta acción se realiza del mismo modo que con el profesor).

Con estas funcionalidades, el archivo *funcionMenu.js* se compone de los métodos que se exponen a continuación:

- **consulta:** función que comprueba que exista una sesión activa, para ejecutar **tipoCorrecto**, si no se realizará la operación indicada en **noConectado**.
- **tipoCorrecto:** función que hace una petición a *compruebaTipoaI* y analiza su respuesta. Si no se trata de un alumno que ya haya cambiado su contraseña inicial, nos mostrará un mensaje denegando el acceso y las causas. Si se trata de un , invoca a la función **asigAI**.
- **asigAI:** función que hace una petición a *AsigsAI* y nos muestra la respuesta. Al pulsar sobre una de las asignaturas se realizará la acción determinada por **verDatos**.
- **verDatos:** función que hace una petición a *datosAsig* y nos muestra la respuesta, a la cual se añadirá la respuesta de **verDatos**, **verNotas**, **notasEval** y **ranking**, junto con una opción de volver, que nos llevará de nuevo a la ejecución del método **asigAI**.
- **verNotas:** función que hace una petición a *examAI* y nos muestra la respuesta.
- **notaEval:** función que hace una petición a *notaEvalAsig* y nos muestra la respuesta.
- **ranking:** función que hace una petición a *ranking* y nos muestra la respuesta.
- **noConectado:** función que muestra un mensaje por pantalla para indicar que no se tiene acceso a la página.
- **desconectar:** función que finaliza la sesión del usuario.
- **addCookie:** subfunción que aumenta en 30 minutos el tiempo de vida de la *cookie* de sesión.
- **convertir:** subfunción que transforma los caracteres especiales de las cadenas, para que sean mostrados correctamente desde JavaScript.

*NOTA: Como en el caso del profesor, existe para esta página un sistema de verificación, compuesto por los métodos **consulta** y **tipoCorrecto**, en el cual no entraremos en detalle por ser invisible para el usuario final.*

*NOTA2: Los métodos **consulta**, **tipoCorrecto**, **noConectado**, **desconectar**, **addCookie** y **convertir** son comunes para las funcionalidades del cliente y del servidor.*

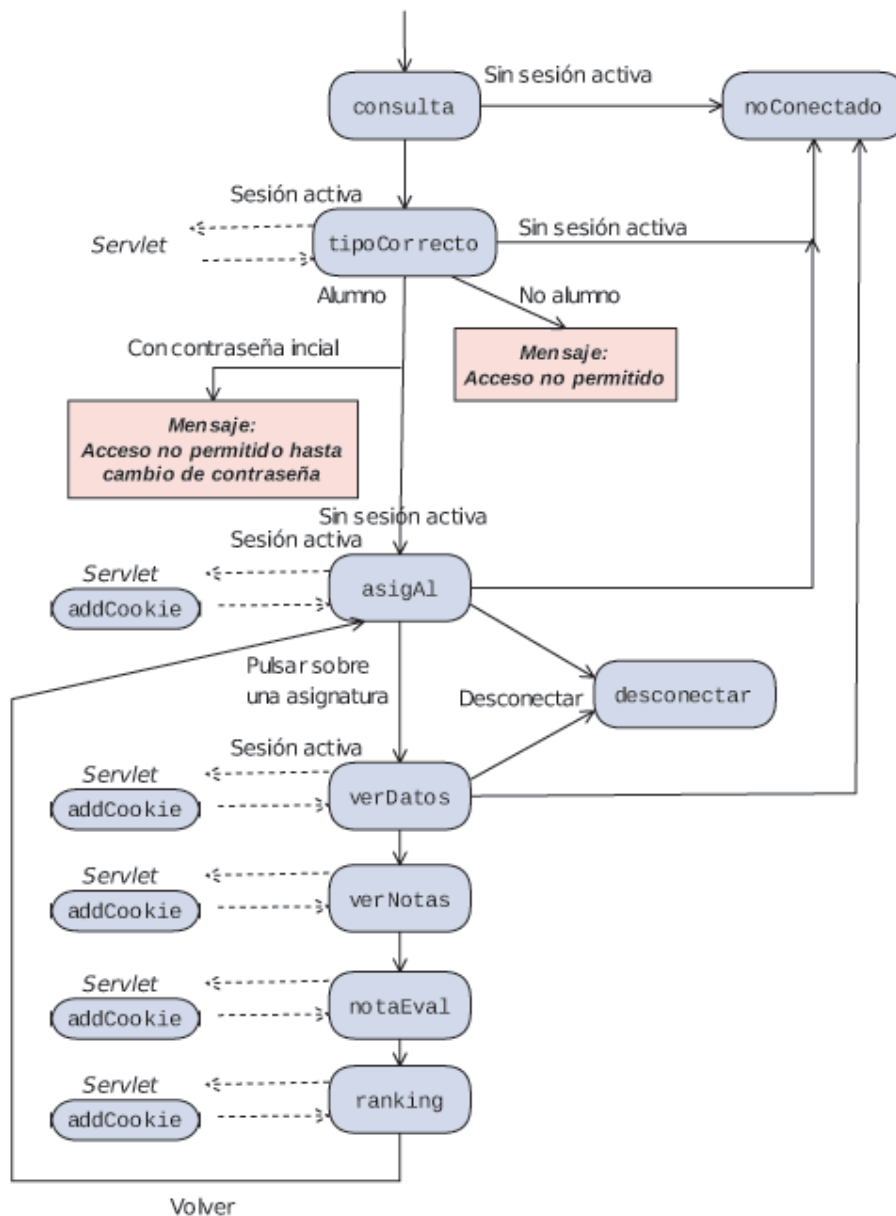


Figura 5.8: Flujo métodos página menú

5.4. Servidor

Se trata de un conjunto de 38 *servlets*, donde se implementan las funcionalidades del sistema. Salvo para los más simples, se esquematizará la información adicionalmente mediante tablas.

NOTA: Para la comunicación se ha seguido lo visto en Ejemplo 2 del Capítulo 2, con la particularidad de que la cabecera **tipo** de los mensajes no sólo se verá reducida a la opción de 0 ó 1, para poder particularizar con más detalle.

5.4.1. Sistema de acceso

Para este proceso, necesitaremos un único *servlet comprobacionAcceso.java*, que es llamado desde *acceso.html*. En él, se revisa si los datos de login y password del usuario que intenta autenticarse en el sistema se encuentran en la base de datos.

Comienza comprobando si el usuario se encuentra definido como profesor, para lo cual, se busca en la tabla de los profesores el *login* (que como ya hemos explicado, tiene la propiedad de unicidad). Si el usuario no se encuentra en esa tabla, se pasa a buscar en la tabla de los alumnos. Si no se ha encontrado en ninguna de las dos tablas, se enviara un mensaje a la parte del cliente en la que indicaremos que el usuario no existe en la base de datos. Si por el contrario, se encontró, ya sea como profesor o como alumno, para ambos casos se comprobará si el usuario ya ha cambiado la contraseña (variable *Tipo*). Finalmente, para cada caso profesor y para cada caso alumno tendremos dos posibles mensajes para redirigirnos convenientemente.

El Administrador será tratado como un caso aparte. Sus datos se encuentran en la tabla *profesor*, concretamente en la posición 1, por lo que sólo se comprobará si el usuario es el profesor que se encuentra en la posición 1 y no se realizará la verificación del cambio de la contraseña. Simplemente, cuando se detecte, se enviará un mensaje indicando que se trata de administrador.

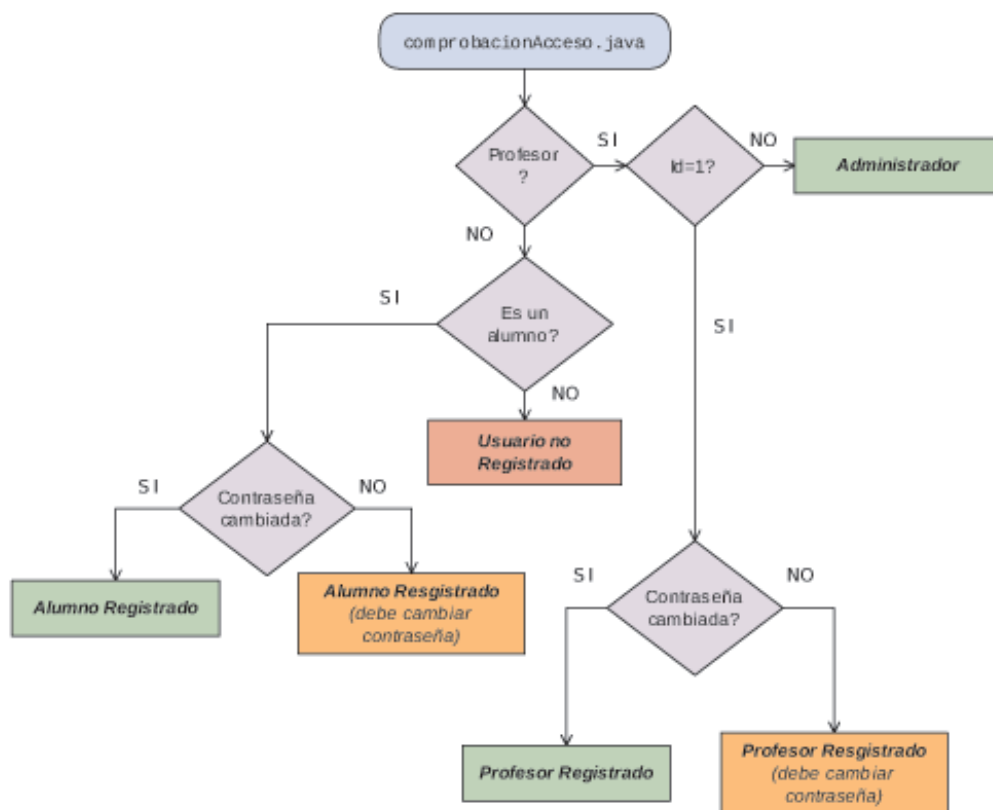


Figura 5.9: Flujo *servlet* comprobación acceso del Servidor

comprobacionAcceso	
Actores	Todos los usuarios
Descripción	Revisa si los datos de login y password del usuario que intenta iniciar sesión en el sistema, se encuentra en la base de datos
Flujo	<ul style="list-style-type: none"> ▪ Búsqueda del usuario en la base datos ▪ Se comprueba si el usuario ha cambiado ya su contraseña inicial (salvo Administrador). ▪ Respuesta con tipo de usuario y acceso. <p><i>Para más detalle véase figura 5.9</i></p>
Excepción	No existe el usuario
Página HTML	acceso.html

Tabla 5.1: Proceso para la comprobación del acceso en el Servidor

5.4.2. Sistema de recuperación de contraseñas

Para esta funcionalidad, se ha creado el *servlet* `enviarEmail.java`, llamado desde `recuperar.html`. En él se da solución a la necesidad del usuario de conocer su contraseña, ya sea porque no se recuerda o porque es la primera vez que se intenta entrar en el sistema y aún no se posee dicho conocimiento.

Esencialmente, tras comprobar que el usuario existe, ya sea profesor o alumno, buscando la dirección de correo electrónico en la base de datos (en el caso del profesor, se comparará con el *login* que es aquel que no se puede modificar), se envía un correo electrónico a dicha dirección con la información de la contraseña.

enviarEmail	
Actores	Todos los usuarios
Descripción	Envío de la contraseña del usuario por email.
Flujo	<ul style="list-style-type: none"> ▪ Comprobación de la existencia del usuario y adquisición de la contraseña. ▪ Creación del correo electrónico y envío. ▪ Envío respuesta al cliente.
Excepción	El usuario no existe en la base de datos.
Página HTML	recuperar.html

Tabla 5.2: Proceso para el envío de la contraseña en el Servidor

Como para el envío del correo es necesaria una cuenta de dirección electrónica, se ha creado una nueva: `noreply.aulavirtualpfc@gmail.com`).

NOTA: Para el correcto funcionamiento de este sistema, ha sido necesario la incorporación de un nuevo paquete *javaMail.jar*, tanto en el proyecto Eclipse como en el servidor.

5.4.3. Sistema de inserción de profesores

Como en los casos anteriores, la acción será realizada mediante un único *servlet* denominado *GuardarArchivo.java*, llamado desde *guardar.html*. Se trata de un *servlet* creado específicamente para facilitar al administrador el ingreso de los profesores, para que en vez de introducirlos de uno en uno, pueda introducirlos en bloque mediante un archivo *csv*.

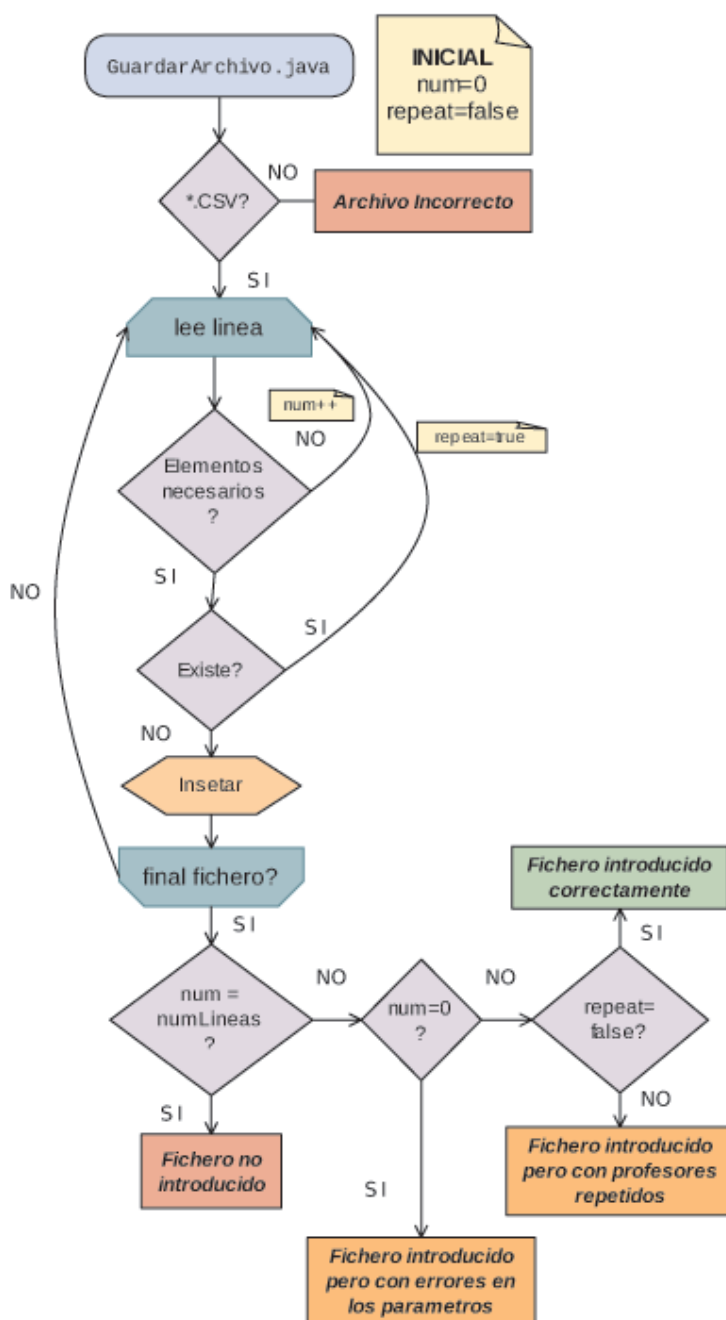


Figura 5.10: Flujo *servlet* guardarArchivo(profesores) del Servidor

GuardarArchivo	
Actores	Administrador
Descripción	Facilita el ingreso de los profesores en bloque mediante un archivo <i>csv</i> .
Flujo	<ul style="list-style-type: none"> ▪ Comprobación extensión del archivo. ▪ Lectura del archivo. ▪ Comprobación de la no previa existencia del profesor a introducir, e inserción. ▪ Envío respuesta al cliente. <p><i>Para más detalle véase figura 5.10.</i></p>
Excepción	<ul style="list-style-type: none"> ▪ Archivo con extensión incorrecta. ▪ El fichero contenía errores y no se ha podido.
Página HTML	guardar.html

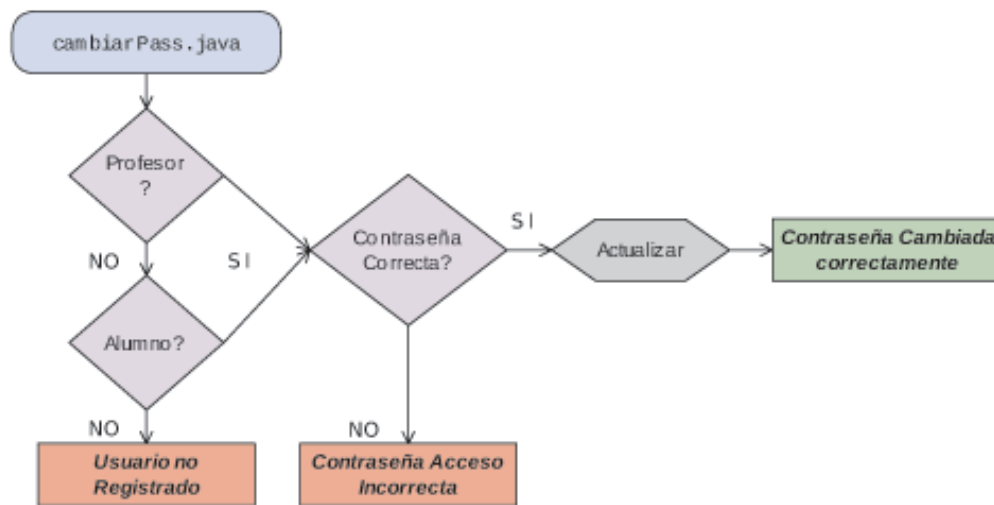
Tabla 5.3: Proceso para la inserción de profesores en el Servidor

Básicamente, en el tratamiento del archivo, lo que se comprueba es si el profesor que se quiere introducir en la base de datos contiene todos los parámetros necesarios y si ya existe (mediante la búsqueda por *login*). Si no existe, se introduce. En caso de existir o de falta de parámetros se pasa al siguiente, aunque en el mensaje que devolvemos al cliente, indicaremos que se han producido dichos errores en algún momento.

Para realizar esta función, el *servlet*, como en el caso de *comprobaciónAcceso* y con el mismo criterio, primero realizará un búsqueda sobre la tabla *profesor* y más tarde, en la de *alumno* (mira si el usuario está registrado en la base de datos). En caso de que el usuario NO esté registrado, se envía un mensaje de error al cliente. En caso afirmativo, se efectuará una segunda verificación en la que se comprueba que la contraseña que se quiere cambiar se corresponde con la almacenada en el sistema. De este modo, se verifica que el usuario que quiere cambiar la contraseña, es el propietario de la cuenta. Una vez confirmado, se realizará el cambio en la base de datos (actualizando el valor de la columna *Tipo*) si no, se envirá un mensaje de error al cliente.

5.4.4. Cambio de contraseña

Esta tarea se realiza en el *servlet* ***cambiarPass.java*** (correspondiente a *cambiarPassword.html*). Como su propio nombre indica, tiene la funcionalidad de cambiar la contraseña, siempre y cuando, el usuario esté registrado.

Figura 5.11: Flujo *servlet* cambiarContraseña del Servidor

cambiarPass	
Actores	Todos los usuarios
Descripción	Tiene la funcionalidad de cambiar la contraseña, siempre y cuando, el usuario este registrado.
Flujo	<ul style="list-style-type: none"> ▪ Búsqueda del usuario en la base datos. ▪ Comprobación contraseña correcta de acceso al servicio. ▪ Actualización datos. ▪ Envío respuesta al cliente. <p><i>Para más detalle véase figura 5.11</i></p>
Excepción	<ul style="list-style-type: none"> ▪ Usuario no registrado. ▪ La contraseña de acceso al servicio no se corresponde con el usuario.
Página HTML	cambiarPassword.html

Tabla 5.4: Proceso para el cambio de contraseña en el Servidor

5.4.5. Funcionalidades Profesor

Se tratan de los *servlets* que son llamados desde *inicio.html*.

5.4.5.1. Sistema verificación

Antes de permitir al usuario visualizar la página, se debe verificar si el usuario que intenta entrar en la página puede hacerlo. Con este cometido ha sido creado el *servlet comprobarTipo.java*.

En este caso, como a la página sólo pueden acceder profesores, sólo se revisa la tabla *profesor*, donde se busca el profesor y se verifica mediante el parámetro *Tipo*, si ya ha cambiado de contraseña inicial. Si el usuario no se encuentra o no se ha cambiado la contraseña inicial aún, se envía un mensaje para el no-acceso. En caso contrario, se enviará el mensaje de permiso.

comprobarTipo	
Actores	Profesores
Descripción	Sirve para verificar si el usuario que intenta entrar en la página, puede hacerlo.
Flujo	<ul style="list-style-type: none"> ▪ Búsqueda del usuario en la tabla <i>profesor</i> de la base datos. ▪ Se comprueba que ya ha cambiado su contraseña inicial. ▪ Envío respuesta al cliente. <p><i>Para más detalle véase figura 5.12</i></p>
Excepción	<ul style="list-style-type: none"> ▪ El usuario no es un profesor. ▪ No se ha cambiado la contraseña inicial.
Página HTML	inicio.html

Tabla 5.5: Proceso para la verificación en el Servidor

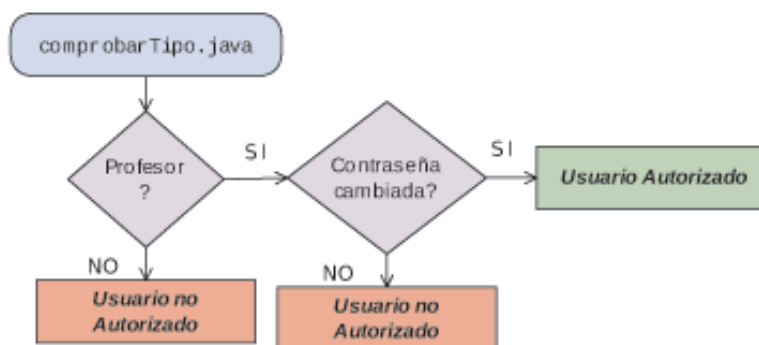


Figura 5.12: Flujo *servlet* comprobarTipo del Servidor

5.4.5.2. Settings

Para la realización de esta función, será necesario el empleo de dos *servlets*. Un primer *servlet*, que nos dé la información que se contiene en el servidor y un segundo, que modifique dicha información con los nuevos datos.

5.4.5.2.1. Obtener datos profesor

Con *buscarP.java* se da respuesta a la petición desde el cliente de los datos modificables (nombre, apellido y correo electrónico) que corresponden al profesor que pide la información.

Se trata de un *servlet* simple en el que se busca concretamente el usuario en la tabla *profesor*. Si existe, se devuelve la información pedida y si no, un mensaje de error, ya que el profesor, no existe.

5.4.5.2.2. Editar datos profesor

editarP.java sirve para cambiar los datos modificables del profesor (nombre, apellido y correo electrónico), que el propio profesor quiera.

En este *servlet* se busca concretamente el usuario en la tabla *profesor*. Si existe, se actualiza la información del profesor y si no, se enviará un mensaje de error.

5.4.5.3. Incorporar nuevo grupo de alumnos

Como en el caso anterior, para esta funcionalidad, será necesario más de un *servlet*. Concretamente, uno que nos permita visualizar las asignaturas, otro que nos permita visualizar los grupos y aquellos que nos permitan crear un grupo y una asignatura respectivamente.

5.4.5.3.1. Visualización asignaturas asociadas

El *servlet cAsigG.java* nos devuelve las asignaturas que tiene asignadas el profesor que realiza la petición, precisando los grupos (idAsig, nombre, grupo).

Busca en la información almacenada en la tabla *relacionAsigProf*, qué datos existen para el profesor que hace la petición, y guarda la información de asignatura y grupo. Como en dicha tabla no disponemos de la información del nombre de la asignatura, sólo del id, a partir de éste buscamos en la tabla *asignatura*, cuál es el nombre, y devolvemos la relación *id-nombre-grupo*.

cAsigG	
Actores	Profesor
Descripción	Nos devuelve las asignaturas que tiene asignadas el profesor que realiza la petición, precisando los grupos (idAsig, nombre, grupo).
Flujo	<ul style="list-style-type: none"> ▪ Búsqueda asignaturas asociadas al profesor en la base de datos. ▪ Búsqueda del nombre que corresponden a dichas asignaturas. ▪ Envío respuesta al cliente con la información.
Excepción	El profesor no tiene asignaturas asociadas.
Página HTML	inicio.html

Tabla 5.6: Proceso para la obtención de las asignaturas asociadas a un profesor en el Servidor

5.4.5.3.2. Crear asignatura

A partir de un nombre y de la información contenida en un archivo *csv*, *crearAsignatura.java* crea una nueva asignatura.

crearAsignatura	
Actores	Profesor
Descripción	A partir de la información contenida en un archivo <i>csv</i> , se crea una nueva asignatura.
Flujo	<ul style="list-style-type: none"> ▪ Comprobación de que la asignatura no exista previamente y creación. ▪ Creación asociación Asignatura-Profesor. ▪ Creación evaluación por defecto de la asignatura (media). ▪ Creación de la planificación de la asignatura. ▪ Envío respuesta al cliente con la información. <p><i>Para más detalle véase figura 5.13.</i></p>
Excepción	La asignatura ya existía y ya la tenía asociada.
Página HTML	inicio.html

Tabla 5.7: Proceso para la creación de una asignatura en el Servidor

El *servlet* comienza verificando que la asignatura no exista ya en la base de datos (comprobación mediante nombre). Si la asignatura ya existe, se comprueba que el profesor no la tenga asignada ya (tabla *relacionAsigProf*). En caso de no estar asignada, se asigna y termina. Si la asignatura no existía, se crea la asignatura sin planificación, se crea la asignación asignatura-profesor, se establece el tipo de evaluación de media por defecto y se

procede a la lectura del archivo para rellenar la parte de la planificación de la asignatura, es decir, se indican cuántos exámenes posee la asignatura (tabla *tipologia y examen*). Esta parte puede no ser rellenada si se encuentra algún error, pero la asignatura existirá.

Finalmente se envía la respuesta al cliente, indicando en qué parte del proceso hemos terminado.

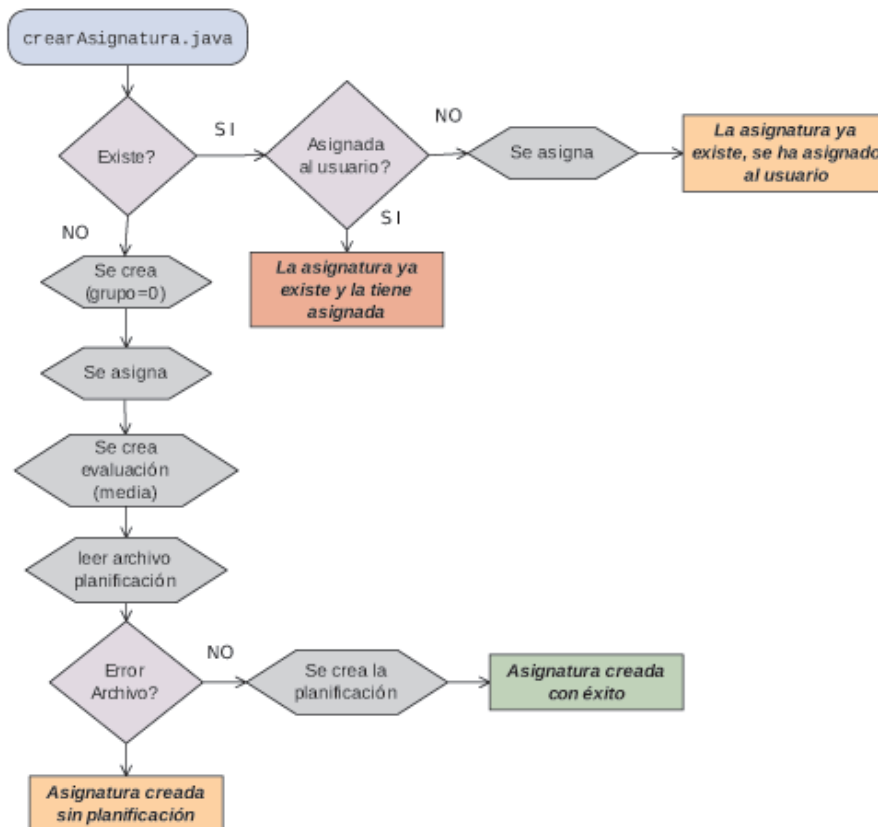


Figura 5.13: Flujo *servlet* crearAsignatura del Servidor

5.4.5.3.3. Visualización Grupos

En *verGruposAs.java* se responde a la petición del cliente sobre cuáles son los grupos que existen para una asignatura determinada, y se puntualiza cuáles están asignados al profesor que ha hecho la petición.

Primero se verifica que la asignatura existe (mediante búsqueda de id). Si no existe, se envía un mensaje de error. Si existe, prosigue comprobando si la asignatura tiene grupos. Si no se han creado grupos, se envía un mensaje al cliente, con dicha información. Si posee grupos, se comprueba si el profesor que pide la información los tiene asignados o no, y envía esa relación *grupo-asignado* al cliente.

verGruposAs	
Actores	Profesor
Descripción	Nos indica cuántos grupos tiene una asignatura determinada, puntualizando cuales están asociados al profesor.
Flujo	<ul style="list-style-type: none"> ▪ Comprobación de que la asignatura existe. ▪ Comprobación de la existencia de grupos. ▪ Comprobación de la asignación de dichos grupos con el usuario que realiza la petición. ▪ Envío respuesta al cliente con la información.
Excepción	La asignatura no existe.
Página HTML	inicio.html

Tabla 5.8: Proceso para la obtención de los grupos de una asignatura en el Servidor

5.4.5.3.4. Crear Grupo

A partir de un archivo *csv*, *guardarArchivoGrupo.java* almacena todos los alumnos que pertenecen al grupo que se pretende crear de la asignatura.

Para cada alumno del grupo, primero se comprueba que tenga todos los parámetros necesarios para poder crearse (si no, se pasa al siguiente), ya que de primeras el sistema asume que el alumno no existe en la base de datos. Lo siguiente a comprobar es si el alumno existe ya en la base de datos (verificación mediante *login*). Si no existe, se inserta en la tabla *alumnos*, (se crea el *login-pass* del alumno) y se crea la relación en *relacionAsigAlum*. Si el alumno ya existe, se comprueba que el alumno ya tenga esa asignatura asignada. En caso de no tenerla, se crea la relación a éste. Si el alumno ya tiene la asignatura asignada, se pasa al siguiente alumno.

El grupo será creado una única vez, cuando se cree la relación del primer alumno, al igual que la relación *relacionAsigProf*. Ya que si todos los alumnos, ya tienen asignada esa asignatura, no se creará un nuevo grupo, evitando así posibles problema de duplicidad de grupos. El grupo tampoco será creado, si ningún alumno a introducir posee los parámetros necesarios para ser creado, a pesar de que ya pueda existir en la base de datos, por la suposición inicial comentada anteriormente.

Debemos señalar que al crear un nuevo grupo nos encontramos con dos contadores, el de la tabla *asignatura* y el del número de grupo. Ambos, al crear un nuevo grupo aumentarán su último valor, pero podemos encontrar el caso en el que sea el Grupo 3, pero sólo existan dos grupos, por lo que en *asignatura* el valor será 2, pero en *relacionAsigProf-relacionAsigalum* el grupo será 3. Esto es debido a que el que encontramos en *asignatura* nos indica cuántos grupos existen actualmente de esta asignatura en la base de datos. Mientras que en las de relación, cuando se crea un nuevo grupo, se aumenta uno al nuevo grupo creado, ya que, por ejemplo, puede haberse borrado el Grupo 1, por lo que el Grupo 2

seguiría existiendo y, de no ser separados ambos contadores, se sobrescribiría.

guardarArchivoGrupo	
Actores	Profesor
Descripción	A partir de un archivo <i>csv</i> , almacena todos los alumnos que pertenecen al grupo a crear de la asignatura.
Flujo	<ul style="list-style-type: none"> ▪ Comprobación extensión correcta. ▪ Lectura fichero. ▪ Comprobación que el alumno a introducir contiene los parámetros necesarios. ▪ Comprobación de la no existencia previa de los alumnos e inserción. ▪ Comprobación de la no existencia previa de asociación Asignatura-alumno-Grupo y creación. ▪ Creación relación Asignatura-Profesor-Grupo. ▪ Envío respuesta al cliente con la información. <p><i>Para más detalle véase figura 5.14.</i></p>
Excepción	<ul style="list-style-type: none"> ▪ Extensión incorrecta. ▪ Todos los alumnos del archivo no contienen los parámetros necesarios. ▪ Todos los alumnos del fichero ya pertenecen a un grupo de esa asignatura.
Página HTML	inicio.html

Tabla 5.9: Proceso para la creación de un grupo de alumnos en el Servidor

Finalmente indicaremos al cliente mediante la contestación, si el grupo se ha creado correctamente, si se ha creado, pero había alumnos ya existentes o sin los parámetros adecuados, o si no se ha creado.

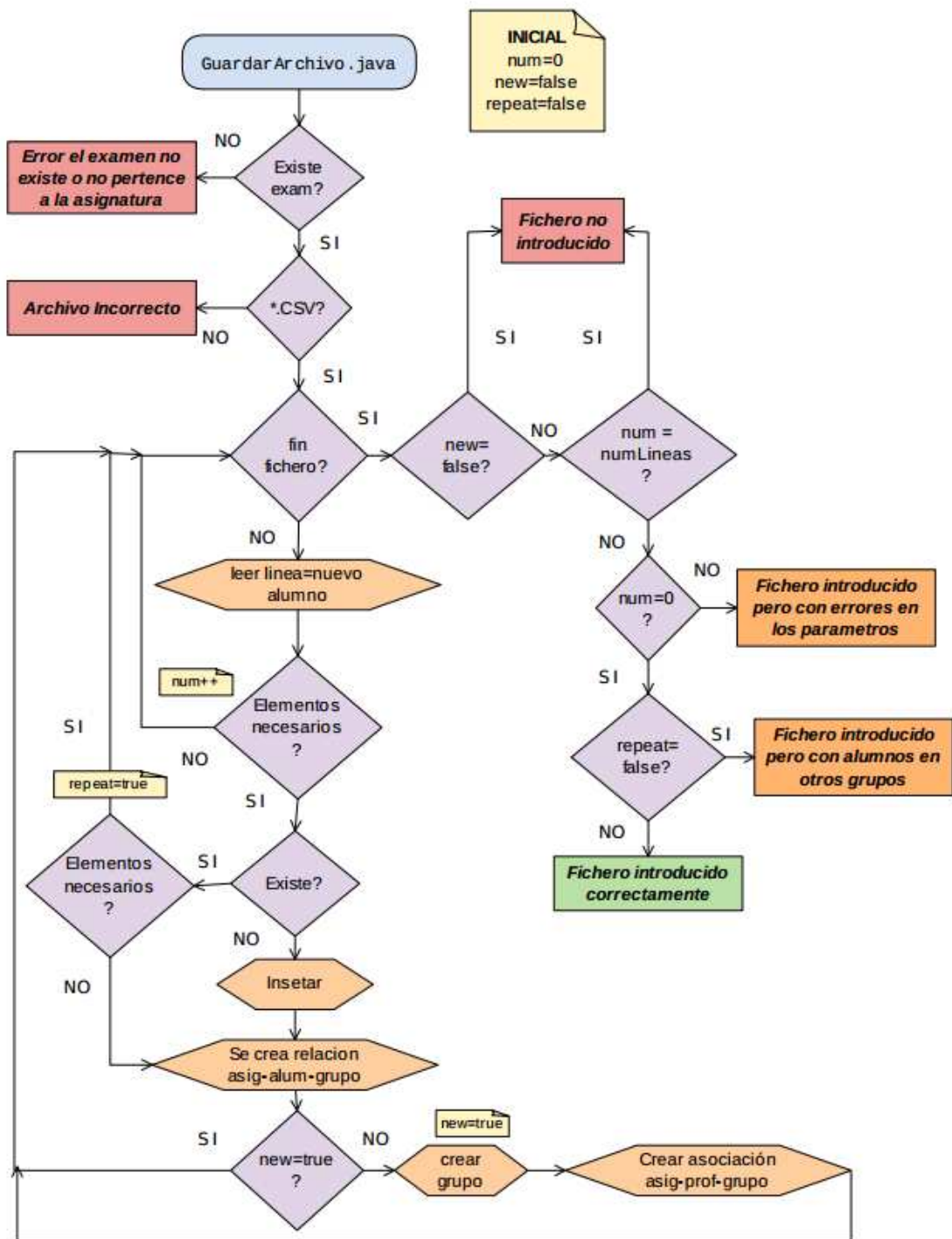


Figura 5.14: Flujo *servlet* guardarArchivoGrupo del Servidor

5.4.5.4. Gestionar mis grupos de alumnos

Inicialmente, el *servlet* *cAsigG.java* comentado anteriormente también será requerido en esta funcionalidad, para poder mostrar las distintos grupos que el profesor podrá gestionar.

A partir de la selección del grupo, se creará como ya se ha indicado un submenú, donde, para cada una de las funciones, necesitaremos uno o varios *servlets*.

5.4.5.4.1. Mostrar alumnos

Para realizar este proceso, se ha creado el *servlet* *mostrarAlum.java*, que nos devuelve los alumnos que componen un determinado grupo de una asignatura con id, nombre y apellido.

Se busca en la tabla *relacionAsigAlum* todos los alumnos de la asignatura y se meten en un *array* sólo aquellos que pertenecen al grupo requerido. Como en esta tabla no se encuentran los datos de nombre y apellido, se busca en la tabla *alumnos* para rellenar los datos. Finalmente, es enviado al cliente el *array* con la información (en caso de no existir alumno para ese grupo, se envía un mensaje informativo).

Como ya dijimos cuando explicamos la parte del cliente, desde esta funcionalidad se tiene acceso a otras: visualización y modificación de las notas del alumno de esa asignatura.

Visualización notas alumno

El *servlet* *notasAlum.java* obtiene para un determinado alumno las notas de los exámenes de una asignatura dada, en caso de haber planificación creada.

Para esta funcionalidad, el *servlet* comienza buscando las notas que tiene ese alumno (id, nota), ya que si no tiene ninguna todas las notas por defecto serán “-”. Prosigue buscando los exámenes que hay planificados para esa asignatura (id, nombre). Y se asocian ambas, donde, ante la falta de nota del alumno, se colocará “-” (id, nombre, nota).

Por defecto, se asume que todos los exámenes tiene apartados. En caso de no tenerlos, la información correspondiente se queda vacía, pero, en el caso de que alguno de los exámenes posea apartados, se obtiene el nombre de dichos apartados y las notas que corresponden a cada uno, y se adjunta esta información.

Finalmente, para cada examen obtenemos un *array* de la siguiente forma: id, nombre, nota, *array* nombreAptt, *array* notasAptt, apartados. En la última variable es donde indicamos al cliente si los campos correspondientes a los apartados están rellenos o no.

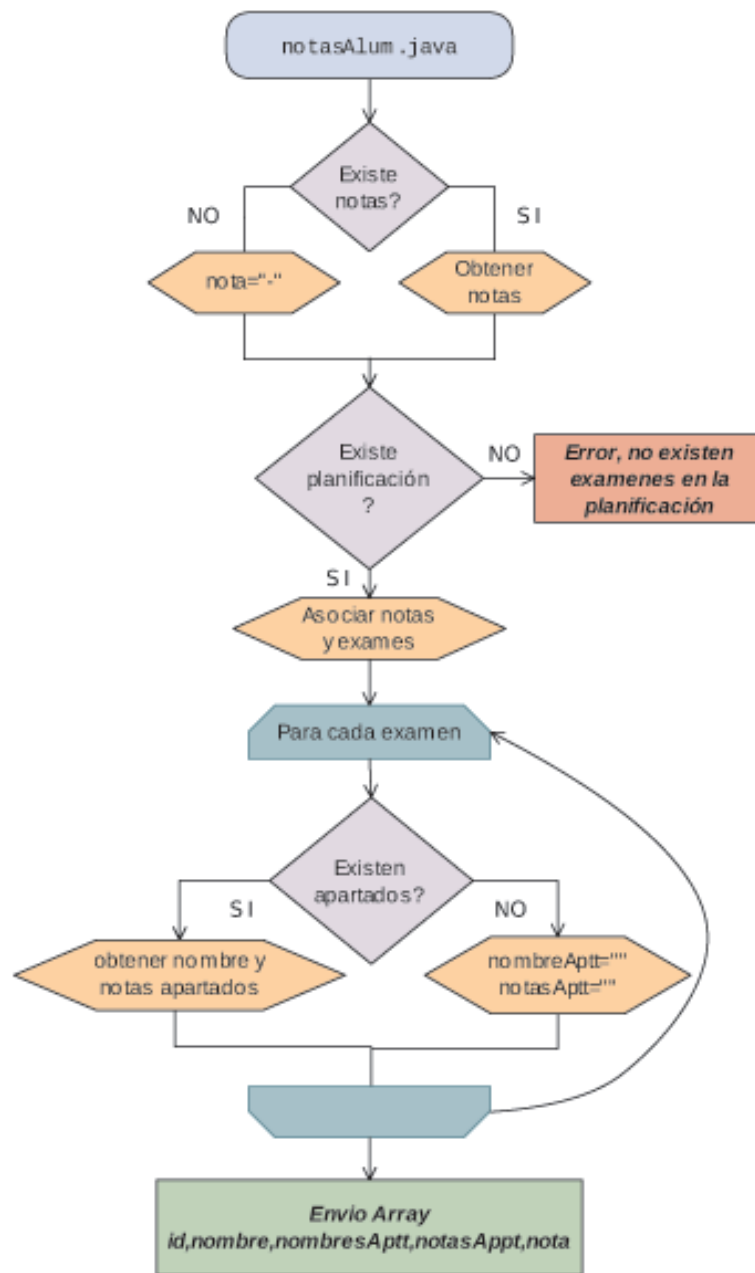


Figura 5.15: Flujo *servlet* notasAlum

notasAlum	
Actores	Profesor
Descripción	Se obtiene para un determinado alumno, las notas de los exámenes de una asignatura dada.
Flujo	<ul style="list-style-type: none"> ▪ Buscar notas exámenes alumno. ▪ Buscar exámenes asignatura. ▪ Asociar id, nombre, nota. ▪ Para cada examen, comprobar si existen apartados y se obtiene sus nombres y notas. ▪ Envío respuesta al cliente el array con la información (id, nombre, nombresAptt[], notasAptt[],nota). <p><i>Para más detalle véase figura 5.15.</i></p>
Excepción	No existe planificación
Página HTML	inicio.html

Tabla 5.10: Proceso para la obtención de las notas de un alumno en el Servidor

Cambiar nota examen

Con la finalidad de cambiar la nota de un examen de un alumno, siempre y cuando éste no exceda la nota máxima asignada a dicho examen, se han creado dos *sevlets*, dependiendo de si el examen contiene o no apartados. El *servlet cambiarNExam.java* es el encargado de modificar la nota en caso de que no haya apartados, mientras que *cambiarNEyA.java* lo es en el caso de tratarse de un examen con aparados.

- Sin apartados

Para realizar este proceso correctamente, primero el sistema comprueba que se dan las condiciones para ello, es decir, verifica que el examen y el alumno pertenecen a la asignatura, y que la nota que se está introduciendo, no supera la máxima permitida. Si el proceso de comprobación ha sido exitoso, se procede a modificar la nota.

Como al introducir o modificar la nota de un alumno, la nota en la evaluación cambia y, por lo tanto, el ranking también, se procederá a recalcularse la evaluación para este alumno pero el ranking para todos, ya que no sabemos a cuantas posiciones por encima o por debajo podría afectar.

- Con apartados

Las comprobaciones iniciales serán las mismas, tras las cuales, se procederá a la realización de nuevas para los apartados (que existen para ese examen y no superan la nota máxima correspondiente). Como en el caso anterior, en caso de que la comprobación se haya realizado sin errores, se procederá a modificar la nota de los apartados y del examen.

NOTA: El cambio en un apartado implica el cambio de la nota final, por lo que se recalculará la nota final a partir de la de los apartados (y se verificará que sea menor que la máxima permitida en el examen). En caso de que un apartado tenga letras, la nota final será la suma de los apartados numéricos.

cambiarNExam	
Actores	Profesor
Descripción	Cambia la nota de un examen de un alumno, siempre y cuando este no exceda la nota máxima asignada a dicho examen.
Flujo	<ul style="list-style-type: none"> ▪ Comprobación de la existencia del examen en la asignatura determinada y obtención de su nota máxima. ▪ Verificación de que la nota a insertar es menor que la nota máxima permitida. ▪ Comprobación de que el alumno pertenece a la asignatura. ▪ Inserción o modificación de la nota. ▪ Recálculo nota evaluación. ▪ Recálculo de la posición en el ranking. ▪ Envío respuesta al cliente con la información. <p><i>Para más detalle véase figura 5.16.</i></p>
Excepción	<ul style="list-style-type: none"> ▪ El examen no existe en la asignatura. ▪ El alumno no pertenece al grupo. ▪ La nota que se está intentando introducir es mayor que la nota máxima del examen.
Página HTML	inicio.html

Tabla 5.11: Proceso para la modificar la nota de un examen sin apartados en el Servidor

NOTA: Para el caso con apartados, la tabla correspondiente es igual a la de la tabla 5.11 matizando que cuando se habla de nota, no sólo se habla de la final, si no de final más la de los apartados.

NOTA2: Para un mayor detalle de cómo se hacen los cálculos de ranking y evaluación, ver `introExam.java`.

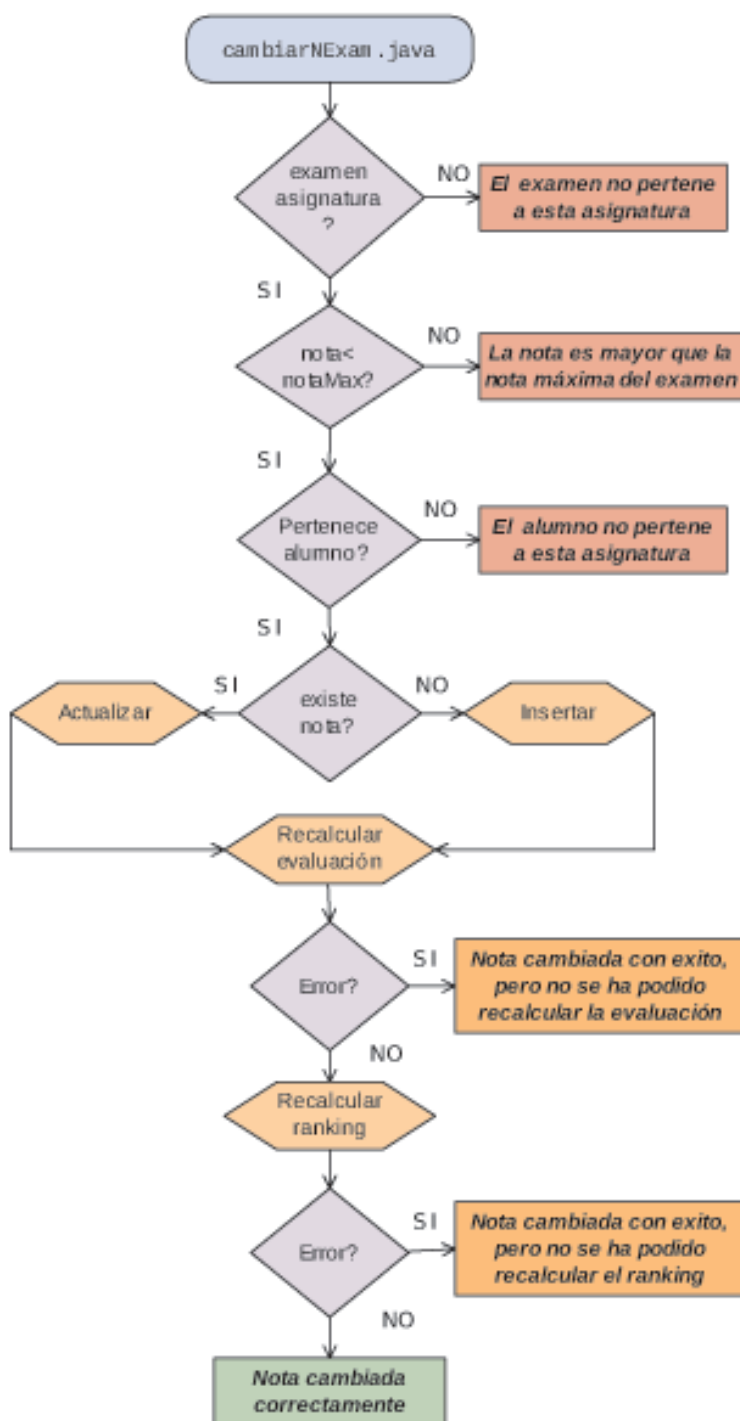


Figura 5.16: Flujo *servlet* Cambiar Nota Examen sin apartados del Servidor

5.4.5.4.2. Borrar alumno

Con esta función se ha creado el *servlet* `borraralum.java`, que sirve para eliminar un alumno de un determinado grupo de una asignatura.

Una vez introducido un alumno, nunca será borrado por completo del sistema, salvo

que sea por el administrador, ya que aunque el alumno no tenga ninguna otra asignatura, lo único que se borrará de la base de datos, será la información contenida en la tabla *relacionAsigAlum*.

NOTA: Aunque siempre que se elimine un alumno, la posición en el ranking de otros se verá afecta, se ha decido por facilidad de diseño, no recalcular en esta parte el ranking, ya que esta modificación sólo afecta a subir una posición a aquellos que se encuentren por debajo, manteniendo así la motivación del alumno y que no cree la falsa esperanza de subir posición sin esfuerzo.

5.4.5.4.3. Introducir alumnos

Con dicho propósito, se ha creado el *servlet* ***introduciralum.java***, el cual introduce un alumno dentro de un determinado grupo.

Primero se comprueba si el alumno existe, en cuyo caso se verifica que no tenga asociada la asignatura, ya que un alumno no puede estar en dos grupos a la vez, y se asocia. Si no existía, se crea y se introduce la asociación. Respondemos al cliente, informando del proceso realizado.

NOTA: Al introducir un alumno, no se modifica la posición en el ranking de ningún alumno, debido a que cuando se introduce el alumno no se introduce ninguna nota. El nuevo alumno insertado será el que se encuentre en la última posición.

5.4.5.4.4. Introducir notas

Para este proceso, será necesaria la llamada a más de un *servlets*, puesto que para cada una de las fases del proceso será necesario la utilización de un *servlet* distinto.

Obtención de exámenes

El *servlet* ***obtenerExams.java*** ha sido implementado de modo que para una determinada asignatura, nos indique los nombres de los exámenes planificados, en caso de haber planificación para esa asignatura.

Esta funcionalidad es debida a la necesidad de permitir únicamente la introducción de notas de aquellos exámenes previamente planificados. Se trata de un sistema de seguridad que no debemos confundir con una limitación de la aplicación, puesto que nuevos exámenes pueden ser creados desde *Planificación*.

Introducir notas examen

A partir de un fichero *csv*, ***introExam.java*** introduce las notas de un examen de un grupo concreto de una asignatura.

Como hasta que un profesor no pone un examen, no sabe cuántos apartados va a poseer dicho examen, no es hasta este momento cuando se crean los apartados necesarios. Además subir una nota, incluyendo apartados o no, dependerá del propio profesor, que puede querer sólo subir al sistema la información final del examen.

Los apartados solamente serán creados la primera vez que se introduzca un alumno, ya que como en el caso del *servlet guardarArchivoGrupo*, podría darse el caso en el que no estuvieran todos los parámetros necesarios para crear lo apartados, o que el alumno no tuviera todas las notas necesarias por error o, incluso, que el examen ya posea apartados, porque se introdujo anteriormente el examen, y el número de apartados nuevos y viejos no coincida (comparando por nombre, el resto de parámetros se actualizarían junto con la nota).

El funcionamiento básico que realiza este proceso consiste en una vez verificado que el examen se corresponde con la asignatura (paso del que obtenemos además su id) y que el archivo tiene la extensión correcta. si el examen no posee apartado, se introduce o actualiza (ya que podría haber una nota anterior) directamente la nota de cada alumno en la tabla *notaExamen* (Siempre tras comprobación de que el alumno pertenece al grupo al que se le están asignando las notas/examen). Si el examen tiene apartados, conclusión que obtenemos a través de un parámetro externo, se crean o actualizan los apartados (*apartados*), y se van añadiendo las notas (*notaApartados*) de los distintos alumnos.

introExam	
Actores	Profesor
Descripción	A partir de un archivo <i>csv</i> , almacena todos los alumnos que pertenecen al grupo a crear de la asignatura.
Flujo	<ul style="list-style-type: none"> ▪ Comprobación extensión correcta. ▪ Lectura fichero. ▪ Si contiene apartados se obtiene su información (serán introducidos con la introducción de las notas del primer alumno). ▪ Introducción de la nota, apartados incluidos si se tienen, tras previa comprobación de que el alumno pertenece al grupo. ▪ Recálculo de la evaluación y el ranking de los alumnos que componen el grupo. ▪ Envío respuesta al cliente con la información.
Excepción	<ul style="list-style-type: none"> ▪ El examen no existe. ▪ Extensión incorrecta. ▪ Problemas al leer el archivo.
Página HTML	inicio.html

Tabla 5.12: Proceso para la introducir un examen en el Servidor

Hay que tener en cuenta que, al igual que en *cambiarNExam.java*, se está modificando una nota, y, al igual que exponíamos en dicho caso, esto acarrea una variación en la nota

de evaluación, que desemboca en la necesidad del recálculo de la nota de evaluación y de la posición del ranking.

Debido a que existen dos tipos posibles de evaluación, para el cálculo de ésta, primero se debe comprobar que tipo de evaluación posee la asignatura. Si la asignatura tiene el sistema de evaluación *Por defecto*, la evaluación consistirá en el cálculo de la nota media, mientras que si el sistema es el *Personalizado*, la nota será aquella que se obtenga de las correspondiente operaciones almacenadas en *evaluación*.

En ambos casos se diferencian dos notas, la temporal y la final. La temporal sólo tiene sentido para el caso en el que no se hayan realizado todos los exámenes de la asignatura, que será cuando se diferencia de la final.

Para el caso *Por defecto*, la forma en la que calculamos la evaluación se muestra a continuación:

$$notaTemporal = \frac{\sum_{i=0}^n NotasObtenidas}{\sum_{i=0}^n NotasMaximasExámenesRealizados}$$

$$notaFinal = \frac{\sum_{i=0}^n NotasObtenidas}{\sum_{i=0}^n NotasMaximasExámenesTotales}$$

$$umbralFinal = \frac{\sum_{i=0}^n NotasMaximasExámenesTotales}{2}$$

$$umbralTemporal = \frac{\sum_{i=0}^n NotasMaximasExámenesRealizados}{2}$$

Para el caso de evaluación *Personalizada*, la forma en la que calculamos la evaluación será aquella que se tiene almacenada en la tabla *evaluación*, donde además de la fórmula, tendremos almacenado el umbral a aplicar. En este caso, la nota temporal, será el resultado de escalar la nota actual del alumno.

$$notaTemporal = \frac{\sum_{i=0}^n NotaActualAlumno * NotaMaximaActual}{NotaMaximaTotal}$$

NOTA: Esta cálculo se realiza por tipologías, ya que la personalización no tiene sentido si no es para resaltar determinadas tipologías frente a otras.

Una vez realizado el cálculo, la información es actualizada en la tabla *relacionAsigAlum*.

Finalmente, con esta nueva nota, se procederá al cálculo de la nueva posición. Este proceso de recálculo de la posición en la clasificación de la asignatura se ha llevado a cabo de modo particular para cada alumno. Es decir, para cada alumno inicialmente suponemos

que su nota es la mejor (inicialmente en la posición 1 del ranking) y se va comparando con la del resto del alumnos. Si la nota con la que se compara es mejor (mayor), se aumenta en una unidad la posición del alumno (baja en la clasificación del puesto 1 al 2). En caso de tener la misma nota, la posición vendrá determinada por el NIA. Aquel con menor NIA, será el que obtenga una mejor posición (cualquier otro sistema sería válido, ya que se trata de una opción diseñada para motivar al alumno).

De nuevo, la tabla *relacionAsigAlum* será actualizada para cada uno de los alumnos, con su nueva posición.

Finalmente se envía la respuesta al cliente, indicando el id del examen creado si todo ha ido bien y si no, indicando cuando ha sido el error.

Mostar notas del examen

Para el examen precisado, *mostrarNGE.java* obtiene la nota de los alumnos que pertenecen a un grupo concreto.

mostrarNGE	
Actores	Profesor
Descripción	Obtiene la nota de los alumnos que pertenecen a un grupo concreto.
Flujo	<ul style="list-style-type: none"> ▪ Se obtiene la nota de todos los alumnos que han realizado ese examen. ▪ Se obtienen todos los alumnos del grupo. ▪ Se asocia el alumno con su nota (si no tiene: guión). ▪ Si el examen tiene apartados, se obtiene sus características y las notas correspondientes a cada alumno. ▪ Envío respuesta al cliente con la información.
Excepción	<ul style="list-style-type: none"> ▪ No se han encontrado notas para ese examen. ▪ No se han encontrado alumno en el grupo.
Página HTML	inicio.html

Tabla 5.13: Proceso para obtener la nota de un examen en el Servidor

Primero se obtienen el id de los alumnos que pertenecen al grupo (tabla *relacionAsigAlum.html*) para que así, aunque un alumno no tenga nota del examen, se muestre. A continuación, con esos identificadores obtenemos el nombre y los apellidos, de la tabla *alumnos*, y la nota del examen, de *notaExamen* (en caso de no encontrar notas para ese examen, se enviará un guión).

Como un examen puede tener apartados, el siguiente paso es confirmar si el examen tiene apartados. En caso de no poseer apartados, se finaliza el proceso y se envía la información. En caso de tener, se obtienen éstos con sus características y las notas del alumno para esos apartados.

Para terminar se construye el *array* con toda la información, que será enviado como contestación al cliente. Diferenciamos si hay apartados o no por el tipo de mensaje que reciba el cliente. En ambos casos, en la posición 0 obtendremos el nombre del examen.

En caso de no poseer apartados, se finaliza el proceso y se envía la información. Si hay apartados, en la posición 1, encontraremos un array con los nombres de los apartados. En las siguientes posiciones, encontraremos las características de estos apartados: *materia*, *web*, *nota máxima* (una posición por apartado) y finalmente los alumnos con sus notas (*login*, *nombre*, *apellido*, *nota*, *notas apartados[]*).

Desde esta vista, se nos dará la opción para modificar las características de los apartados. Proceso que se explica a continuación.

NOTA: Este servlet es llamado en este punto del sistema, para mostrarnos las notas del examen una vez introducidas.

Cambiar características apartados

Con el objetivo de poder variar las características iniciales (materia, web, nota máxima) de los apartados de un examen, se ha creado el *servlet cambiarAptt.java*.

NOTA: Durante el proceso, se tiene en cuenta que si se cambia la nota máxima de un apartado, la nota máxima del examen también variará y lo modifica en función de ésta.

5.4.5.4.5. Planificación

Para esta funcionalidad, será necesario el uso de varios *servlets*, entre ellos el *servlet mostrarNGE.java* (y como consecuencia *cambiarAptt.java*) ya comentado anteriormente. En este caso, su uso responde a la necesidad por parte del profesor de ver las notas de un determinado examen en el momento que éste lo requiera.

Obtener exámenes

El *servlet obtenerExam.java* adquiere los distintos exámenes que pertenecen a una asignatura con sus características (nombre, tipología y nota máxima), siempre que exista una planificación para dicha asignatura.

Obtener tipología

Dada una asignatura, *obtTipologia.java* obtiene la tipología asociada a los exámenes, en caso de existir.

NOTA: Esta acción es necesaria, ya que en la creación o modificación de un examen se muestra un desplegable, para la selección de la tipología, que muestra los elementos existentes.

Crear examen

Para el proceso de crear un nuevo examen en una determinada asignatura empleamos el *servlet* **nuevoExam.java**.

Comienza comprobando que no haya ningún otro examen con el mismo nombre, ya que podría crear un conflicto. Continúa, comprobando que la tipología elegida se encuentra dentro de las permitidas, y si todo ha ido bien, crea el examen, si no se envía la información con el problema al cliente.

NOTA: Introducir un nuevo examen en un planificación implica que el cálculo de la nota de evaluación varíe, por lo que se recalculará la nota de evaluación y ranking de todos los alumnos del grupo.

Crear nueva tipología de examen

Con la funcionalidad de crear una nueva tipología de examen de una asignatura, siempre y cuando esta no exista ya en la base de datos, se ha implementado el *servlet* **nuevoTipo.java**.

Borrar examen

Para el proceso de borrar un examen de una asignatura, se ha creado **borrarExam.java**.

NOTA: Como en el caso de introducir un nuevo examen en una planificación, eliminarlo también implica que el cálculo de la nota de evaluación varíe, por lo que se recalculará la nota de evaluación y el ranking de todos los alumnos del grupo.

Cambiar características examen

Como para el caso de los apartados, en nuestra aplicación existe la opción de cambiar las características de los exámenes de la planificación a través del *servlet* **cambiarExamen.java**.

NOTA: En caso de que el examen contenga apartados, aunque se modifique la nota máxima, no cambiará la de los apartados, por lo cual debe tenerse cuidado al emplear esta función si el examen ya se ha realizado y contiene apartados.

5.4.5.4.6. Evaluación

En esta opción de la aplicación se emplean dos *servlets*, uno que nos da la información acerca de la evaluación que tiene la asignatura seleccionada, y otro que nos permite personalizarla.

Mostrar evaluación

Se ha creado el *servlet* `evaluacion.java`, para obtener toda la información correspondiente a la evaluación: tipo, notas de los alumnos del grupo y si se ha obtenido tras realizar todos los exámenes o sólo parte de ellos.

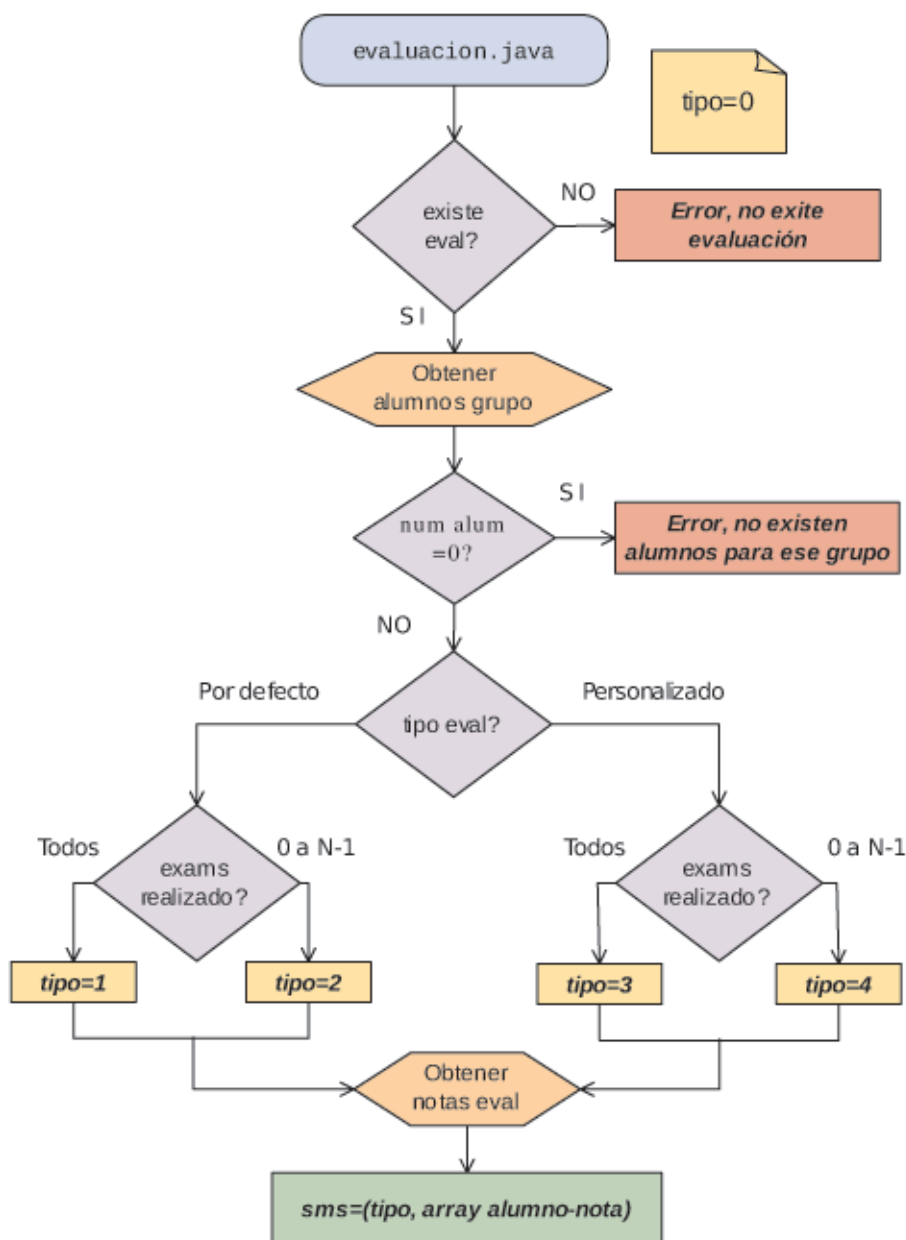


Figura 5.17: Flujo *servlet* Evaluación del Servidor

Para obtener estos datos, el proceso implementado comienza obteniendo el tipo de

evaluación, y verificando la realización de todos los exámenes de la asignatura. La combinación de ambas comprobaciones serán indicadas al cliente a través del tipo de mensaje. El siguiente paso, será aquel que nos da la información a insertar en el cuerpo del mensaje y que se corresponde con la obtención de los alumnos que componen el grupo, junto con sus notas de la evaluación.

evaluación	
Actores	Profesor
Descripción	Obtiene la información correspondiente a la evaluación.
Flujo	<ul style="list-style-type: none"> ▪ Se comprueba el tipo de evaluación. ▪ Se obtienen todos los alumnos del grupo. ▪ Se comprueba que existan exámenes en la planificación. ▪ Se obtiene la nota para cada alumno. ▪ Envío respuesta al cliente con la información. <p><i>Para más detalle véase figura 5.17.</i></p>
Excepción	<ul style="list-style-type: none"> ▪ No existen exámenes en la planificación. ▪ No existen alumnos en la asignatura.
Página HTML	inicio.html

Tabla 5.14: Proceso para la obtención de los distintos parametros de la evaluación en el Servidor

Personalizar evaluación

Con la función de poder personalizar el criterio de evaluación mediante un archivo *csv*, se ha implementado el *servlet* ***personalizarEval.java***.

El proceso obtiene del archivo la fórmula para el cálculo de la evaluación de la asignatura y la introduce en el sistema. Esta modificación implica el recálculo de la nota de evaluación (final y temporal) y por consiguiente de la posición en el ranking para todos los grupos de la asignatura, ya que, aunque se trate del cambio en un solo grupo, automáticamente se cambiará en todos los grupos (ya que se supone igualdad de criterio para los alumnos de una asignatura, independientemente de su grupo).

Será en la realización de este recálculo, cuando se compruebe si la fórmula que se ha introducido otorga nota en todos los casos posibles. En caso de que esto no sea así, se volverá a cambiar el modo de evaluación a *Por defecto*, y se enviará el mensaje de error.

personalizarEval	
Actores	Profesor
Descripción	Cambiar de un modo personalizado el modo de evaluación mediante un archivo <i>csv</i> .
Flujo	<ul style="list-style-type: none"> ▪ Comprobación extensión archivo. ▪ Obtención de la fórmula de evaluación (fórmula=fórmula+línea;). ▪ Modificación de la evaluación a tipo 1 e inserción de la fórmula. ▪ Recálculo de la nota de evaluación y ranking para todos los grupos de la asignatura (En caso de error, se vuelve a la configuración no personalizada: tipo=0 y fórmula="-"). ▪ Se envía la respuesta al cliente con la información.
Excepción	<ul style="list-style-type: none"> ▪ Extensión de archivo incorrecta. ▪ No existen exámenes en la planificación. ▪ No existen alumnos en la asignatura. ▪ No existe evaluación para la asignatura. ▪ No existe tipología para la asignatura. ▪ El archivo no contempla la posibilidad de nota en todos los casos (ejemplo IF sin ELSE).
Página HTML	inicio.html

Tabla 5.15: Proceso para la personalización de la evaluación en el Servidor

5.4.5.4.7. Download

Con la funcionalidad de poder obtener una copia de seguridad mediante la descarga de un archivo *csv*, con la información que existe en el sistema, se ha creado *servlet download.java*.

Esencialmente, tras comprobar que existen alumnos para ese grupo y una planificación, se adquieren los datos almacenados en el servidor acerca de las notas correspondientes a las notas de los distintos exámenes para cada alumno y la nota de la evaluación final. Esta información se introduce en un archivo y se envía (descarga).

En caso de error el archivo también será creado y enviado, aunque, en ese caso, el cuerpo del mensaje contendrá la información con el error.

download	
Actores	Profesor
Descripción	obtener una copia de seguridad mediante la descarga de un archivo <i>csv</i> , con la información que existe en el sistema.
Flujo	<ul style="list-style-type: none"> ▪ Se obtienen todos los alumnos del grupo. ▪ Se comprueba que existan exámenes en la planificación. ▪ Se obtiene las notas para cada alumno (exámenes y de evaluación). ▪ Se crea el archivo con la información. ▪ Se descarga el archivo.
Excepción	<ul style="list-style-type: none"> ▪ No existen exámenes en la planificación. ▪ No existen alumnos en la asignatura.
Página HTML	inicio.html

Tabla 5.16: Proceso para descargar una copia de seguridad del Servidor

5.4.5.5. Borrar Grupo/Asignatura

El *servlet cAsigG.java*, comentado anteriormente, también será requerido en esta funcionalidad, puesto que nos permite visualizar inicialmente los grupos asociados al profesor, entre los cuales, se seleccionarán aquellos que quieran ser borrados.

Aunque la acción de borrar, como tal, la hará el *servlet borrarGrupo.java*.

Aunque se habla de borrar un grupo, esta acción sólo se realizará propiamente si para el grupo seleccionado el único profesor asociado es el que quiere borrarlo (hay más grupos, por lo que no puede borrarse la asignatura). La acción consistirá en eliminar de la base de datos la relación del profesor y la de los alumnos que componen el grupo con la asignatura.

En caso de haber más profesores asociados al grupo, sólo se puede la borrar la asociación al grupo del profesor, ya que el resto de profesores no tiene que verse afectado por esta acción.

La asignatura completa sólo será borrada, si tiene un único grupo (el que se intenta borrar) y el único profesor asignado es el que intenta borrarlo. De este modo, para borrar una asignatura completa, únicamente será necesario seleccionar todos los grupos que la componen. Aunque el proceso irá borrando los grupos de uno en uno, finalmente se acotará a uno que desenlazará en la eliminación total de la asignatura.

borrarGrupo	
Actores	Profesor
Descripción	Borra un grupo de una asignatura, o la asignatura completa si corresponde.
Flujo	<ul style="list-style-type: none"> ▪ Se comprueba que la asignatura existe. ▪ Se determina cuantos profesores hay asociados a esta asignatura (y cuantos al grupo). <ul style="list-style-type: none"> • En caso de haber un solo usuario asignado a la asignatura (el que realiza la acción), se borra la asignatura. • Si hay más de un profesor asociado al grupo, se elimina la asociación al grupo del profesor. • Si para el grupo seleccionado, el único profesor asociado es el que quiere borrarlo, se borra el grupo. ▪ Se envía la respuesta al cliente con la información.
Excepción	La asignatura no existe
Página HTML	inicio.html

Tabla 5.17: Proceso para la borrar un o varios grupos en el Servidor

5.4.6. Funcionalidades alumno

Todos los *servlets* que aquí se exponen son los llamados desde *menu.html*.

5.4.6.1. Sistema verificación

Como en el caso de *las funcionalidades del profesores*, antes de permitir al usuario visualizar la página, se debe verificar si el usuario que intenta entrar en la página puede hacerlo. Con este cometido ha sido creado un nuevo *servlet* ***comprobarTipoAl.java***, que se corresponde con el funcionamiento del *servlet* ***comprobarTipo.java*** pero, en este caso, verificaremos que se trata de de un alumno revisando en la tabla *alumnos*.

5.4.6.2. Sistema alumno

Denominamos sistema alumno a aquel donde se encuentran las funcionalidades básicas del alumno.

5.4.6.2.1. Asignaturas alumno

Para obtener las asignaturas que tiene asociadas el alumno se ha creado el *servlet* ***AsigAl.java***, cuyo funcionamiento es igual al de ***cAsigG.java*** (explicado anteriormente), donde la diferencia se encuentra en la tabla a consultar (*relacionAsigAl* en vez de *relacionAsigProf*, ya que en este caso se trata de la información concerniente al alumno en vez de al profesor).

5.4.6.2.2. Información asignatura

Esta vista muestra una la combinación de varios *servlets*, que mostraran la información relevante para el alumno.

Obtener profesores asignatura

Inicialmente, la vista nos mostrará los profesores que imparten dicha materia. Con la finalidad de obtener dicha información, se ha creado el *servlet* ***datosAsig.java***.

Su función es sencilla, simplemente busca los profesores asociados a una asignatura-grupo y le devuelve esta información al cliente.

Obtener notas examen

A continuación, se mostrarán los exámenes de la asignatura con sus características (en caso de poseer apartados las de éstos también) y con la nota correspondiente del alumno.

En la parte del profesor, ya existía un proceso parecido (***notasAlum.java***), que nos daba las notas de los exámenes de una asignatura de un alumno, pero sin entrar en sus características. Simplemente hemos añadido al proceso anterior la información correspondiente (el *array* enviado al cliente contendrá la información *idexamen*, *nombre*, *materia*, *web*, *apartados con toda su información[]*), en el *servlet* que hemos creado para este proceso ***examAl.java***.

Obtención de la nota evaluación

En este caso para obtener los datos requeridos se ha implementado ***notaEvalAlAsig.java***. Se trata de una particularización del *servlet* ***evaluación.java*** para un único alumno.

Ranking

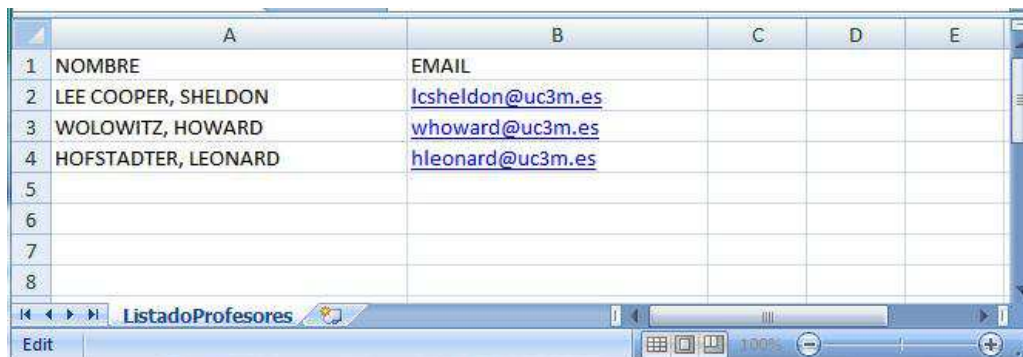
Con la finalidad de motivar al alumno, al final de la vista se muestra su posición en el ranking, que se obtiene del proceso creado por el *servlet* ***ranking.java*** que extrae la información correspondiente de la base de datos (tabla *relacionAsigAlum* correspondiente a esa asignatura).

NOTA: El administrador, al encontrarse en la tabla profesores, no tendrá acceso a esta página.

5.5. Archivos

Como hemos visto, una gran parte de la información que se va a introducir en la base de datos es leída de ficheros *csv*, por lo que en esta sección explicaremos como debe ser cada uno de los distintos archivos para el correcto funcionamiento del sistema.

El primer archivo del que vamos a hablar, que vemos en la figura 5.18, es aquel en el que se encuentran los profesores que van a pertenecer al sistema, aquel que introduce el administrador en el sistema a través del formulario de la página *guardar.html*.



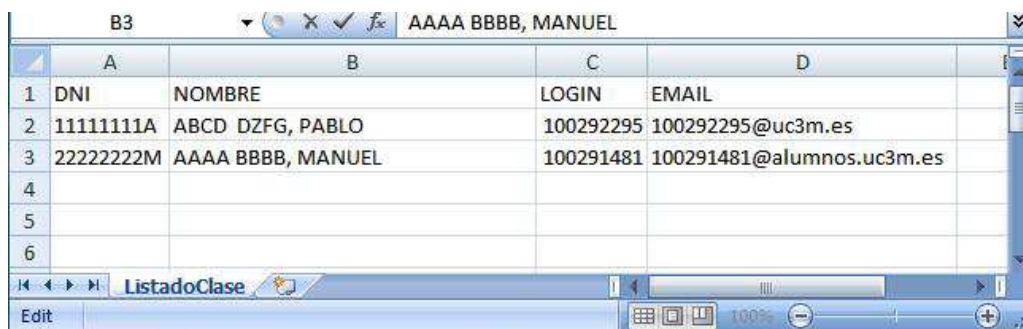
	A	B	C	D	E
1	NOMBRE	EMAIL			
2	LEE COOPER, SHELDON	lcsheldon@uc3m.es			
3	WOLOWITZ, HOWARD	whoward@uc3m.es			
4	HOFSTADTER, LEONARD	hleonard@uc3m.es			
5					
6					
7					
8					

Figura 5.18: Archivo Profesores

Básicamente consiste en un archivo compuesto de dos columnas, una primera, donde debe aparecer el nombre y los apellidos del profesor en orden inverso al mencionado y separados por una coma (*apellido, nombre*), y una segunda, donde debe encontrarse el correo electrónico (que será lo que pasará a ser el *login*) de dicho profesor.

Hay que tener en cuenta que la primera línea no vacía del archivo debe contener las cabeceras de las columnas *NOMBRE* e *EMAIL*, tal y como aparece en la figura 5.18, ya que en caso de no aparecer, el primer profesor del fichero sería obviado.

El siguiente archivo que vamos a analizar, que vemos en la figura 5.19, es el que se corresponde con un conjunto de alumnos correspondientes a un grupo de una asignatura.



	A	B	C	D
1	DNI	NOMBRE	LOGIN	EMAIL
2	11111111A	ABCD DZFG, PABLO	100292295	100292295@uc3m.es
3	22222222M	AAAA BBBB, MANUEL	100291481	100291481@alumnos.uc3m.es
4				
5				
6				

Figura 5.19: Archivo Grupo Alumno

Cada línea del archivo se corresponde con un alumno (salvo la primera que, como en el caso del archivo anterior, debe ser de cabeceras *DNI, NOMBRE, LOGIN* e *EMAIL*, si no queremos que se omita el primer alumno del grupo). Cada línea está compuesta por cuatro parámetros *DNI, NOMBRE* (*apellido, nombre*), *LOGIN* e *EMAIL*, que deben encontrarse estrictamente en ese orden para un correcto funcionamiento (cada parámetro debe estar contenido en un única celda).

NOTA: Aunque la información del DNI es una columna exigida, esta información no será introducida en la base de datos. Se ha exigido esta columna, porque se han tomado como referencia ficheros de cursos pasados.

Otro archivo que encontramos necesario es el que vemos en la figura 5.20, aquel en el que se introduce la planificación de una asignatura, es decir, aquel en el que encontramos los diferentes exámenes programados para la asignatura.

	A	B	C	D	E	F	G	H	I
1	individual	grupal	formativo	adicional					
2									
3	e1	grupal		5					
4	e2	individual		6					
5									
6									
7									
8									

Figura 5.20: Archivo Planificación Asignatura

Este archivo contiene dos zonas claramente diferenciadas por una línea en blanco. Podemos apreciar que en ninguna de las zonas se contiene una primera línea cabecera como en los ficheros anteriores. La primera zona está compuesta por una única línea donde se encuentren las distintas tipologías permitidas para los exámenes de la asignatura (una por celda y sin límite). Esto se ha realizado de este modo ya que para poder comprobar que los exámenes tienen la tipología adecuada, debe estar creada primero. La segunda zona, es la zona de los exámenes, en ella encontramos que cada línea corresponde con un examen, donde cada uno estará compuesto por 3 parámetros característicos *Nombre*, *Tipología* y *Puntuación Máxima* (parámetro=celda).

Otro fichero es aquel que necesitamos a la hora de introducir en bloque las notas de los alumnos. En esta acción encontramos dos tipos de ficheros distintos, uno, para el caso en el que el examen tiene apartados y queremos reflejar las notas en el sistema (figura 5.21), y otro, para las ocasiones en las que el examen no tenga apartados o el profesor simplemente no quiera dejar reflejada dicha información (figura 5.22).

En el archivo de la figura 5.21, tras la primera línea de cabeceras, encontramos una combinación de 2 columnas por línea: login (alumno) y nota final.

Sin embargo, el otro tipo de archivo, se trata de un archivo mucho más complejo, ya que debe poseer dos zonas (ambas con cabeceras, tal y como vemos en la figura 5.22) separadas por una línea en blanco. La primera zona se corresponde con la definición de los apartados del examen (apartado=línea) y la segunda es aquella en la para cada alumno aparecen las notas, de los distintos apartados y la global, del examen.

	A	B	C	D	E	F	G	H	I
1	LOGIN	NOTAS							
2	100291749	7							
3									
4									
5									
6									
7									
8									

Figura 5.21: Archivo Notas examen sin apartados

	A	B	C	D	E	F	G	H	I
1	NOMBRE	NOTA MAX	MATERIA	WEB					
2	c1	1.5	perro	asd					
3	c2	1.5	vaca	sdf					
4	p1	4	oveja	dfg					
5									
6	LOGIN	c1	c2	p1	TOTAL				
7	100291749	1	0.5	2	3,5				
8									
9									

Figura 5.22: Archivo Notas examen con apartados

Uno de los archivos más importantes, y por lo que se planteó todo el proyecto, es aquel en el que se personaliza la forma de evaluar la asignatura. Al igual que en el caso anterior, encontramos varios tipos de ficheros distintos para una misma acción (figura 5.23 y figura 5.24).

La diferencia consiste en si se trata de una fórmula pura y dura, que ocupará una única celda (figura 5.24), o si se tratará de una evaluación limitada a ciertas condiciones, tal y como vemos en figura 5.23, donde la primera celda será ocupada por un IF; la segunda, por la condición; seguida del cálculo a hacer en caso de que se cumpla, si no se buscará donde se encuentra el ELSE y se ejecutará el cálculo que lo continúa.

IMPORTANTE: Como se indicó anteriormente, será necesario que cuando se personalice una evaluación las fórmulas que se escriban mantengan los nombres de las tipologías como variables, si no el sistema será incapaz de resolver las ecuaciones/condiciones planteadas.

NOTA: Siempre que se intente personalizar condicionalmente una evaluación se comprueba que existe un ELSE.

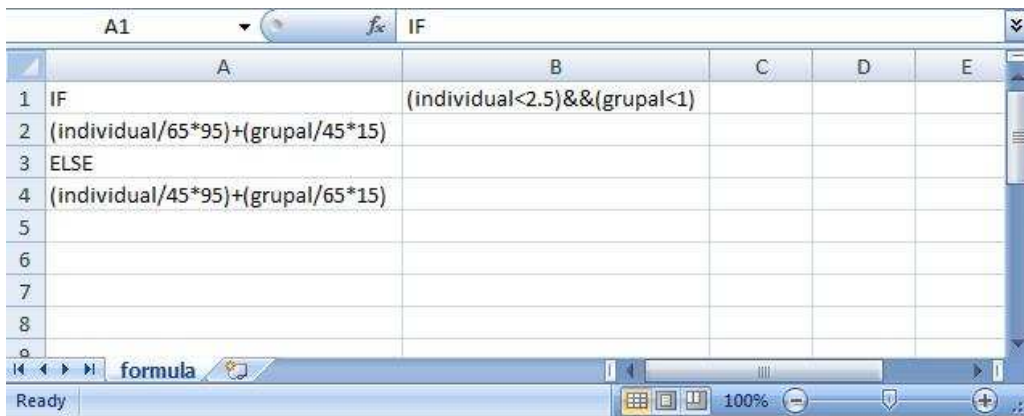


Figura 5.23: Archivo fórmula evaluación simple personalizada asignatura

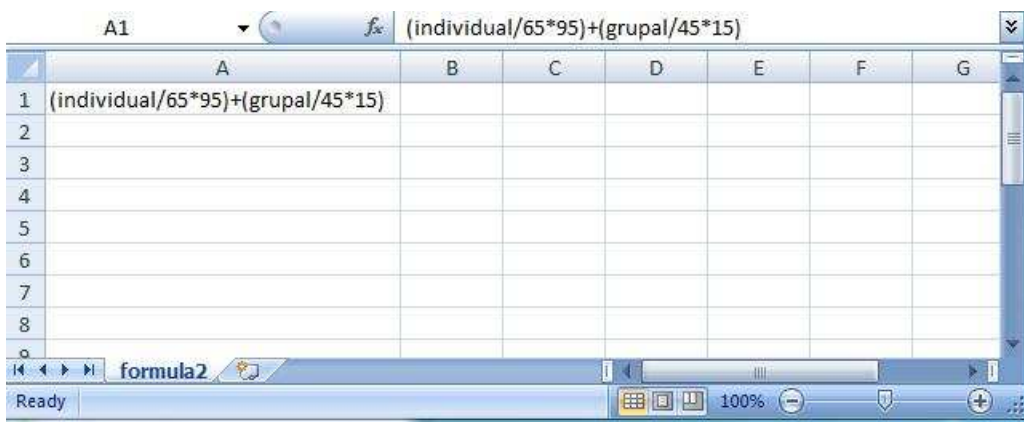


Figura 5.24: Archivo fórmula evaluación condicionada personalizada asignatura

Como último archivo a analizar, que vemos en la figura 5.25, tenemos aquel que nos devuelve el sistema como copia de seguridad.

	A	B	C	D	E	F	G	H	I	J	K
1	NIA	NOMBRE	APELLIDO	e2	e3	e1	Temporalidad	A/S Temporal	Nota Final	Aprobado/Suspenso	
2	100291749	Pepito	Asdf Asdf	1.5	-	3,5	-	Aprobado	7.7	Aprobado	
3	100039003	Sheresita	Fraille Paniagua	-	-	-	-	-	-	-	
4											
5											
6											
7											
8											

Figura 5.25: Archivo download

En él, se nos muestra toda la información importante concerniente a una asignatura, contenida en la base de datos para un determinado grupo de alumnos, es decir, la nota de cada uno de los exámenes de la asignatura y las notas temporales y finales asociadas.

NOTA: Aunque se tengan los apartados de los exámenes, solo se mostrará la nota final del examen, que es la que se emplea para el cálculo de la evaluación y que podemos por lo tanto

catalogar de importante.

Finalmente queda decir que se ha elegido un formato *csv* para los ficheros, por la facilidad con la que leemos el archivo, ya que se trata de un texto plano, donde los parámetros (normalmente columnas de un archivo excel) están separados por ;.

En el presente capítulo detallaremos las pruebas que hemos realizado al sistema para confirmar que los diferentes requisitos enunciados en el Capítulo 4 se ven cumplidos, además utilizaremos distintas capturas de pantallas que ayudarán a su comprensión.

El equipo que se ha necesitado para el test de la aplicación ha sido un ordenador con conexión a Internet y capacidad para abrir un navegador.

Identificaremos los diferentes requisitos, que en cada caso, se verifican introduciendo al final del párrafo entre corchetes con una R seguida del número que dimos a cada requisito en el Capítulo 4.

6.1. Resultados obtenidos

Una de las pruebas consistió en ver la independencia de la aplicación web frente al navegador que utilizamos normalmente, habiendo realizado la misma con los programas Internet Explorer, Mozilla Firefox y Google Chrome, obteniendo resultados positivos.

Para poder acceder a la zona privada de usuario, será necesario realizar *login*, tal y como vemos en la figura 6.1, que nos muestra la página de inicio de la plataforma web [R1].

Inicialmente, si no se ha ingresado anteriormente en el sistema, es muy posible que el usuario no conozca su contraseña en el sistema. Con esa finalidad y la de un posible olvido, en la página anteriormente comentada, se ha creado un enlace *Olvidó su contraseña?* donde se redirige al usuario a la página de recuperación de contraseñas que vemos en la figura 6.2. Una vez que hemos iniciado, el sistema nos redigirá dependiendo del tipo de usuario que seamos, si no se mostrará un mensaje indicando que el usuario no está registrado [R7].

En el caso de que el usuario sea el administrador, se le redigirá a la única página a la que tiene un acceso directo, que es aquella que se nos muestra, en la figura 6.3, que permite la inserción en bloque de un grupo de profesores (para acceder a las otras páginas tendrá que realizar una llamada explícita desde el navegador) [R2].



Figura 6.1: Página Acceso

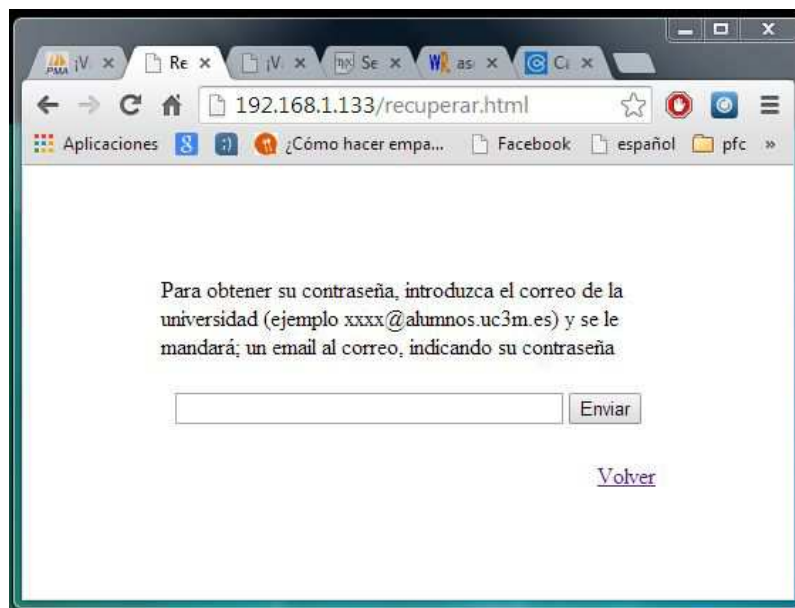


Figura 6.2: Página recuperacion

Al tratarse de la primera vez que entra el usuario (no administrador) en el sistema, nos redirigirá a la pantalla de cambio de contraseña (figura 6.4) para que se cambie la contraseña que la aplicación otorgó al usuario cuando sus datos se introdujeron en la base de datos, ya que hasta que no se realice esta acción la aplicación no nos permitirá el acceso a ninguna otra página, siendo siempre redirigidos a ella al iniciar. A esta página se tiene acceso, sin necesidad de encontrarse autenticado en el sistema, por lo que el cambio de contraseña puede realizarse en cualquier momento. Una vez realizado el paso anterior, el usuario ya tendrá acceso a su área privada [R32, R4].

Dependiendo del tipo de usuario (profesor o alumno) el acceso a nuestra cuenta tendrá una forma distinta. Por ello, a partir de este punto hablaremos de los resultados obtenidos, realizando una división en los mismos, por un lado la parte del profesor y por otro lado la parte del cliente.

NOTA: El sistema de verificación nos garantiza el acceso a la página por el usuario destinado a ello [R31].

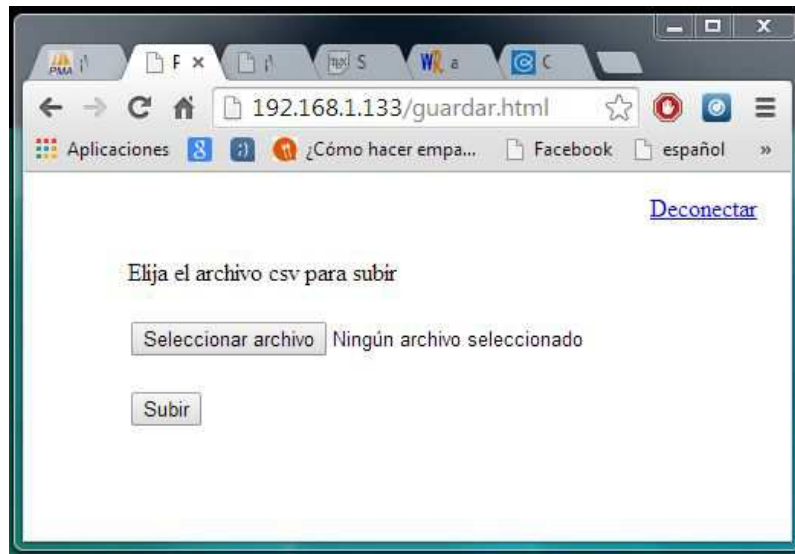


Figura 6.3: Página guardar

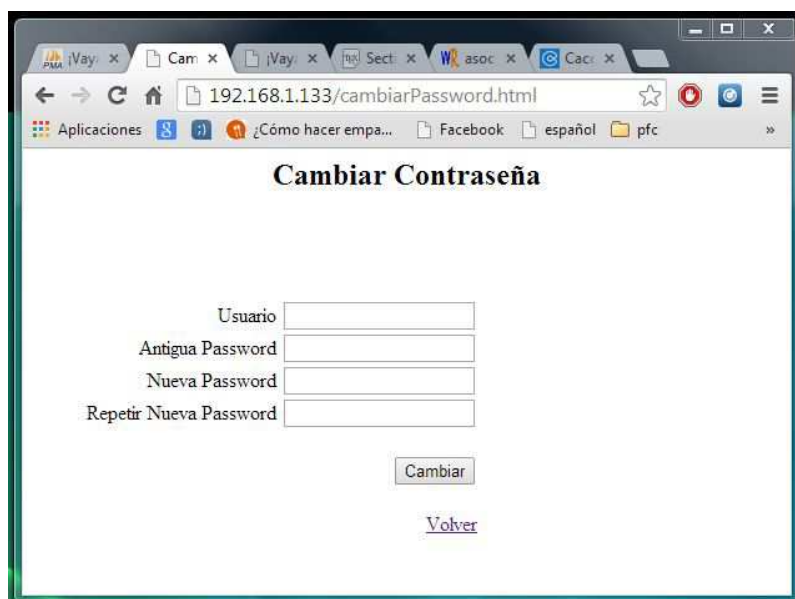


Figura 6.4: Página cambiarPassword

6.1.1. Profesor

Para el caso del profesor se trata de la página que contiene un menú con un conjunto de enlaces que se corresponden con cada una de las funcionalidades creadas para él (véase figura 6.5).

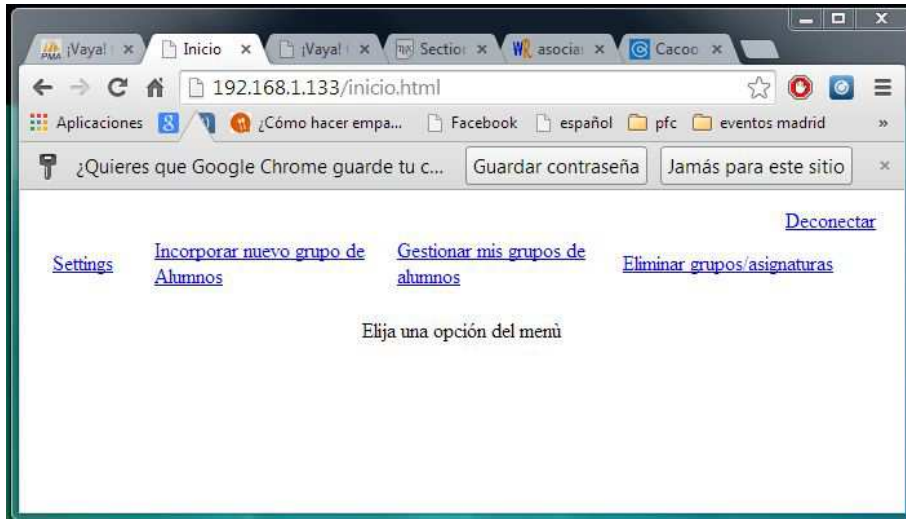


Figura 6.5: Pagina inicio

El primer enlace que encuentra el profesor, *Settings*, es aquel que permite modificar sus datos de usuario, incluyendo contraseña, tal y como podemos ver en la figura 6.6 [R3, R4, R9].

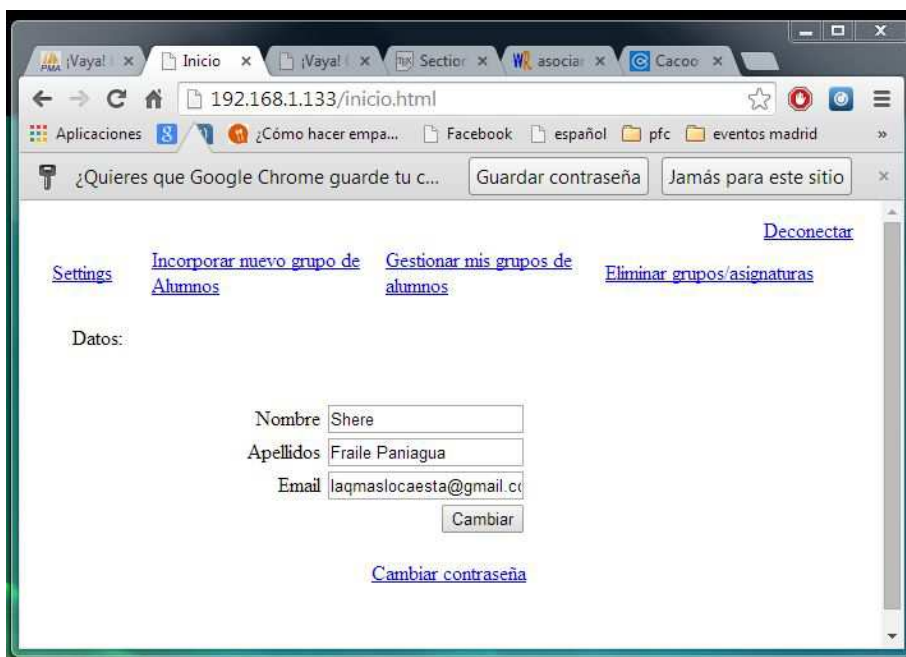


Figura 6.6: Vista Settings

Si, por el contrario, lo que el profesor quiere hacer es añadir un nuevo grupo de alumnos

que gestionar podrá hacerlo a través del enlace *Incorporar nuevo grupo de alumnos*. En esta opción se nos mostrará una lista con todas las asignaturas en las que el profesor participa, junto con la opción de añadir una nueva, *Crear nueva asignatura* (véase 6.7) [R10].

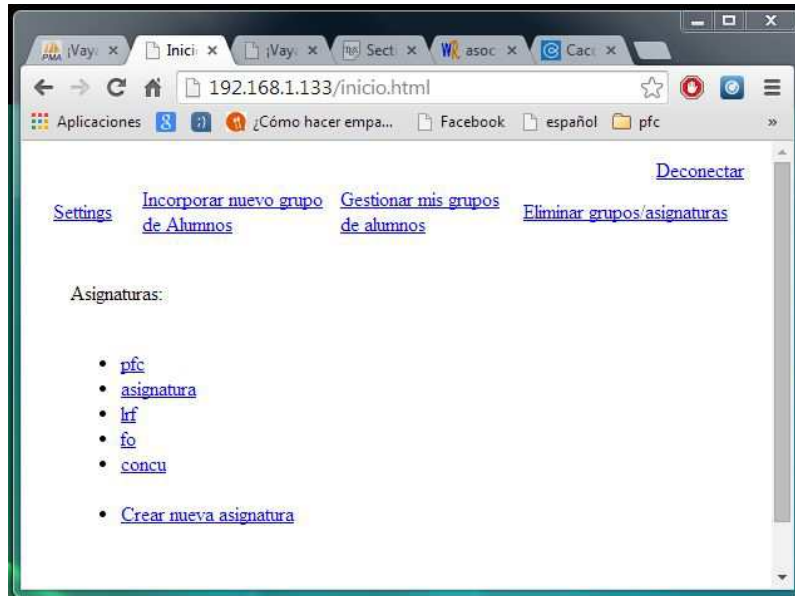


Figura 6.7: Vista Incorporar nuevo grupo de Alumnos

Para que una asignatura exista en esa vista, previamente, deberá haber sido creada (*Crear nueva asignatura*), ya sea porque el profesor la introduzca de cero en el sistema (figura 6.8), o porque cuando intentó crearla, el sistema encontró que ya existía y simplemente se la asignó.



Figura 6.8: Vista Crear una nueva asignatura

Pero tener una asignatura asignada no implica que tengamos gestión sobre sus grupos, es más, si se ha creado de cero, no existirá ningún grupo. Obtendremos información sobre

si existen grupos, cuales son y si tenemos gestión sobre ellos cuando pulsemos sobre una de las asignaturas (ejemplo en la figura 6.9), permitiéndonos a su vez la posibilidad de añadir la gestión de grupos sobre los cuales no la teníamos al pulsar sobre *Asignar*. Además, se posibilita la creación de un nuevo grupo para esa asignatura (*Crear nuevo grupo* - figura 6.10) [R6, R11].



Figura 6.9: Visualización grupos existentes de una asignatura



Figura 6.10: Vista Crear nuevo grupo

Si lo que el profesor desea es gestionar uno de sus grupos, deberá acceder al enlace *Gestionar mis grupos de alumnos*. En este caso, como en la vista anterior, también tendremos una lista con las asignaturas, pero particularizando los grupo sobre los que tiene gestión, tal y como se ve en la figura 6.11 [R12].

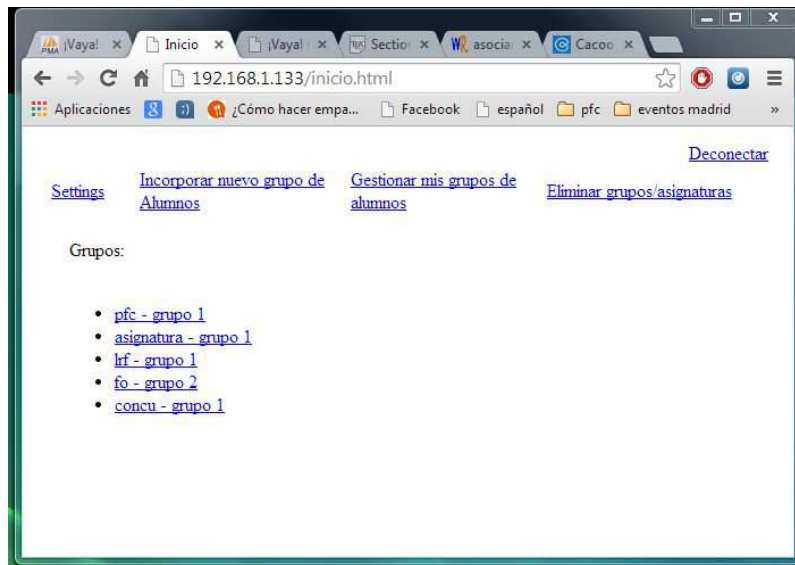


Figura 6.11: Visualización gestionar grupos

Al pulsar sobre uno de ellos, un submenú con las funciones aplicables al grupo aparecerá tal y como se ve en la figura 6.12, donde el efecto de pulsar en *Mostrar alumnos*, que consiste en mostrar todos los alumnos del grupo, ya está incorporado en la propia vista de inicio del submenú. Podemos encontrar en esta funcionalidad un enlace *Ver* al final de cada alumno, cuya finalidad es visualizar todas las notas de los exámenes programados en la asignatura, para poder modificarlas (ver figura 6.13) [R13, R18, R15].

NOTA: Los exámenes que contienen un enlace en el nombre son aquellos que tienen apartados. Para poder visionar los apartados y modificar la nota del examen, deben ser desplegados pulsando sobre el nombre.



Figura 6.12: Visualización alumnos del grupo



Figura 6.13: Visualización notas alumno

Aunque como se ve en la figura hay un total de siete subfuncionalidades, sólo vamos a destacar mediante ilustración explicativa aquellas en cuya vista se encuentran nuevas acciones (se hicieron las pruebas oportunas y el resto de opciones tenía un funcionamiento correcto: añadir alumno, borrar alumno y descargar archivo con la copia de seguridad). Concretamente se trata de *Planificación* y *Evaluación*, que son acciones claves donde se encuentra la información de la asignatura tras su creación [R14, R16, R22, R33].

Para el caso más simple, el de *Evaluación*, cuya vista se muestra en la figura 6.14, la opción extra que se nos da, aparte de características (mostrar tipo de evaluación y la nota de evaluación de los alumnos del grupo implicado), es la de poder personalizar la evaluación pulsando sobre *Cambiar* [R20, R23, R24].

NOTA: Aunque la evaluación ya haya sido personalizada, se sigue permitiendo el cambio a una nueva evaluación personalizada, facilitando poder cambiar en cualquier momento las formas de evaluar la asignatura.

Para el caso de *Planificación*, al seleccionar esta opción se muestra la información sobre cuántos exámenes planificados tiene la asignatura y qué características poseen, siendo posible además modificar esta información mediante los diferentes enlaces (*Crear nuevo examen*, *Borrar*, *Editar*), tal y como se aprecia en la figura 6.15 [R21].

*NOTA: La opción de **Crear nueva tipología** ha sido diseñada ante la posibilidad de necesitar una nueva tipología para un examen o de haber olvidado en la introducción de la planificación algún tipo.*

Settings [Incorporar nuevo grupo de Alumnos](#) [Gestionar mis grupos de alumnos](#) [Eliminar grupos/asignaturas](#) [Deconectar](#)

asignatura - Grupo 1

[Mostrar alumnos](#) [Incorporar nuevo alumno](#) [Eliminar alumno](#) [Introducir notas](#) [Planificación](#) [Evaluación](#) [Download](#)

Evaluación

Actualmente, el tipo de evaluación con el que se califica esta asignatura es: Configuración Personalizada

[Cambiar](#)

La nota ha sido calculada con los exámenes realizados hasta el momento

NIA	Nombre	Apellidos	Temporalidad	A/S Temporal	Nota Final	Aprobado/Suspenseo
100039003	Sheresita	Fraile Paniagua	5.1	Aprobado	5.1	Aprobado

Figura 6.14: Vista Evaluación

Settings [Incorporar nuevo grupo de Alumnos](#) [Gestionar mis grupos de alumnos](#) [Eliminar grupos/asignaturas](#) [Deconectar](#)

asignatura - Grupo 1

[Mostrar alumnos](#) [Incorporar nuevo alumno](#) [Eliminar alumno](#) [Introducir notas](#) [Planificación](#) [Evaluación](#) [Download](#)

Exámenes planificados

Nombre	Tipología	Puntuación	Acciones
e2	grupal	5	Borrar Editar Ver notas
e3	formativo	4	Borrar Editar Ver notas
e1	individual	7	Borrar Editar Ver notas

[Crear nuevo examen](#)

La tipología permitida es: individual; formativo; grupal [Crear nueva tipología](#)

Figura 6.15: Vista Planificación

Al pulsar sobre **Ver notas**, obtendremos la nota del examen de todos los alumnos. En caso de poseer apartados, aparte de ser mostrada dicha nota, se mostrará la información correspondiente a los apartados, permitiendo además su modificación, tal y como se muestra en la figura 6.16 [R19, R17].

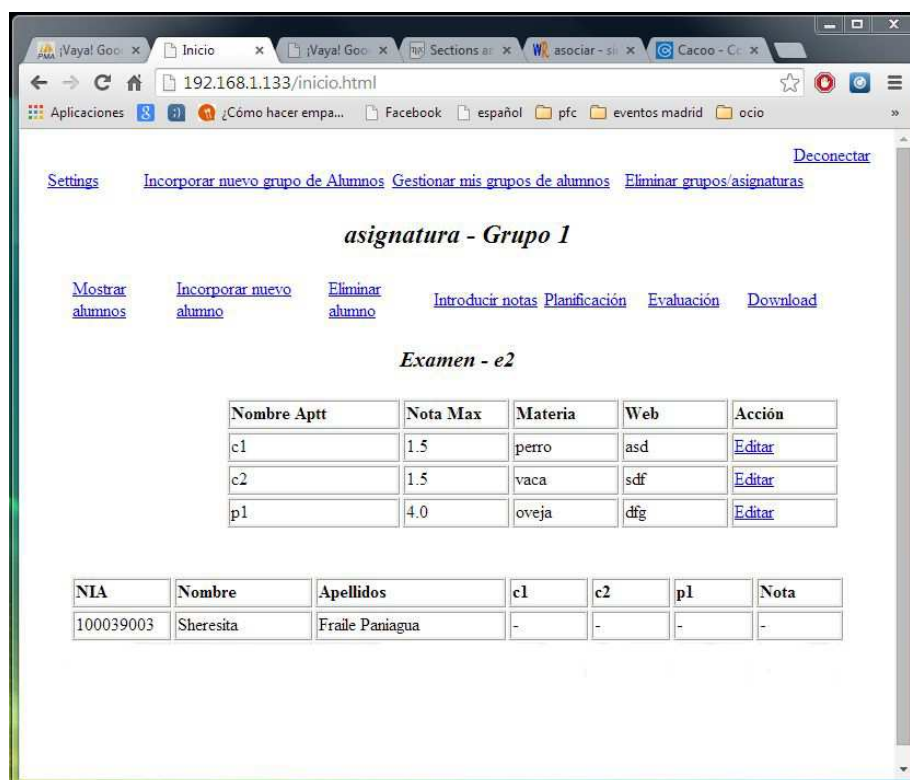


Figura 6.16: Visualización notas examen

*NOTA: En **Introducir notas**, al finalizar la acción, obtendremos el mismo resultado que si pulsamos sobre **Ver notas** en planificación [R19, R17].*

La última opción del menú principal, *Eliminar grupos/asignaturas*, mediante un formulario multiselección nos permitirá borrar todas aquellos grupos mal creados, confundidos, cuya evaluación ya haya finalizado..., siempre y cuando tengamos gestión sobre ellos (ver figura 6.17) [R25].

*NOTA: Las asignaturas sobre las que no tengamos grupos con gestión aparecerán con el calificativo **Grupo 0**.*

Como último comentario sobre la vista correspondiente al profesor, *inicio.html*, indicar que en todo momento podemos encontrar el menú con las funcionalidades principales, *Settings*, etc., tal y como hemos visto en las distintas ilustraciones.

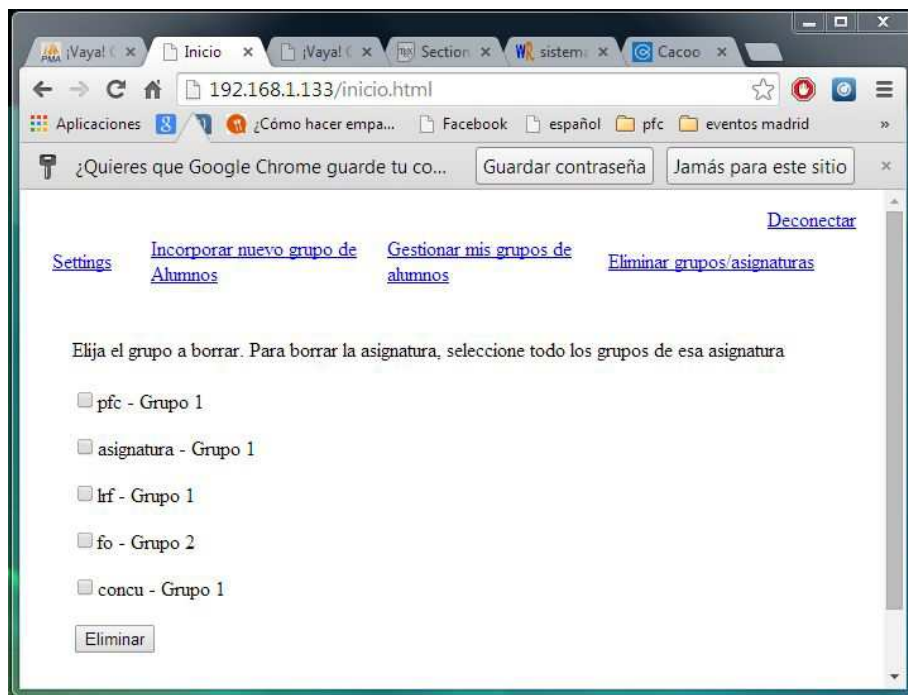


Figura 6.17: Vista Eliminar Grupo

6.1.2. Cliente

Para el caso del alumno, la vista inicial consistirá en una lista de enlaces que se corresponderán con las asignaturas en las que el alumno esté matriculado (siempre y cuando el profesor esté utilizando este sistema), además de la opción desconectar que comentaremos al final [R26].



Figura 6.18: Página menu - Listado de asignaturas

Una vez seleccionada una asignatura de la lista, se mostrará toda la información correspondiente a dicha asignatura tal y como se muestra en la figura 6.19: una primera parte, con los profesores que imparten dicha asignatura; una segunda, con los exámenes planificados de la asignatura con todas sus características y nota correspondiente al alumno en caso de tener; una tercera, con la evaluación de la asignatura; y una final, con la posición que ocupa el alumno en el ranking de la asignatura (siempre y cuando se haya realizado algún examen de la asignatura) [R27, R28, R29, R30].

[Deconectar](#)

Aula Virtual

lrf - Grupo 1

Los profesores asociado a esta asignatura son:

- Prof. Shere Fraile Paniagua, laqmaslocaesta@gmail.com

Exámenes:

EXAMEN	TIPOLOGIA	NOTA MAX	NOTA
e1	individual	5	-

La nota ha sido calculada con los exámenes realizados hasta el momento

NIA	Nombre	Apellidos	Temporalidad	A/S Temporal	Nota Final	Aprobado/Suspenso
100039003	Sheresita	Fraile Paniagua	-	-	-	-

Ranking: No procede calcular un ranking, puesto que aún no tenemos ninguna nota

[Volver](#)

Figura 6.19: Página menu - Información asignatura

Para terminar, cabe señalar que tanto en la vista para el profesor, como en la vista del alumno, en la parte superior derecha se encuentra la opción *Desconectar*, que permite al usuario cerrar sesión en cualquier momento. Si dejamos inactivo el sistema por más de 30 minutos, al tratar de realizar una función, el sistema nos redirige a la página de inicio, ya que la sesión habrá expirado [R5, R8].

6.2. Conclusiones

Una vez realizadas las comprobaciones pertinentes, obtenemos resultados favorables acerca del funcionamiento de la aplicación, verificándose todos los requisitos que al co-

mienzo del desarrollo se fijaban, y por lo tanto, podemos concluir que el funcionamiento de la aplicación desarrollada es correcto, no presentando problemas en las diferentes acciones del usuario.

Conclusiones y Trabajo futuro

7.1. Conclusiones

Esta experiencia ha mostrado cómo se puede diseñar e implementar una aplicación Cliente-Servidor basado en tres niveles.

En ella, se ha tratado de crear un sistema cuyas características principales sean el fácil manejo y una alta flexibilidad (derivada del empleo de archivos). Pero no será hasta que nuestra aplicación se utilice en un ámbito real, cuando definitivamente podamos asegurar que se cumplen estas expectativas.

Asimismo, este trabajo sirvió para interiorizar y aprender de una manera adecuada como se realiza un proyecto, es decir, para tener una idea más formada de cuál será el trabajo a que nos enfrentaremos cuando salgamos al mercado laboral. Además, ha permitido afianzar los conocimientos adquiridos durante la carrera.

Finalmente, cabe destacar que, gracias a la realización de un estudio previo con ejemplos prácticos, nuestro trabajo a la hora de realizar el proyecto ha sido menor, puesto que, la mayoría de las complicaciones se dieron durante esa etapa.

7.2. Trabajo futuro

Una vez realizada la aplicación prototipo que cuenta con la funcionalidad deseada, el siguiente paso es mejorar la misma por medio de nuevas opciones o modificaciones de las funcionalidades ya existentes.

Así pues, podemos enumerar algunas de las posibles líneas de trabajo futuro que se pueden basar en este proyecto:

- Mejora en la apariencia.

Aunque nuestro sistema muestra la información clara y ordenada, será necesario la aplicación de un estilo CSS que le dé una apariencia fresca, atractiva y fácil de usar.

- Incorporación de funcionalidades.
 - Añadir la posibilidad de cambiar las notas en la vista grupal de las notas del examen.
 - Agregar un acceso directo al cambio de contraseña en la vista del alumno.
 - Ordenar alfabéticamente los listados.
 - Permitir al alumnos visualizar sus notas por tipología.
- Traducción a distintos idiomas.
- Realización de un sistema alumno más visual mediante el empleo de gráficas.

Bibliografía

- [Ado] Adobe web-site: <http://www.adobe.com/>, Consultado: 15/4/2013.
- [AJA] AJAX web-site: <https://developer.mozilla.org/en-US/docs/AJAX>, Consultado: 13/4/2013.
- [Apa] Apache web-site: <http://apache.org>, Consultado: 3/3/2013.
- [con] Configuración avanzada Apache web-site: luismido.wikidot.com/apache2-configuracion-avanzada, Consultado: 3/10/2013.
- [DB2] DB2 web-site: www.ibm.com/software/data/db2/, Consultado: 11/2/2013.
- [GSO] GSON web-site: code.google.com/p/google-gson/, Consultado: 3/3/2013.
- [IIS] IIS web-site: www.iis.net/, Consultado: 10/4/2013.
- [Java] JavaScript web-site: www.ecmascript.org/, Consultado: 1/4/2013.
- [Javb] JavaMail web-site: javamail.java.net/, Consultado: 20/5/2014.
- [JSO] JSON web-site: www.json.org/, Consultado: 30/4/2013.
- [MyS] MySQL web-site: www.mysql.com, Consultado: 10/2/2013.
- [Ora] Oracle web-site: www.oracle.com/es/index.html, Consultado: 8/7/2013.
- [PHP] PHP web-site: www.php.net/, Consultado: 25/3/2013.
- [Pos] PostgreSQL web-site: www.postgresql.org.es, Consultado: 12/2/2013.
- [SQL] SQL Server web-site: www.microsoft.com/sqlserver/, Consultado: 3/6/2013.
- [Sym] Symfony web-site: www.Symfony.com, Consultado: 3/6/2013.

- [Tom] Apache-Tomcat web-site: tomcat.apache.org/download-70.cgi, Consultado: 18/7/2013.
- [W3] W3 web-site: www.w3.org/, Consultado: 9/5/2013.
- [Web] WebLogic web-site: www.oracle.com/.../weblogic/overview/index.html, Consultado: 2/3/2013.
- [Wik] Wikipedia web-site: www.wikipedia.org/, Consultado: 1/4/2013.
- [Yam] YAML web-site: www.yaml.org/, Consultado: 5/4/2013.

Presupuesto del proyecto

En este apéndice se presentan justificados los costes globales de la realización de este Proyecto Fin de Carrera. Tales costes, imputables a gastos de personal y de material, se pueden deducir de las Tablas A.1 y A.2.

Tabla A.1: *Fases del Proyecto*

Fase 1	<i>Documentación</i>	350 horas
Fase 2	<i>Desarrollo del software</i>	90 horas
Fase 3	<i>Análisis de la base de datos</i>	500 horas
Fase 4	<i>Redacción de la memoria del proyecto</i>	250 horas

En la Tabla A.1 se muestran las fases del proyecto y el tiempo aproximado para cada una de ellas. Así pues, se desprende que el tiempo total dedicado por el proyectando ha sido de 1.190 horas, de las cuales aproximadamente un 30 % han sido compartidas con el tutor del proyecto, por lo que el total asciende a 1.547 horas. Teniendo en cuenta que la tabla de honorarios del Colegio Oficial de Ingenieros Técnicos de Telecomunicación establece unas tarifas de 60 /hora, el coste de personal se sitúa en 92.820 .

Tabla A.2: *Costes de material*

<i>Ordenador de gama media</i>	1.300
<i>Local (durante 12 meses, con un coste de 120 /mes)</i>	1.440
<i>Documentación</i>	200
<i>Gastos varios</i>	700

En la Tabla A.2 se recogen los costes de material desglosados en equipo informático, local de trabajo, documentación y gastos varios no atribuibles (material fungible, llamadas telefónicas, desplazamientos...). Ascenden, pues, a un total de 3.640 .

A partir de estos datos, el presupuesto total es el mostrado en la Tabla A.3.

Tabla A.3: *Presupuesto*

Concepto	Importe
Costes personal	78.000
Costes material	3.640
Base imponible	96.460
I.V.A. (16 %)	15.433,6
TOTAL	111.893,6