Treball de Fi de Grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques
Universitat de Barcelona

SSISTEMA DE CORRECCIÓ DE PROGRAMES: PREPARACIÓ PER A UN ENTORN DE PRODUCCIÓ

Jordi Serral Zamora

Director: Lluís Garrido Ostermann
Realitzat a: Departament de Matemàtica Aplicada i Anàlisi. UB

Barcelona, 17 de Gener de 2013

Agradecimientos

Antes de iniciar la lectura de la memoria de este proyecto me gustaría agradecer a un conjunto de personas su ayuda a la hora de realizar este proyecto.

- Primero de todo, a mi tutor Lluís Garrido por su dirección y soporte en este proyecto.
- Segundo, a mi familia por animarme en esos momentos de nerviosismo.
- Tercero y último, a mis compañeros de la Facultad. Con los cuales he compartido grandes momentos con la realización de nuestros proyectos. Y que han servido de gran apoyo y ayuda.

Resumen

La idea de este proyecto surgió durante la implantación del Grado de Informática. Este grado se enmarca en el Espacio Europeo de Educación Superior (EEES), el cual potencia la evaluación continuada del alumno. Para ello el alumno está usualmente obligado a realizar un mayor número de prácticas y por lo tanto se incrementa la necesidad del alumno de aprender de forma autónoma. También se incrementa la carga de trabajo de corrección del profesor dado el alto número de prácticas por corregir.

La principal idea del Sistema de Corrección de Programas (SCP) es ayudar al alumno a la hora de realizar sus entregas, dotándole de autonomía en la corrección de estas y beneficiando el hecho de recibir una respuesta directa tras realizar la entrega. Además, también se beneficia al profesor evitando el largo proceso de corrección de estas y la carga de trabajo que ello supone.

La funcionalidad principal de la aplicación SCP es la auto-corrección de las prácticas realizadas por los alumnos, con lo que el alumno podrá realizar la entrega de un programa y conocer al momento si este está correcto o no. El profesor podrá visionar los resultados y código realizado por el alumno a través del sistema. De esta forma se evita así el proceso de corrección manual en que el profesor ha de compilar y ejecutar el código enviado por los alumnos.

Este proyecto es la continuación de otros dos proyectos realizados por alumnos de la UB. El objetivo principal en este proyecto es la de dotar al SCP de mayor robustez i nuevas funcionalidades para permitir que la aplicación funcione de forma robusta con un gran número de usuarios.

En éste documento se podrá encontrar la información necesaria del proceso de implementación de este proyecto, así como el análisis realizado de la versión anterior y las tecnologías utilizadas.

Resum

La idea d'aquest projecte va sorgir durant la implantació del Grau d'Informàtica. Aquest grau s'engloba en l'Espai Europeu d'Educació Superior (EEES), el qual potencia l'avaluació continuada de l'alumne. Podem trobar que l'alumne està usualment obligat a realitzar un major nombre de pràctiques i per tant s'incrementa la necessitat de l'alumne d'aprendre de forma autònoma. També s'incrementa la càrrega de treball de correcció del professor donat l'alt nombre de pràctiques per corregir.

La principal idea del Sistema de Correcció de Programes (SCP) és ajudar l'alumne a l'hora de realitzar les seves entregues, dotant d'autonomia en la correcció d'aquestes i beneficiant el fet de rebre una resposta directa després de fer el lliurament. A més, també es beneficia el professor evitant el llarg procés de correcció d'aquestes i la càrrega de treball que això suposa.

La funcionalitat principal de l'aplicació SCP és l'autocorrecció de les pràctiques realitzades pels alumnes, de manera que l'alumne podrà realitzar el lliurament d'un programa i conèixer al moment si aquest està correcte o no. El professor podrà visionar els resultats i codi realitzat per l'alumne a través del sistema. D'aquesta manera s'evita així el procés de correcció manual en què el professor ha de compilar i executar el codi enviat pels alumnes.

Aquest projecte és la continuació de dos projectes realitzats per alumnes de la UB. L'objectiu principal en aquest projecte és la de dotar el SCP de major robustesa i noves funcionalitats per permetre que l'aplicació funcioni de forma robusta amb un gran nombre d'usuaris.

En aquest document es podrà trobar la informació necessària del procés d'implementació d'aquest projecte, així com l'anàlisi realitzada de la versió anterior i les tecnologies utilitzades.

Abstract

The idea for this project came during the implementation of the Degree of Computing at the University of Barcelona. This degree is framed in the European Higher Education Area (EHEA), which enhances the student's continuous assessment. The students are usually required to perform a greater number of practices and therefore increases the student's need to learn autonomously. It also increases the workload of the teacher correction given the high number of practices to review.

The main idea Correction System Program (SCP) is to help students when making their deliveries, giving it autonomy in their review and benefiting the fact of receiving a direct answer after making the delivery. In addition, the teacher also benefits by avoiding the hard process of correcting these and the workload involved.

The main application functionality SCP is self-review practices undertaken by the students, so that students can make the delivery of a program and know at the moment whether it's correct or not. The teacher is able to watch the results and the student code through the system, avoiding the manual correction process in which the teacher has to compile and run the code sent by students.

This project is a continuation of two other projects made by students of the UB. The main goal of this project is to provide more robustness to the SCP and add new functionalities to allow the application to work smoothly with large numbers of users.

In this document you can find the necessary information of the implementation process of this project, as well as the analysis of the previous version and the technologies that has been used.

Índice de contenido

1. Introducción 9

1.1 Motivación 9

1.2 TFG Previos 9

1.3 Objetivo General 10

2. Análisis del SCP anterior 11

2.1 Base de Datos 12

2.1.1 Análisis 13

2.1.2 Problemas 15

2.2 Vista Administrador 15

2.2.1 Problemas 17

2.3 Vista Alumno 17

2.3.1 Problemas 18

2.4 Vista Profesor 19

2.4.1 Problemas 22

2.2 Objetivos de la nueva versión de SCP 23

3. Tecnologías utilizadas 24

3.1 Hibernate 24

3.2 Spring 26

3.3 MySQL 27

3.4 Java Server Faces (JSF) 28

4. Especificaciones de requerimientos 29

4.1 Actores y Funcionalidades 29

4.1.1 Funcionalidades Administrador. 29

4.1.2 Funcionalidades Alumno 29

4.1.3 Funcionalidades Profesor. 30

4.2 Casos de Uso. 30

4.2.1 Casos de Uso Administrador. 31

4.2.2 Casos de Uso de Alumno 40

4.2.3 Casos de Uso del Profesor 43

4.2.4 Casos de Uso Comunes 49

5. Aplicación. 51

5.1 Estructura Clases Java 51

5.2 Diagrama de Clases ... **56**
5.2.1 Diagrama de Clases Administrador ... 57
5.2.2 Diagrama de Clases Alumno ... 58
5.2.3 Diagrama de Clases Profesor ... 60
5.3 Base de Datos ... **62**
5.3.1 Modelo E-R ... 62
5.3.2 Comentarios. ... 63
5.4 Mapa de Navegación ... **64**
6. Pruebas y resultados ... **65**
6.1 Pruebas con distintos navegadores ... **65**
6.1.1 Internet Explorer ... 65
6.1.2 Google Chrome ... 66
6.1.3 Mozilla Firefox ... 66
6.2 Pruebas de vistas ... **67**
6.2.1 *Login* ... 67
6.2.2 Vista Administrador ... 67
6.2.3 Vista Alumno ... 69
6.2.4 Vista Profesor ... 69
7. Dificultades ... **71**
8. Conclusiones ... **72**
9. Bibliografía ... **75**
10. Anexos ... **76**
10.1 Manual de instalación ... **76**
10.2 Manual Administrador ... **78**
10.3 Manual Alumno ... **80**
10.4 Manual Profesor ... **82**

1. Introducción

1.1 Motivación

Desde la implantación del Grado de Ingeniería Informática en la UB, se han vuelto necesarias herramientas que faciliten el aprendizaje autónomo del estudiante y la corrección por parte de los profesores.

El Sistema Corrección de Programas se extiende de dos proyectos anteriores realizados por alumnos de la UB. Este fue pensado, sobretodo, para los primeros cursos de programación, en los cuales los alumnos han de realizar un alto número de entregas de pequeños programas. Dado que la mayoría del tiempo dedicado a realizar los programas prácticos es fuera del horario de laboratorio, la comunicación con el profesor se hace complicada. Esto conlleva a que el estudiante necesita una auto-corrección de sus programas para saber si sigue el camino correcto o no. Se encuentra más problemas sobre todo en el aspecto de la entrega de estos programas, dado que esta es prácticamente consecutiva y el profesor necesita un tiempo para la corrección del programa. Esto hace molesto el aprendizaje del estudiante debido a que puede necesitar la corrección del programa anterior para no acarrear un posible error.

A través de esta herramienta también se quiere facilitar la faena del profesor a la hora de corregir los programas subidos por los alumnos. El profesor deberá ejecutar en su ordenador cada uno de los programas entregados por los alumnos para corregir los resultados de éste. Mientras que con esta herramienta tan solo deberá acceder a la web y comprobar los resultados de la ejecución y los códigos de los programas subidos por los alumnos.

La motivación de este proyecto es principalmente el diseño e implementación de nuevas funcionalidades en el proyecto SCP, Sistema Corrección de Programas.

1.2 TFG Previos

Tal y como se ha comentado en el apartado anterior, este proyecto es la extensión de otros realizados anteriormente.

El primero de estos, el proyecto original, fue el realizado por Jordi Salvatella el primer semestre de 2011. Este proyecto se realizo con el objetivo de asegurarse que los alumnos entregaran programas con una correcta compilación y que pasaran unos juegos de pruebas mínimos. Para ellos se creó el SCP, el cual había de ser capaz de recibir el código del alumno, compilarlo y ejecutarlo contra dichos juegos de prueba, para así informar al alumno si su programa era correcto o contenía algún error.

Durante este proyecto, se desarrollo una aplicación web que se basaba en un Servlet java, que permitía subir un programa en un fichero comprimido, el sistema lo descomprimía, lo compilaba, lo ejecutaba y escribía los resultados en un fichero de *test*. El cual era leído por el sistema, mostrándolo en la página web.

El segundo proyecto, del cual se extiende este, es el realizado por Daniel Gil. En este caso se sigue con la idea del proyecto de Jordi Salvatella, pero se intenta conseguir un programa con una mayor perspectiva empresarial, es decir un código mucho más robusto y con mayor facilidad para su extensión. Para ello se utilizan herramientas y frameworks amplia-mente utilizados y documentados, como son el caso de Hibernate, Spring, MySQL y otros. Con esto se consigue solucionar problemas que contenía el primer proyecto, como la administración de usuarios, la modificación de la arquitectura monolítica y otros errores que contenía la primera versión del proyecto.

1.3 Objetivo General

El Objetivo general de este proyecto es disponer de una aplicación robusta y con un código claro, que se pueda desplegar en el Grado de Ingeniería Informática de la UB para facilitar el aprendizaje autónomo de los alumnos en las asignaturas de programación, así como facilitar el trabajo del profesor en la tarea de corrección de prácticas. Esto genera que sea una aplicación utilizada por un gran número de personas.

Para llegar a esos objetivos será necesario adquirir las competencias y conocimientos necesarios para trabajar con una aplicación con una estructura empresarial, además de adquirir los conocimientos básicos para trabajar con nuevos *frameworks* como son los utilizados en este proyecto. *Frameworks* que se analizaran a continuación en la sección *3*., como son *Hibernate, Spring, Java Server Faces, MySQL.*

Para ello se realizara una expansión del proyecto realizado por Daniel Gil, el cual se analizara en la sección *2. Análisis del SCP anterior* y del cual se sacaran los objetivos específicos.

2. Análisis del SCP anterior

Como se ha comentado en el apartado *1.2,* este proyecto proviene de otros proyectos realizados por alumnos de la UB. A continuación vamos a estudiar el estado del proyecto anterior realizado por Daniel Gil, se analizá de lo que se dispone.

Para analizar la versión anterior del proyecto, nos centraremos desde cuatro puntos de vista:

- *La base de datos*
- *Vista usuario Administrador*
- *Vista usuario Alumno*
- *Vista usuario Profesor*

2.1 Base de Datos

Primero nos centraremos en la base de datos, examinaremos el modelo ER, lo podemos ver en la *Ilustración 1,* y analizaremos sus entidades.

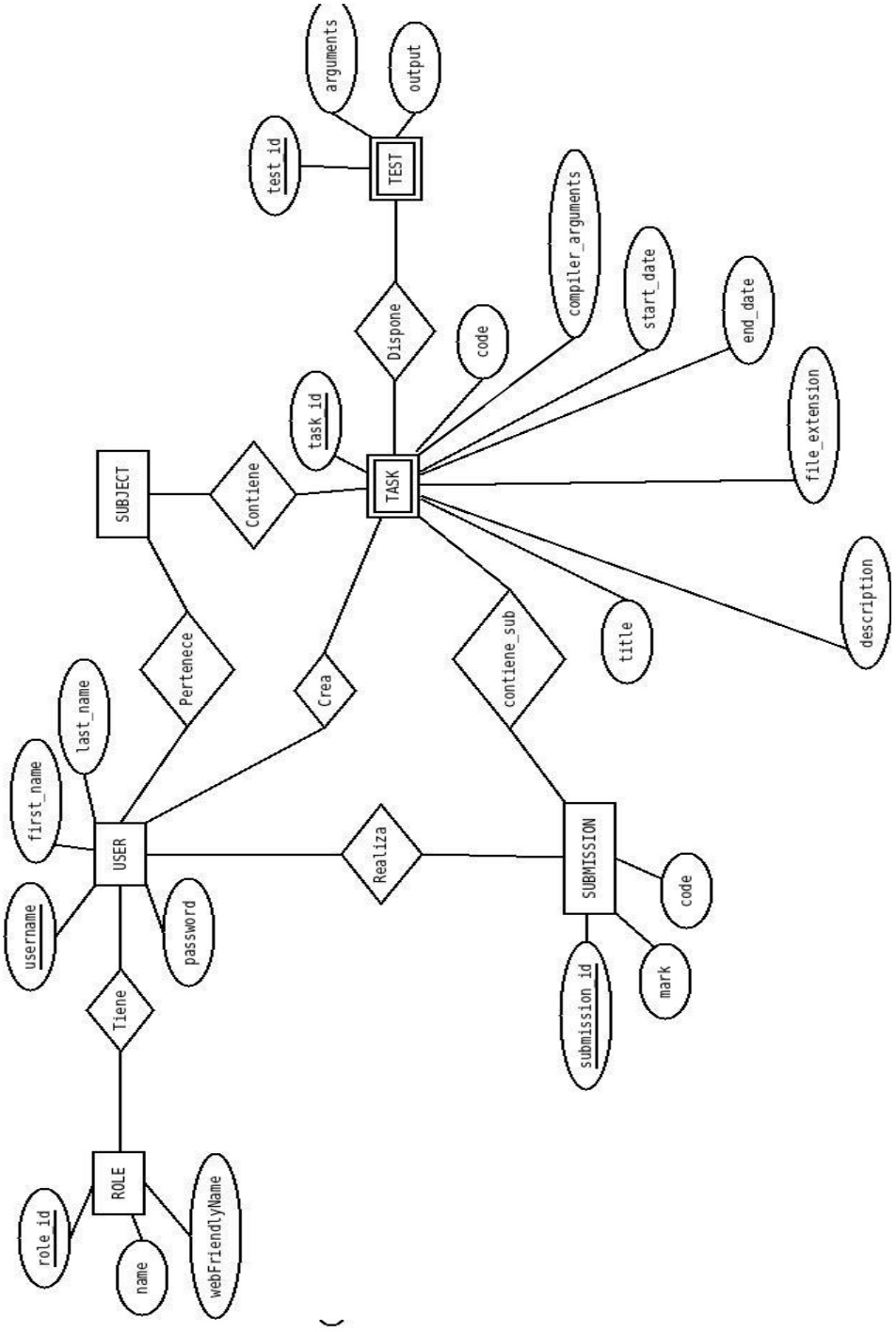


Ilustración 1: Diagrama ER del SCP Anterior

2.1.1 Análisis

ROLE: Entidad que contiene la información necesaria de los posibles roles de un usuario.

- *role_id*: Campo que identifica inequívocamente una entrada en la entidad Role. Está representado como un entero y no puede tener valor nulo.
- *name*: Nombre del Role. Está representado como un VarChar de tamaño máximo 255 y no puede ser nulo.
- *webFriendlyName*: Nombre del Role con el que se muestra en la aplicación Web. Igual que el *name* está representado como un VarChar de tamaño máximo 255 y no puede ser nulo.

USER: Es la persona que utilizara la aplicación, ya sea de la parte de administrador, profesor o alumno.

- *username*: Campo que identifica inequívocamente una entrada en la entidad User. Está representado como un Varchar de tamaño máximo 255.
- *password*: Campo obligatorio para el *loguin* del usuario en la aplicación. Está representado como un Varchar de tamaño máximo 255 y no puede ser nulo.
- *role:* Campo relacional que define el role del usuario.
- *first_name*: Nombre del usuario.
- *last_name*: Apellido del usuario.

SUBJECT: Representa cada una de las asignaturas que se han dado de alta en la aplicación.

- *subject_id*: Campo que identifica una entrada en la entidad Subject. Está representado con un entero y no puede ser nulo.
- *title*: Campo que nos indica el titulo de la asignatura. Está representado con un Varchar de tamaño máximo 255.

LANGUAGE: Entidad que contiene la información necesaria de los lenguajes posibles.

- *language_id*: Campo que identifica una entrada en la entidad Language. Está representado como un entero y no puede ser nulo.
- *name*: Nombre del Language. Está representado con un Varchar de máximo 255 caracteres.

- *file_extension*: Campo que identifica la extensión de los ficheros para ese lenguaje

TASK: Son las tareas que crea el profesor para cada asignatura.

- *task_id*: Campo que identifica una entrada en la entidad Task. Está representado como un entero y no pude ser nulo.
- *title*: Nombre de la tasca. Está representado como un Varchar de tamaño máximo 255, no pude ser nulo.
- *start_date*: Campo que representa la data de inicio de la Tarea, data a partir de la cual se pueden realizar las entregas. Está representado como un Date y no pude ser nulo.
- *end_date*: Campo que representa la data fin de la Tarea, data a partir de la cual ya no se pueden efectuar más entregas. Está representado como un Date y no puede ser nulo.
- *description*: Campo que almacena la descripción de la Tarea. Está representado como un Texto, no puede ser nulo.
- *owner*: Campo que relaciona la Tarea con el usuario propietario.
- *file_extension*:
- *code*: Código subido por el creador de la Tarea. Está representado como un Texto.
- *compiler_arguments*: Campo que representa los argumentos de compilación a la hora de compilar el código de esta tarea.
- *subject_id*: Campo que relaciona a que asignatura pertenece la tarea.

*SUBMISSION***:** Representa la entrega del alumno a una tarea del profesor.

- *submission_id*: Campo que identifica una entrada en la entidad Submission. Está representado como un entero y no puede ser nulo.
- *task_id*: Campo que relaciona una entrega con la *task*.
- *mark*: Representa el porcentaje de acierto de esa entrega. Está representado con un entero.
- *user*: Campo que relaciona una entrega con el usuario que la realiza.
- *code*: Código subido por el usuario que realiza la entrega. Está representado con un Texto.

TEST: Son los ejemplos que el profesor añade en una tarea para comprobar el código del alumno.

- *test_id*: Campo que identifica una entrada en la entidad Test. Está representado como un entero y no puede ser nulo.
- *task_id*: Campo que relaciona los *tests* creados con la tarea a la cual pertenecen.
- *arguments*: Argumentos de entrada para los *tests*.
- *output*: Salida al ejecutar el código con el *test*.

2.1.2 Problemas

Una vez visto y analizado el diagrama y las entidades de la base de datos, encontramos varios problemas funcionales.

1.El primer problema que se puede observar, es que los resultados del *test* del alumno no se guardan en ningún atributo o entidad de la base de datos. Tan solo se almacena el % de aciertos, gracias al atributo *mark,* así como el código entregado por el alumno. Esto provoca un problema de si el profesor quiere visualizar los resultados específicos obtenidos por el alumno.

2.El segundo problema encontrado está relacionado con la entidad Language. Esta entidad dispone de 3 atributos necesarios para informar un lenguaje de programación, pero como una de las ideas es ampliar la aplicación a más lenguajes de programación, esta entidad debería ser más extensa en cuanto a atributos. Por ejemplo necesitaremos atributos como, saber si el lenguaje es compilado o ejecutado, las instrucciones de ejecución para dicho lenguaje o si el fichero donde se encuentra el código a compilar necesita un nombre especifico, como sería el caso de Java.

2.2 Vista Administrador

Si analizamos la vista y procesos de los cuales es capaz el usuario Administrador observamos al entrar que la página se compone por dos tabuladores, uno de control de Usuarios y el otro de control de Clases, como se puede comprobar en la *Ilustración2*.

En el primero disponemos de una lista paginada de todos los usuarios de la aplicación, tanto los administradores, como profesores o alumnos, mostrando en una columna el nombre de usuario y en otra el Role. Con la selección de cualquiera de los alumnos utilizando el botón derecho del ratón, nos aparece un submenú con las opciones:

- view(Visualizar los atributos del usuario)
- Edit(Opción para editar al usuario).

A su vez se dispone de un botón “New User” que abre un diálogo para la creación de un nuevo Usuario.

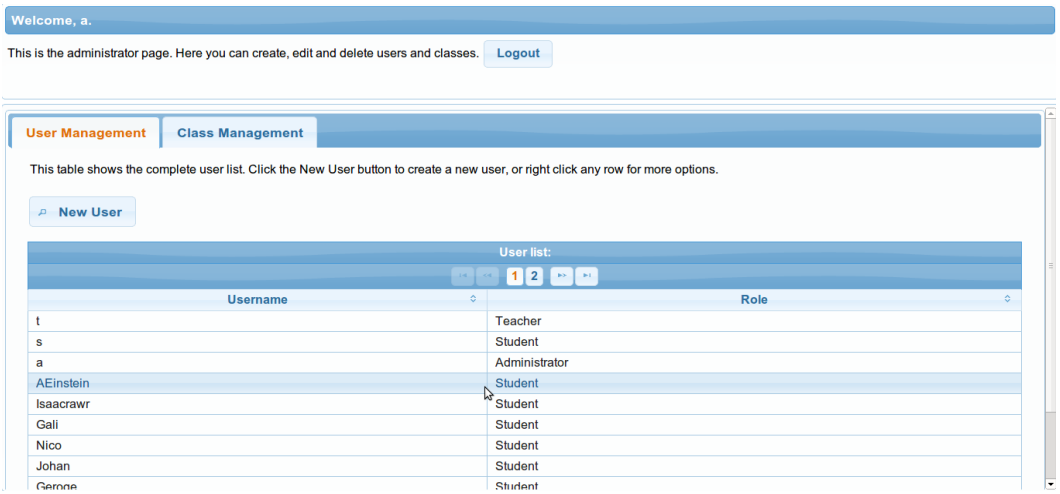


Ilustración 2: Vista Administrador SCP Anterior, Tabulador User

En el segundo tabulador se dispone de un formato parecido al primero, en el cual disponemos de una lista de las clases existentes. En este caso también disponemos del menú si seleccionamos con el botón derecho cualquier asignatura, con la opción de visionar la clase o editar. Además del botón con la funcionalidad para añadir una nueva Clase a la aplicación, la cual se registrara en la base de datos.

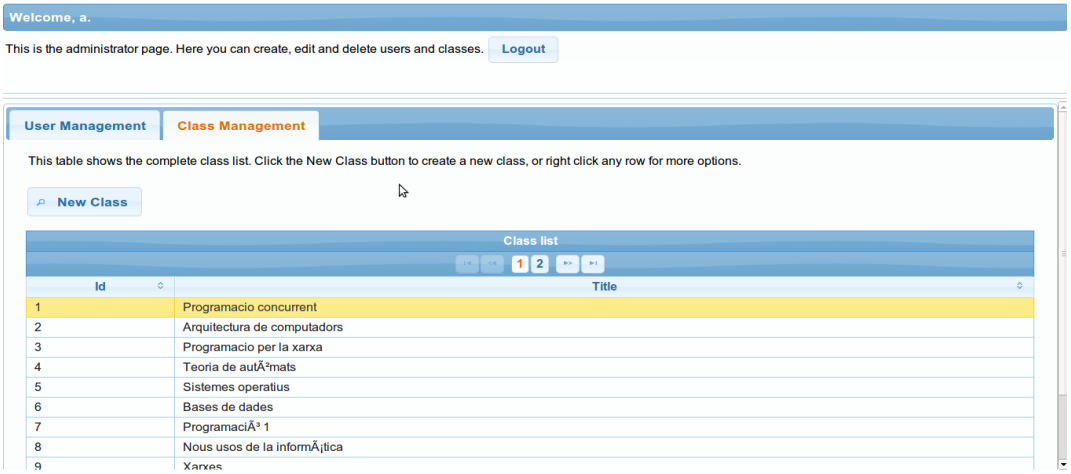


Ilustración 3: Vista Administrador SCP Anterior, Tabulador Class

En esta ocasión el diálogo que se abre cuando escogemos editar una clase es una ventana *drag&drop* en la cual aparecerá una lista de los usuarios que no estén en la clase y otra con

los usuarios que pertenecen a esta, se puede visualizar en la *Ilustración4*. Con la utilización de la opción del *drag&drop* se arrastra a los usuarios que deseemos añadir a la clase o los eliminar de esta.

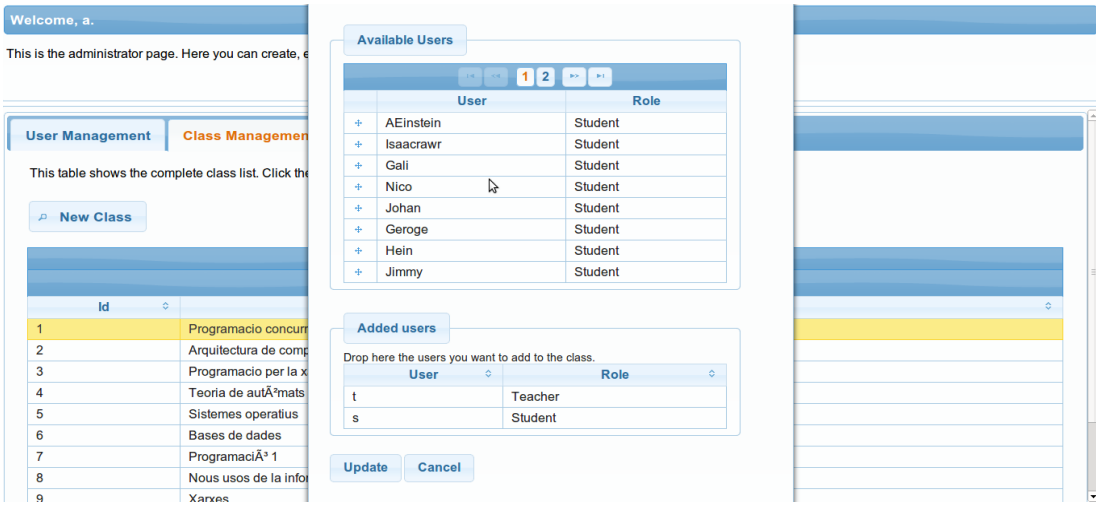


Ilustración 4: Vista Administrador SCP Anterior, Edit Class

2.2.1 Problemas

Una vez visto y analizado de lo que disponemos como usuario Administrador, cabe comentar que a grandes rasgos la aplicación no permite gestionar de forma eficiente una gran cantidad de usuarios, como son los casos que contemplamos. En particular:

1.A la hora de crear usuarios hay que crear manualmente uno a uno los usuarios y añadirlos manualmente a la asignatura en la cual debería estar asignado. Esta forma de gestionar puede ser un poco ineficiente, sobre todo teniendo en cuenta que en una clase puede haber más de 50 alumnos.

2.El segundo estaría más centrado en el diseño, dado que gestionar directamente a todos los usuarios puede llegar a ser molesto. Tal y como se ha comentado en el problema anterior, en una aplicación de este estilo se dispone de muchos usuarios y por lo tanto se necesita una forma eficiente de gestionarlos.

2.3 Vista Alumno

A continuación analizamos la vista y las funciones que puede realizar el alumno en esta versión de la aplicación. Esta vista es de las más sencillas, una vez entramos como alumno disponemos de dos listas distintas, la de las tareas activas y las que ya han expirado. Si seleccionamos una de las tareas activas con botón derecho disponemos de la opción *upload code*, opción para realizar la entrega de la tarea.

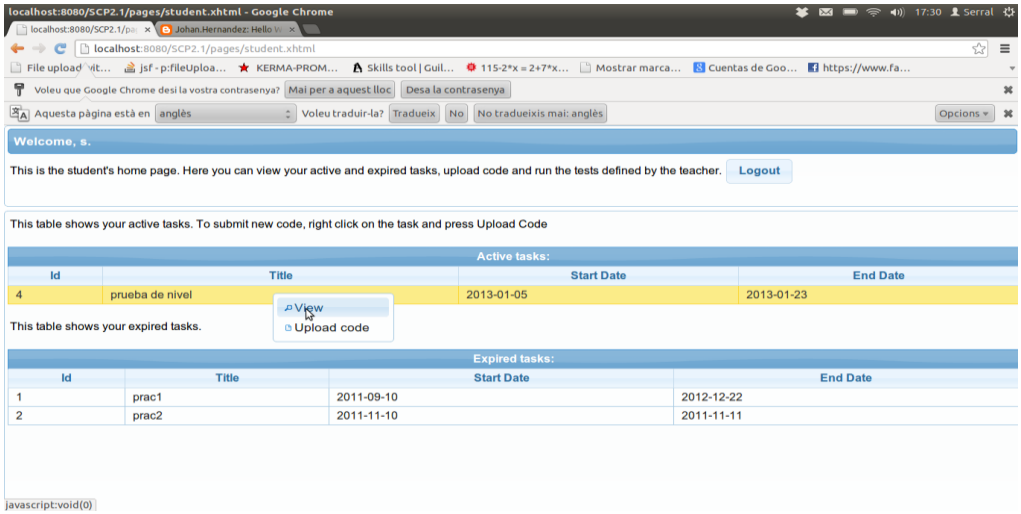


Ilustración 5: Vista Alumno SCP Anterior

Con la opción de menú *upload code* accedemos a una nueva vista donde se realizará la entrega del código. Esta se compone de un cuadro de texto para introducir nuestro código, una tabla donde se mostraran los resultados de los *test* realizados y dos botones:

- Run Test, para realizar la validación del código con los *test* subidos por el profesor previamente
- Save para realizar el guardado de esta entrega.

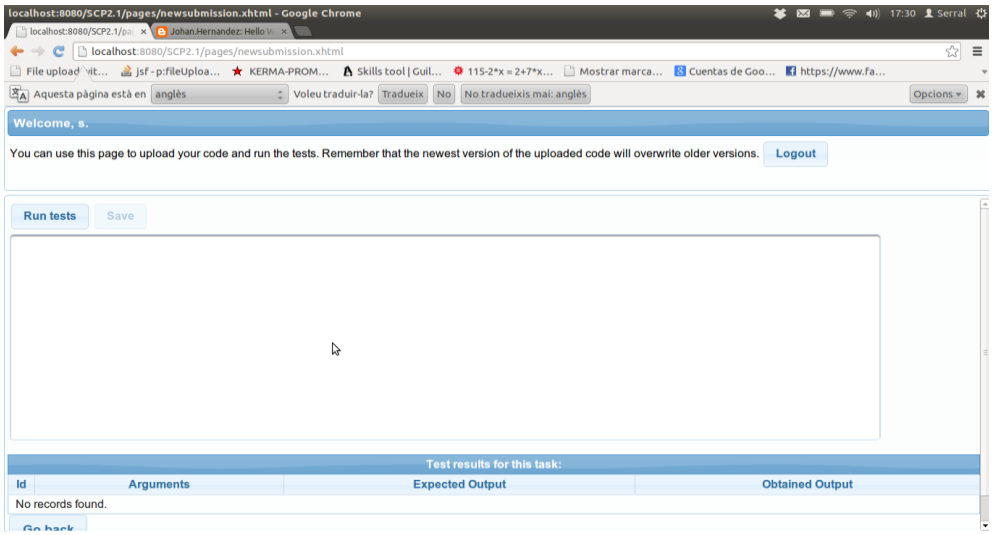


Ilustración 6: Vista Realizar Entrega SCP Anterior

2.3.1 Problemas

En este caso después de analizar la vista y funciones como usuario Alumno, tan solo se ha encontrado, a grandes rasgos, un problema. Este está relacionado con la entrega de una tarea; actualmente no existe la opción de subir o actualizar el código a través de la subida de

un fichero. Está funcionalidad es útil si interpretamos que el alumno previamente realizara las pruebas en su equipo y casi seguramente utiliza un fichero para escribir el código.

2.4 Vista Profesor

La ultima vista para analizar de la aplicación, es la vista del profesor. Esta vista está orientada para que el profesor sea capaz de controlar las tareas de sus asignaturas.

En la vista del profesor encontramos un diseño similar al del Alumno. En esta encontramos dos listas, una con las tareas activas y otra con las tareas expiradas. Si se selecciona con el botón derecho una tarea activa, se nos mostrará un menú con las opciones:

- *View*(visualizar los atributos de la tarea),
- *Edit Test*(Para añadir *test* a la tarea)
- *View Report*(Para ver el *report* de la entrega de los alumnos para esa tarea)
- *Delete*(Eliminar la tarea).

Además se dispone de un botón *New Task*, que nos redirigirá a una nueva vista, la cual se ocupara de la creación de la nueva tarea.

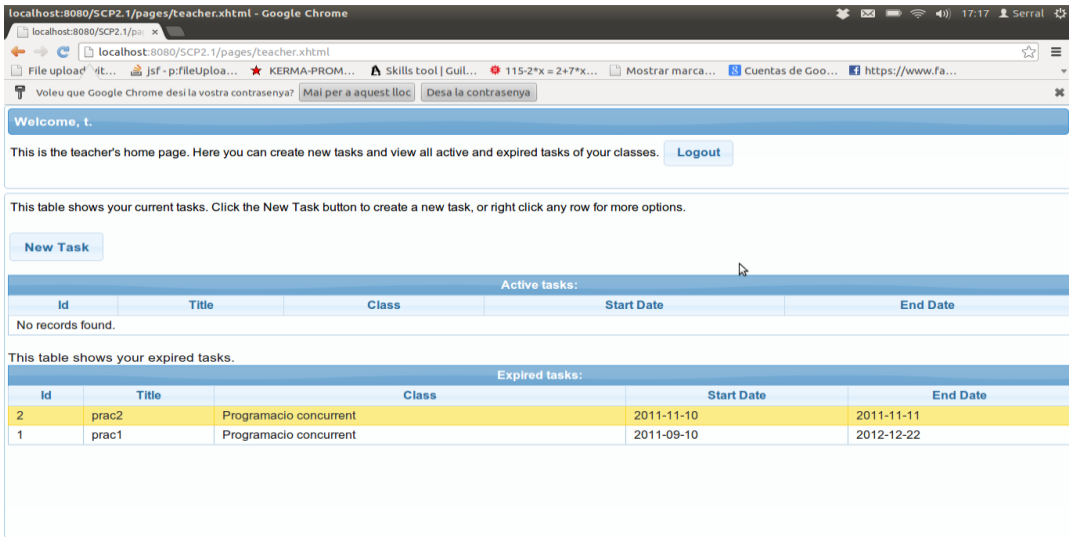


Ilustración 7: Vista Profesor SCP Anterior

La vista de la nueva tarea, se compone por tres pestañas donde configurar los datos de la nueva tarea. En la primera se nos pedirá introducir: el titulo de la tarea, la asignatura a la que pertenece esta tarea, la data de inicio y fin y una breve descripción.

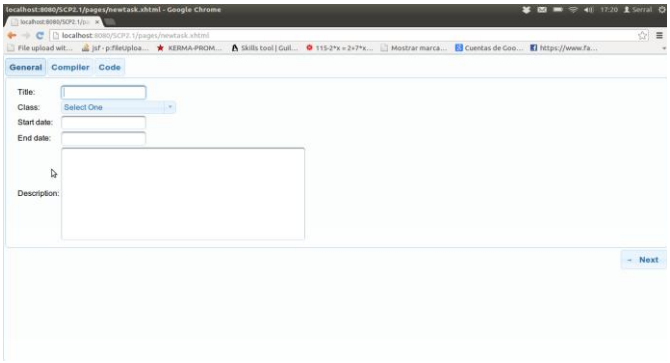


Ilustración 8: Vista Nueva Tarea SCP Anterior, Tabulador General

En la segunda se pedirá seleccionar un lenguaje y escribir los argumentos de compilación si son necesarios.

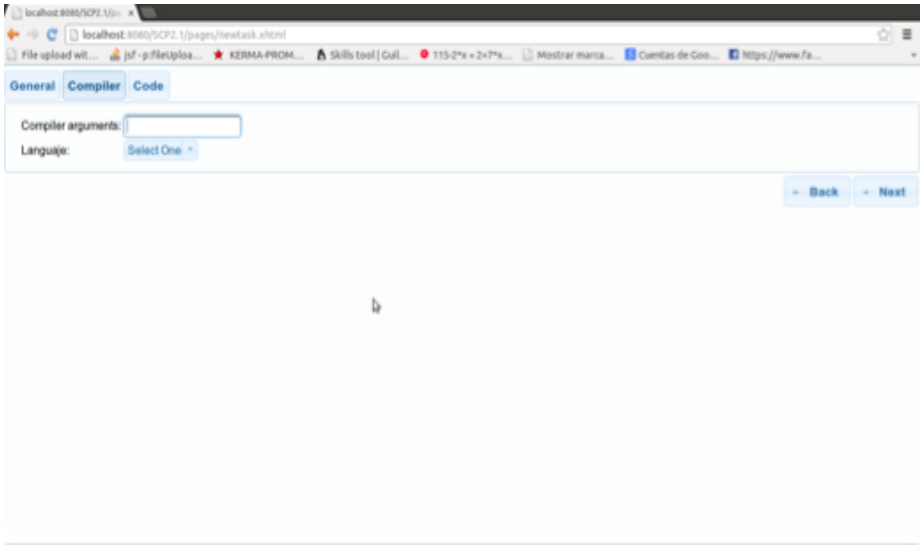


Ilustración 9: Vista Nueva Tarea SCP Anterior, Tabulador Compiler

Por último en la última, se dispone de un cuadro de texto donde escribir el código, los botones

- *Check Code* (para compilar el código y comprobar si contiene errores)
- *Submit* para crear la tarea.

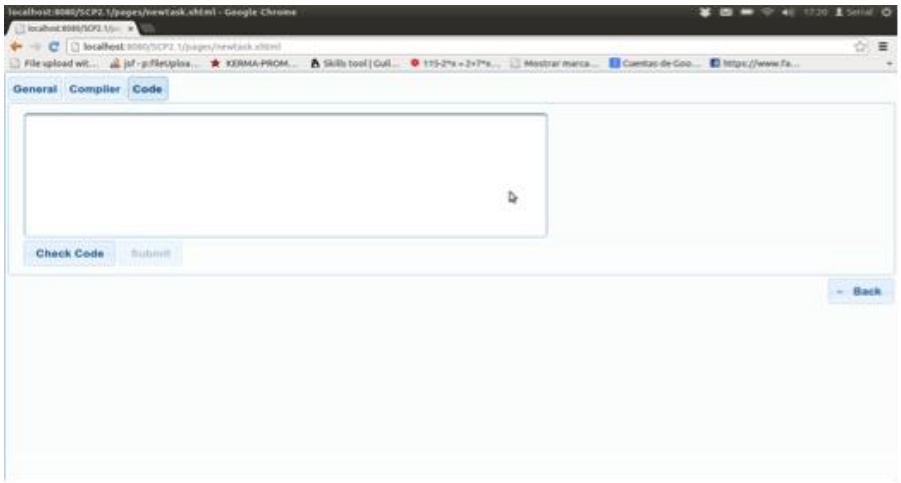


Ilustración 10: Vista Nueva Tarea SCO Anterior, Tabulador Code

Una vez se crea y guarda la nueva tarea, se redirige a la vista de añadir *tests* a la tarea.

Esta vista se accede tanto desde cuando se crea una nueva tarea o desde la opción del menú *add Test*. Esta vista está compuesta por un botón *AddTest*, que nos abrirá un dialogo en el cual podremos escribir los argumentos de entrada para la ejecución y dos salidas de texto que nos mostraran los resultados o los errores, así como los botones de *RunTest*, encargado de ejecutar el código subido con el *test* introducido, *Save y Cancel*. Debajo de este botón AddTest encontramos una lista con los diferentes *tests* añadidos por el profesor.

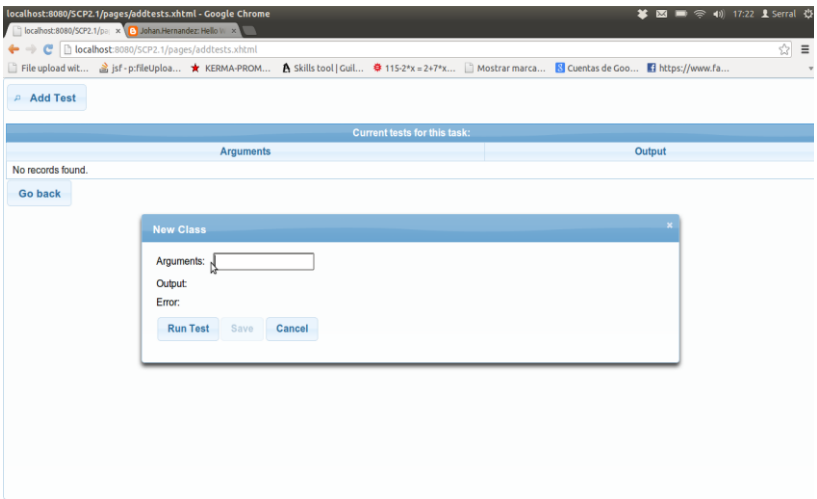


Ilustración 11: Vista Añadir Test SCP Anterior

2.4.1 Problemas

Una vez analizado las funciones y vista del usuario Profesor, cabe comentar varios errores de funcionalidad encontrados.

1.El primero que podemos citar es el que ya se ha comentado en la vista del Alumno, es el hecho de no poder subir el código con la carga de un fichero. Dado que si un profesor realiza el código en su equipo no tiene porque volver a escribirlo sino con tan solo cargar el fichero ya estaría.

2.Relacionado con el tema de introducir un código para una tarea, esta también relacionado el segundo problema. Actualmente el profesor se ve siempre obligado a subir su código para que así el sistema pueda realizar una comparación entre los resultados del código del profesor con los resultados del código de los alumnos. Esto provoca que aunque el profesor únicamente quiera comprobar los resultados de los alumnos, sin necesidad de comparativa, se vea obligado a subir el código. La comparativa de resultado puede provocar confusión al alumno, dado que por un simple espacio de más entre caracteres la comparativa podía dar la un resultado como erróneo, cuando en realidad para el profesor serian unos resultados correctos.

3.Por último, se encuentra que en la vista de añadir *test* se está obligando al profesor a utilizar un único estilo de argumentos, sin dar la opción de pasar a la ejecución del código ficheros como argumentos u otras variantes.

Además se han encontrado otros pequeños problemas como la posibilidad de editar *test* una vez inicializada una tarea o el no poder acceder a ver el *report* de tareas ya expiradas.

2.2 Objetivos de la nueva versión de SCP

Una vez analizado el estado del proyecto con sus actuales errores marcamos como objetivo principal la creación de una nueva versión del proyecto SCP, utilizando las tecnologías y la funcionalidad de la antigua versión se consiga una versión más funcional y más robusta del SCP. Para ello nos centraremos en los siguientes objetivos:

- **1.** Para solucionar el problema en la vista del Administrador, se ha planteado un nuevo diseño de la vista que nos facilite la gestión de usuarios. Además para solucionar el problema de dar de alta a nuevos usuarios y asignarlos a sus respectivas asignaturas se ha optado por implementar una solución basada en la subida de un fichero que contenga los datos de los usuarios de esa asignatura. Este se podrá extraer de la correspondiente asignatura a través del Moodle(Campus Virtual).
- **2.** Facilitar las funcionalidades de realizar la entrega en el caso del alumno o crear una nueva tarea en el caso del Profesor. Para ello se habilitará entre otras cosas, la función de subir el código en un fichero.
- **3.** Modificar la base de datos y la aplicación para que el profesor pueda visualizar los resultados del alumno en el momento en que revise el *report* de la tarea. Así como modificar en la base de datos la entidad Language, para que disponga de más atributos interesantes para la ejecución de código en otros lenguajes.
- **4.** Habilitar la opción para el Profesor de cuando se crea una nueva tarea, pueda decidir si introducir código o no. Se modificará en la aplicación todos los sitios donde sea necesario por la intervención de este campo. Esto puede afectar tanto la vista del Profesor como la del Alumno, dado que en los *tests* no existirá los resultados del profesor con los que realizar la comparativa.
- **5.** Por ultimo se corregirán pequeños problemas de funcionalidad que puedan surgir, como es el ejemplo poder editar los *test* una vez inicializada la tarea o otros que puedan surgir.

3. Tecnologías utilizadas

En el actual proyecto se ha seguido con las tecnologías utilizadas en su versión anterior. Para ello seguiremos utilizando el modelo de arquitectura de 3 capas modelo-vista-controlador (MVC) y la programación en Java EE con un contenedor de *servlets Tomcat*.

Java nos aporta una gran seguridad, la cual es el principal requisito para nuestra aplicación. Esto por encima de inconvenientes como que el desarrollo de aplicaciones web con Java es más lento que con otros lenguajes más especializados para ello.

Además de Java, se seleccionaron los *frameworks* estándar del diseño de aplicaciones web empresariales. Estos se describen a continuación.

3.1 Hibernate

Cuando trabajamos en Java con aplicaciones web, uno de los principales problemas que encontramos es la transformación necesaria de los objetos Java a las tablas SQL de la base de datos. Usualmente en el diseño de estas aplicaciones, las clases encargadas de esta función se les conocen como *DAO* (Data Access Object).

Hibernate, es uno de los ORM (Object Relational Mapping) más conocidos y utilizados. Este tipo de *framework* se encarga de la conversión de los objetos Java y las tablas SQL, además con su sintaxis simple se nos permite la realización de consultas complejas de forma más sencilla. Gracias a esto se nos permite la creación de clases DAO mucho más simples y de mas fácil mantenimiento. Esto es posible dado que con las clases DAO que utilicen Hibernate , no nos veremos obligados a realizar una revisión del código que accede a la base de datos por cada modificación efectuada en esta o en las clases Java. Como podemos ver en el ejemplo de DAO con Hibernate.

```
Session session = sessionFactory.getCurrentSession();


            try{
                    Criteria criteria = session.createCriteria(Task.class);
                    criteria.add(Restrictions.eq("user", user));
                    criteria.add(Restrictions.ge("endDate", new Date()));
                    criteria.addOrder(Order.asc("endDate"));
                    return criteria.list();
            }catch(Exception e){
```

```
                e.printStackTrace();
            }
        return null;
```

Código: Ejemplo de una función de busqueda con Hibernate

Con Hibernate simplemente definimos la clase sobre la que realizar la consulta (Task.class), y utilizando el objeto Criteria, facilitado por este, añadimos los criterios para la consulta, i nos retornara una lista con los objetos de dicha consulta.

A continuación se muestra un ejemplo del *mapping* que relaciona un objeto Java con su correspondiente tabla SQL. Para ello necesitamos tener la tabla en la base de datos, un objeto Java y el fichero de configuración de Hibernate que se ocupa del *mapping* entre la clase y la tabla.

Código SQL

```
CREATE TABLE user (
        username VARCHAR(255),
        password VARCHAR(255) NOT NULL,
        role INT NOT NULL,
        first_name VARCHAR(255) NOT NULL,
        last_name VARCHAR(255) NOT NULL,
        PRIMARY KEY (username),
        FOREIGN KEY (role) REFERENCES
role(role_id)
);
```

Código: Ejemplo de código de tabla SQL

Código Java

```
public class User implements
java.io.Serializable{

        private String username;
        private Role role;
        private String firstName;
        private String lastName;
        private String password;
        private Set<Task> tasks = new
HashSet<Task>(0);
        private Set<Submission> submissions =
new HashSet<Submission>(0);
        private Set<Subject> subjects = new
HashSet<Subject>(0);

        public User() {
        }
```

Código: Ejemplo de código de entidad Java

Código Mapping

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- Generated Nov 4, 2011 3:38:03 PM by Hibernate Tools 3.4.0.CR1 -->
<hibernate-mapping>
    <class name="ub.scp2.entities.User" table="user" catalog="scp2">
        <id name="username" type="java.lang.String">
            <column name="username" />
        </id>
    </class>
</hibernate-mapping>
```

Código: Ejemplo de código del mapeo de Hibernate

Esta configuración nos permite que gracias al *mapping* de la clase, podamos cambiar de base de datos sin tener que cambiar ninguna línea de código Java, únicamente modificar la configuración de Hibernate. Además nos facilita el acceso a registros de la base de datos desde las clases Java, ya sea por ejemplo para realizar inserciones o borrado de estos. Como ejemplo en lugar de tener que crear una consulta para realizar un guardado de un objeto Java, éste se puede guardar únicamente utilizando:

```
session.save(task);
```

Lo mismo pasaría con el ejemplo del borrar un objeto de una tabla de la base de datos.

```
session.delete(task);
```

3.2 Spring

Cuando trabajamos con aplicaciones web de gran tamaño, como puede ser este proyecto. Nos encontramos con la dificultad de mantener un código limpio y de fácil mantenimiento, dado que en ocasiones encontramos que la lógica de negocio

se realiza en la propia clase del *servlet*. Esto para casos simples de programación puede ser una forma muy válida, pero para casos como el nuestro, en el que la lógica de negocio puede ser muy amplia, nos encontramos en que rápidamente se puede llegar a generar un código muy complejo y de difícil mantenimiento.

Spring es un *framework* de inyección de dependencias con un amplio uso en el mundo empresarial por los desarrolladores de Java, aunque también hay versiones para .NET. La inyección de dependencias se refiere a que podremos declarar qué objetos necesitamos sin necesidad de instanciarlos, de eso se ocupara Spring que será el encargado de realizar la instanciación y la destrucción. Esto nos beneficia a la hora de disponer de un código mucho más eficiente y limpio.

Para que Spring sepa que instanciaciones debe realizar se realiza la configuración en el fichero applicationContext.xml a través de una configuración como la del ejemplo.

```
<bean id="role" class="ub.scp2.boImpl.RoleBoImpl">
        <property name="roledao" ref="roleDAO"/>
</bean>
```

Código: Ejemplo de configuración Spring

```
public class TaskBoImpl implements TaskBo{

    private TaskDao taskdao;
   ...
```

Código: Ejemplo de clase sin instancia

Spring Security 3

Es un *framework* derivado de Spring, encargado de controlar la identificación, la autorización de las diferentes paginas, sesiones, roles etc.

3.3 MySQL

Como la intención es seguir trabajando con la misma base de datos se seguirá utilizando MySQL, ya que es una base de datos con una documentación excelente, con licencia libre y con amplitud de funcionalidades.

MySQL es un gestor de base de datos relacional, *multi-threaded*, multi usuario, desarrollado por la empresa MySQL AB y programado en lenguaje ANSI C. Los usuarios pueden elegir entre usar el software MySQL como un producto Open Source bajo los términos de la licencia GNU GPL o pueden adquirir una licencia comercial estándar de MySQL AB. Está soportado en la gran mayoría de distribuciones Linux así como para un gran número de versiones de Windows. Se utiliza en su gran mayoría para el desarrollo de aplicaciones web, al ser nuestro proyecto una aplicación web nos servirá de gran ayuda utilizar MySQL, ya que se dispone de mucha y buena documentación.

3.4 Java Server Faces (JSF)

Para el desarrollo de la capa de la vista se utilizó la tecnología Java Server Faces, la cual se seguirá utilizando para la modificación de vistas así como para la posible creación de nuevas vistas.

Java Server Faces es una tecnología para la implementación de interfaces de usuario en aplicaciones web desarrolladas en lenguaje Java. JSF utiliza la tecnología Java Server Pages(JSP) como tecnología que permite el despliegue de las páginas de la interface de usuario, con posibilidad de adaptarse a otras como Html.

La tecnología JSF está basada en librerías para el desarrollo de las paginas JSP o HTML y por otra parte de Beans Java utilizados como el sistema de inyección de dependencias que se ocupan de controlar las páginas que visualiza el usuario y también del control de las "*response*" y "*requests*" de estas páginas, conocidos como ManagedBean.

- ***Primefaces***

Es un conjunto de librerías de componentes JSF2 que incluye un gran número de componentes ricos para una buena funcionalidad, tales como botones, tablas, menús etc. Que nos permiten diseñar una vista de usuario de forma sencilla.

4. Especificaciones de requerimientos

A continuación se expondrá las especificaciones de los requerimientos necesarios para la aplicación. Por ello encontrara las funcionalidades de los diferentes tipos de usuario de la aplicación, así como los casos de uso, que marcan las diferentes acciones que el usuario podrá ejercer.

4.1 Actores y Funcionalidades.

Se dispone de tres actores principales: administrador, alumno y profesor. Para ofrecer funcionalidades distintas para cada uno de los actores se presentara un punto de entrada común.

4.1.1 Funcionalidades Administrador.

El usuario Administrador será el responsable del control de la aplicación. Así pues sus funciones serán la administración de las asignaturas y usuarios. Por eso la aplicación debe permitir las siguientes funcionalidades:

- Creación, edición y borrado de usuarios.
- Creación, edición y borrado de asignaturas.
- Limpiar la base de datos de la aplicación.

4.1.2 Funcionalidades Alumno.

El usuario Alumno tendrá que poder ver las tareas de las distintas asignaturas en las cuales está registrado y poder realizar la entrega hasta que la tarea finalice. Para ello la aplicación debe disponer de las siguientes funcionalidades:

- Listado de las tareas, con su correspondiente información.
- Vista para realizar la entrega para la tasca. Esta ha de permitir la subida del fichero, así como mostrarnos si dicho código compila y supera los *test* declarados por el profesor.
- Modificación de su Password

4.1.3 Funcionalidades Profesor.

El usuario Profesor será el responsable de la administración de las tareas para las clases en las que este asignado. Para ello la aplicación debe disponer de las funcionalidades siguientes:

- Creación y borrado de tareas, introduciendo todos los datos necesarios para estas.
- Visualización de las entregas de los alumnos para las tareas.
- Añadir *tests* en las tareas aun no inicializadas.
- Modificación de su *Password.*

4.2 Casos de Uso.

Los casos de uso son las acciones que el usuario podrá realizar en nuestra aplicación. En los siguientes diagramas se mostraran las distintas acciones que el usuario podrá realizar en la aplicación.

El que sería el diagrama de casos de uso de toda la aplicación se ha descompuesto en tres diagramas distintos, para que así sea mucho más fácil de entender. Cada uno de estos representa a los distintos Actores explicados anteriormente en el apartado 4*.1.*

4.2.1 Casos de Uso Administrador.

Este diagrama, *Ilustración 12,* nos muestra las acciones de las cuales dispone el actor Administrador. A continuación se procede a la explicación detallada de cada uno de los casos de uso.

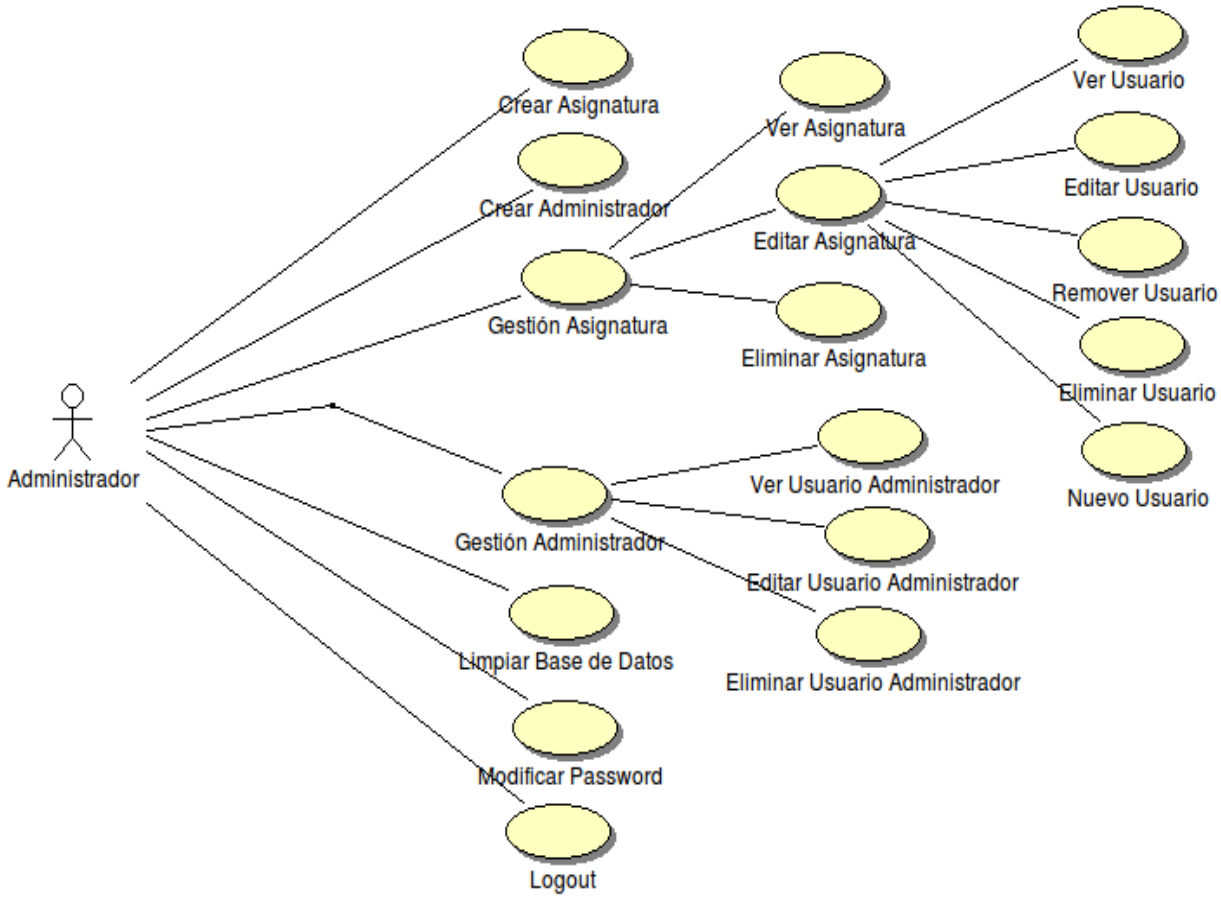
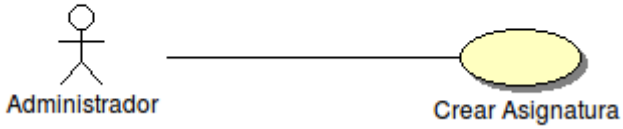


Ilustración 12: Diagrama de Casos de Uso Administrador

4.2.1.1 Crear Asignatura



Caso de Uso: Creación de una Asignatura

Actor: Administrador

Descripción: Solicita al Administrador los datos necesarios para crear una asignatura. Además de la selección de un fichero con los usuarios, alumnos y profesores, que se quiera registrar en esta asignatura. La asignatura es registrada en la Base de Datos y será accesible para las funciones que lo requieran.

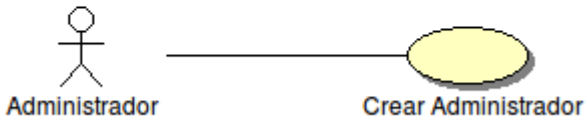
Requerimientos: Completar todos los campos obligatorios.

Excepciones:

1.No completa todos los datos necesarios. La aplicación retorna un aviso de falta de datos.

2.El usuario no sube ningún fichero. La aplicación creara la asignatura y la registrara en la Base de Datos pero sin asignarle ningún usuario.

4.2.1.2 Crear Administrador



Caso de Uso: Creación de un Nuevo Administrador.

Actor: Administrador

Descripción: Solicita al Administrador los datos para la creación de un nuevo usuario administrador. El nuevo administrador será añadido a la Base de Datos y será accesible para todas las funciones que lo requieran.

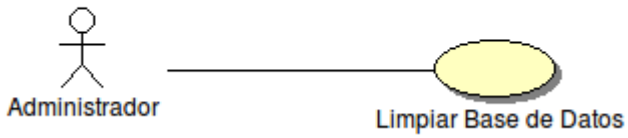
Requerimientos: Completar todos los datos necesarios.

Excepciones:

1.No completa todos los datos necesarios. La aplicación retorna un aviso informando que campo nos falta por completar.

2.La contraseña que se le asigna al usuario es de un tamaño menor o no coincide con la confirmación de contraseña. La aplicación retorna un aviso de error en la contraseña.

4.2.1.3 Limpiar Base de Datos



Caso de Uso: Limpiar "*resetear*" la base de datos.

Actor: Administrador

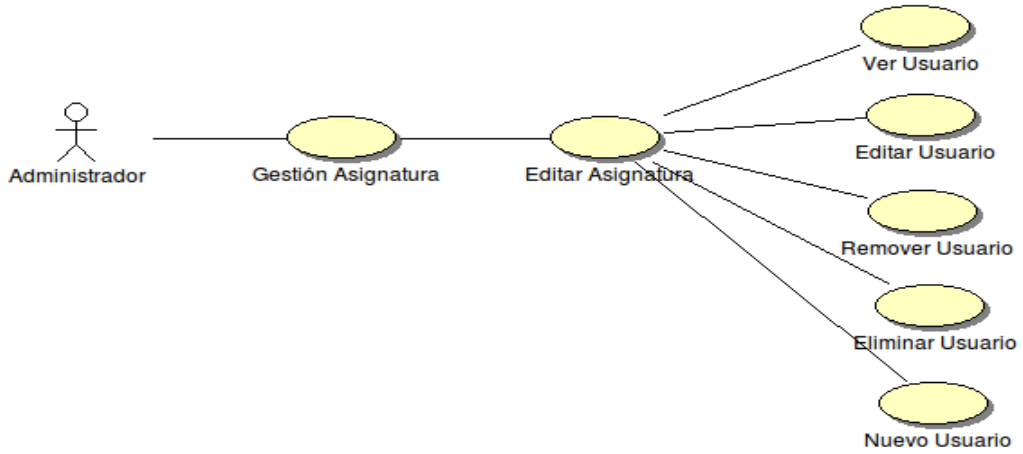
Descripción: El Administrador decide dejar la Base de Datos con la configuración inicial. Se vacía el contenido de las tablas de la Base de Datos, dejando únicamente los registros de la configuración inicial.

Requisitos: Aceptar el mensaje de confirmación.

Excepciones:

1.El Administrador no acepta el mensaje de precaución. La aplicación no realiza la limpieza de la base de datos hasta que no se acepte.

4.2.1.4 Editar Asignatura



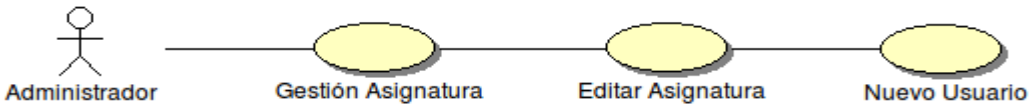
Caso de Uso: Editar una Asignatura.

Actor: Administrador

Descripción: Se procede a editar una asignatura, una vez seleccionada esta acción disponemos de una lista de usuarios, además de todas las acciones de gestión de usuarios, visualizar el alumno, editar al alumno, remover al alumno de la clase, eliminar o agregar un alumno de la Base de Datos. Una vez realizado los cambios de la clase esta se actualizara en la Base de Datos, así como los cambios realizados en sus usuarios.

Requerimientos: Se necesitara que exista alguna asignatura en la Base de Datos y que esta contenga usuarios asignados. Si no existieran usuarios en la asignatura la única acción posible sería la de la creación de nuevo usuario.

4.2.1.5 Nuevo Usuario



Caso de Uso: Creación de nuevo usuario en asignatura.

Actor: Administrador

Descripción: Se decide crear un nuevo usuario para esa asignatura, ya sea alumno o profesor. Solicita al Administrador los datos para un nuevo usuario para esa asignatura, este es guardado en la base de datos.

Requerimientos: Se debe haber realizado la acción del caso de uso 4*.2.1.4.*

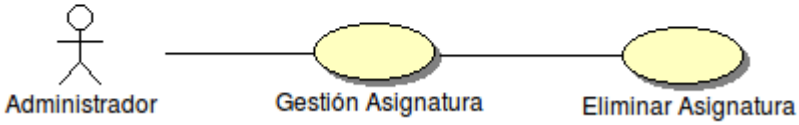
Excepciones:

1.El usuario ya existe en la Base de Datos. La aplicación no guardara el usuario en la Base de datos sino que directamente se asignara ese usuario a la asignatura.

2.No completa todos los datos necesarios. La aplicación retorna un aviso informando que campo nos falta por completar.

3.La contraseña que se le asigna al usuario es de un tamaño menor o no coincide con la confirmación de contraseña. La aplicación retorna un aviso de error en la contraseña.

4.2.1.6 Eliminar Asignatura



Caso de Uso: Eliminar una Asignatura.

Actor: Administrador

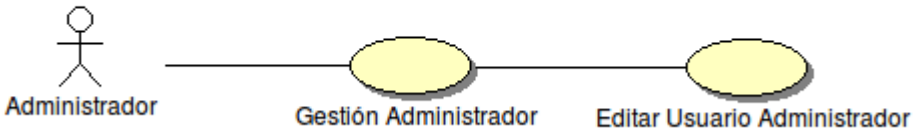
Descripción: Solicita al administrador si está de acuerdo en eliminar la asignatura de la Base de Datos. La aplicación eliminara de la Base de Datos la asignatura así como las tareas y entregas que pertenezcan a dicha asignatura.

Requerimientos: Haber seleccionado una asignatura y aceptar el mensaje de confirmación de eliminar la asignatura.

Excepciones:

1.El Administrador no acepta el mensaje de confirmación. La aplicación no eliminará la asignatura de la base de datos hasta que no se acepte.

4.2.1.7 Editar Usuario Administrador



Caso de Uso: Editar a un usuario Administrador.

Actor: Administrador

Descripción: Solicita los datos para editar al usuario administrador. La aplicación realiza la actualización en la Base de Datos.

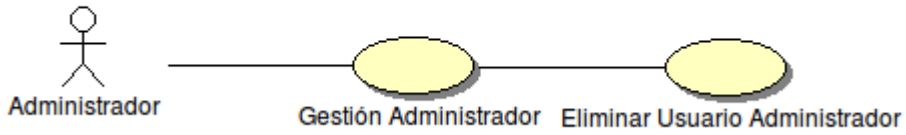
Requerimientos: Se ha debido seleccionar al usuario administrador que se desea editar, además de completar todos los datos solicitados para la edición.

Excepciones:

1.No completa todos los datos necesarios. La aplicación retorna un aviso informando que campo nos falta por completar.

2.La contraseña que se le asigna al usuario administrador es de un tamaño menor, o no coincide con la confirmación de contraseña. La aplicación retorna un aviso de error en la contraseña.

4.2.1.8 Eliminar Usuario Administrador



Caso de Uso: Eliminar a un usuario Administrador.

Actor: Administrador

Descripción: Solicita al administrador si está de acuerdo en eliminar el usuario de la Base de Datos. La aplicación eliminara de la Base de Datos al usuario con rol administrador, al ser borrado de la Base de Datos el usuario no podrá volver a conectarse a la aplicación a no ser que se vuelva a dar de alta por otro Administrador.

Requerimientos: Se ha debido seleccionar a un usuario Administrador, y aceptar el mensaje de confirmación que mostrará la aplicación.

Excepciones:

1.No se acepta el mensaje de confirmación. La aplicación no eliminará al usuario Administrador, seleccionado, de la base de datos hasta que no se acepte.

4.2.2 Casos de Uso de Alumno

Este diagrama, *Ilustración 13*, nos muestra las diferentes acciones de las cuales dispone el actor Alumno. A continuación se procede a la explicación detallada de cada uno de los casos de uso.

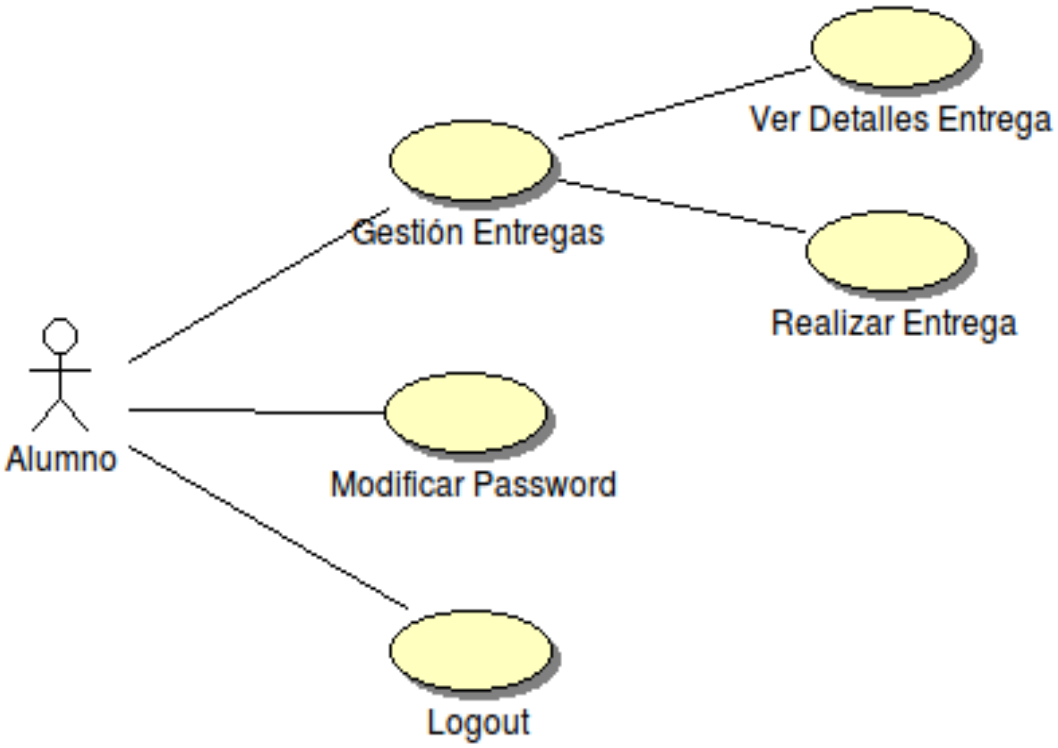
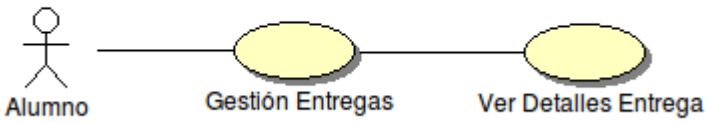


Ilustración 13: Diagrama de Casos de Uso Alumno

4.2.2.1 Ver Detalles Entrega



Caso de Uso: Ver detalles de la entrega.

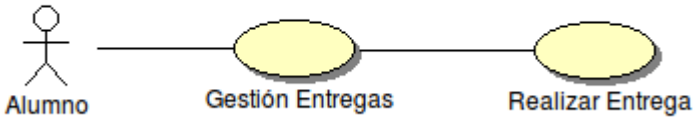
Actor: Alumno

Descripción: Se mostrará varios datos sobre la entrega.

Requerimientos: Se ha de seleccionar una entrega de la lista.

Excepciones: -

4.2.2.2 Realizar Entrega



Caso de Uso: Realizar una Entrega.

Actor: Alumno

Descripción: Se solicita al alumno el código que desea comprobar y entregar, éste podrá ser escrito directamente o subido desde un fichero para facilitar la faena al alumno. La aplicación compilara y ejecutara el código subido por el alumno mostrando los resultados así como si ha obtenido algún error de compilación.

Requerimientos: Se ha de haber seleccionado una tarea. Además el código subido por el alumno no ha de dar ningún problema de compilación.

Excepciones:

1.El código introducido por el alumno contiene errores de compilación. La aplicación al compilar el código y ver errores de compilación en éste, dará un aviso y no habilitará la opción de entrega de la tarea.

4.2.3 Casos de Uso del Profesor

Este diagrama, *Ilustración 14,* nos muestra las acciones de las cuales dispone el actor Profesor. A continuación se procede a la explicación detallada de cada uno de los casos de uso.

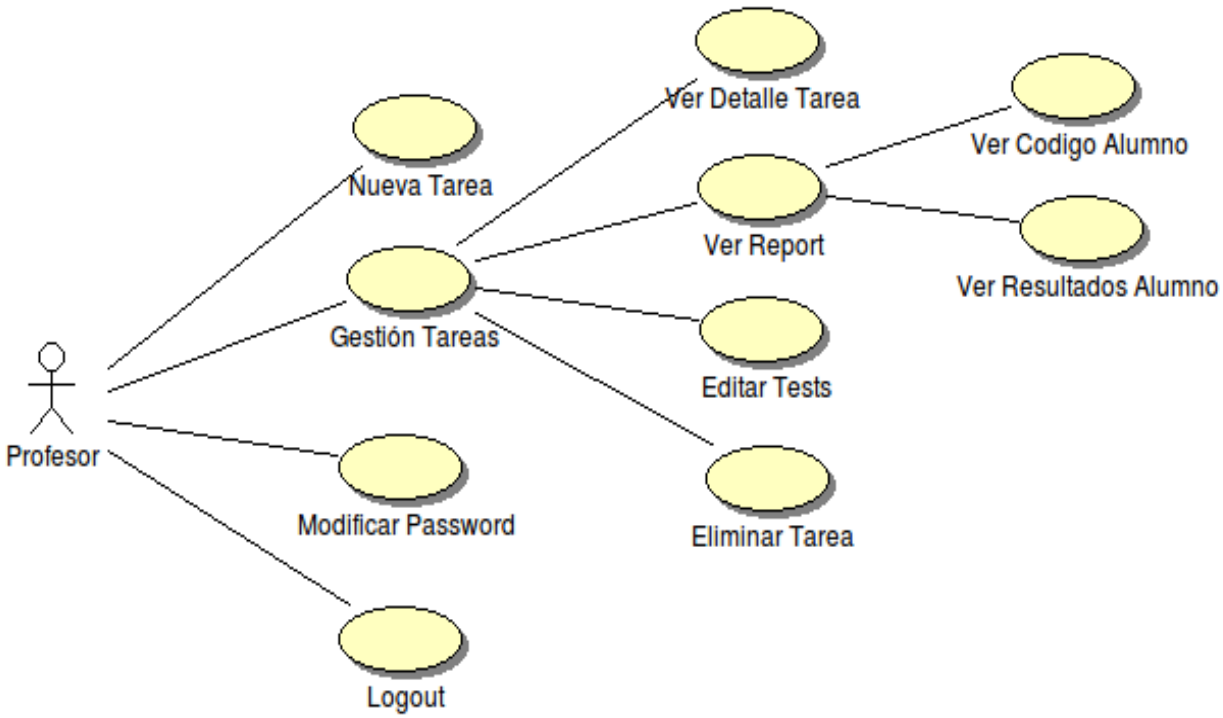
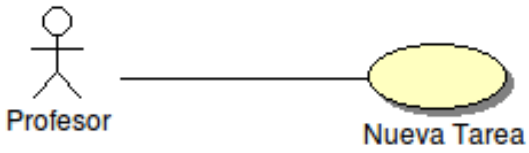


Ilustración 14: Diagrama de Casos de Uso Profesor

4.2.3.1 Nueva Tarea



Caso de Uso: Crear una Nueva Tarea

Actor: Profesor

Descripción: Solicita todos los datos necesarios para la creación de la nueva tarea. El profesor decidirá si desea subir código o no. En caso de escoger subir código, la aplicación le solicitará éste y realizará la comprobación de que no contenga errores de compilación. El código se podrá subir desde un fichero o ser escrito directamente. Una vez finalizado el proceso, esta será guardada en la Base de Datos y será accesible para todas las funciones que lo requieran.

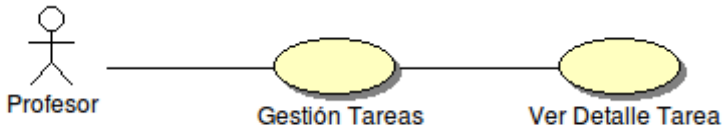
Requisitos: Completar todos los datos necesarios para la creación de la nueva tarea. Además el código subido por el profesor no ha de dar problemas al realizar su compilación.

Excepciones:

1.No se completa todos los datos obligatorios para la creación de la nueva tarea. La aplicación nos informará de los datos incompletos sin dejarnos avanzar, hasta que estos estén completos.

2.El código introducido por el profesor contiene errores de compilación. La aplicación al compilar el código y ver errores de compilación en éste, nos informará de la existencia de errores de compilación y no habilitará la opción de guardado de la tarea en la Base de Datos.

4.2.3.2 Ver Detalle Tarea



Caso de Uso: Ver los detalles de una Tarea

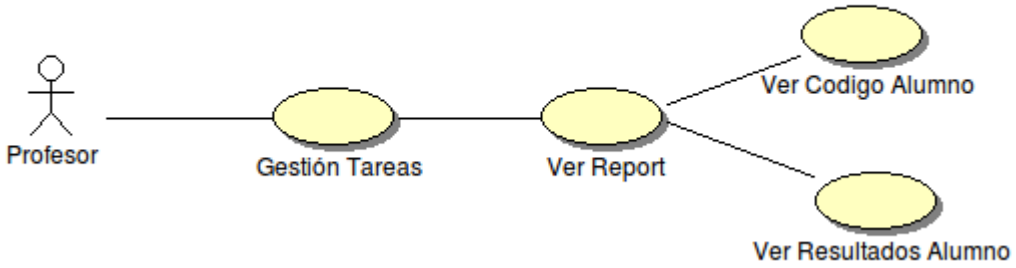
Actor: Profesor

Descripción: Se mostrarán varios datos sobre una tarea.

Requerimientos: Haber seleccionado una tarea.

Excepciones: -

4.2.3.2 Ver Report



Caso de Uso: Ver el *report* de las entregas de los alumnos para una tarea.

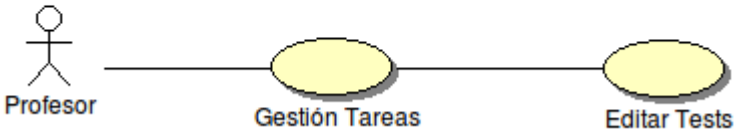
Actor: Profesor

Descripción: El profesor podrá visualizar las diferentes entregas para una tarea, efectuadas por los alumnos. En ellas podrá encontrar y visualizar el código subido por el alumno y los resultados de los juegos de pruebas asignados a esa tarea.

Requerimientos: Haber seleccionado una tarea. Además esa tarea ha de tener alguna entrega realizada.

Excepciones: -

4.2.3.3 Editar Test



Caso de Uso: Editar los Juegos de Prueba para una tarea.

Actor: Profesor

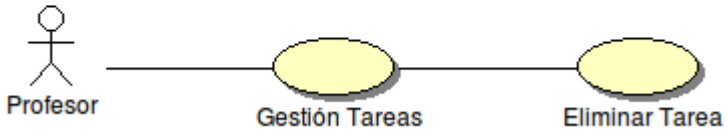
Descripción: El profesor podrá añadir o eliminar diferentes *tests* en las tareas que aun no se hayan inicializado. Se solicitara los datos necesarios para la creación del *test*, y dependiendo de si el profesor subió código para esa tarea o no, se ejecutará con el *test* añadido. Se obtendrá una salida para ese juego de pruebas, con el que comparar posteriormente la entrega del alumno. El *test* será guardado en Base de Datos y accesible a todas las funciones que requieran de él.

Requerimientos: Se debe haber escogido una tarea para añadir o eliminar los *tests.* Esta no debe haber iniciado para poder configurar los *tests.*

Excepciones:

1.Si la tarea está iniciada, no se dará la opción de añadir nuevos test. Tan solo se podrá visionar los ya introducidos.

4.2.3.4 Eliminar Tarea



Caso de Uso: Eliminar una Tarea.

Actor: Profesor

Descripción: Solicita una confirmación para eliminar la tarea de la Base de Datos. La aplicación eliminará de la Base de Datos la tarea, eliminando también todas las entregas, realizadas por los alumnos relacionados con la tarea.

Requerimientos: Se debe haber seleccionado una tarea y aceptar el mensaje de confirmación.

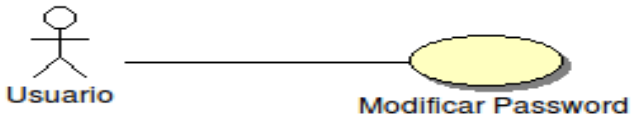
Excepciones:

1.No se acepta el mensaje de confirmación. La aplicación no eliminará tarea, seleccionada, de la base de datos hasta que no se acepte.

4.2.4 Casos de Uso Comunes

Como todos los actores dispondrán de los casos de uso, *logout* y modificar *password* se ha decidió explicar conjuntamente para ahorrar espacio, dado que no era necesario la explicación para los distintos actores. Cuando estos dos casos serán idénticos sea quien sea el actor.

4.2.4.1 Modificar Password



Caso de Uso: Modificar el *Password* del usuario con el que nos hemos *logueado*

Actor: Administrador, Profesor, Alumno

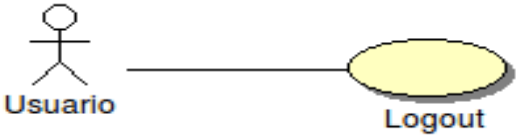
Descripción: Solicita introducir el nuevo *password,* además se pedirá que se repita para confirmar el nuevo *password.* La aplicación modificara en la Base de Datos a nuestro usuario para que la siguiente vez que intentemos el *login,* ya deberíamos introducir el nuevo *password.*

Requerimientos: Haber accedido a la aplicación con un usuario, sea cual sea el rol. Además el *password* y la confirmación de éste han de coincidir.

Excepciones:

1.La contraseña que se introduce es de un tamaño menor, o no coincide con la confirmación de contraseña. La aplicación retorna un aviso de error en la contraseña.

4.2.4.2 Logout



Caso de Uso: Logout

Actor: Administrador, Profesor, Alumno

Descripción: El usuario se *deslogue*a de la aplicación.

Requerimientos: Haber accedido a la aplicación con un usuario, sea cual sea el rol.

Excepciones: -

5. Aplicación.

En este apartado podrá consultar los diferentes aspectos del diseño de la aplicación. Por ello podrá consultar información de la base de datos, diagramas de clases y mapas de navegación.

5.1 Estructura Clases Java

La aplicación está estructurada en diferentes *packages* siguiendo la estructura modelo-vista-controlador, como se puede ver en la *Ilustración 15*.

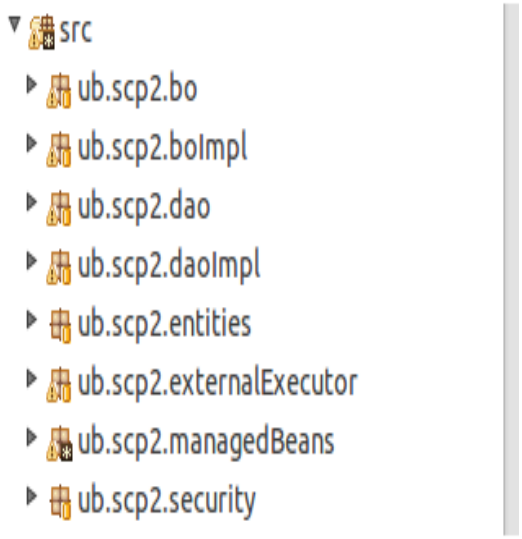


Ilustración 15: Estructura de packages

- ***bo:*** interfaces de la lógica de negocio. Este *package* contiene las diferentes interfaces que se utilizaran desde la capa de vista.

Ilustración 16: Ficheros asociados al package bo

- ***boImpl:*** implementación de la lógica de negocio. Este *package* contiene la implementación de las funciones de las clases *bo.* Se realiza esta separación a través de interfaces para facilitar que en un futuro posibles cambios en la implementación no afecten al resto de código.

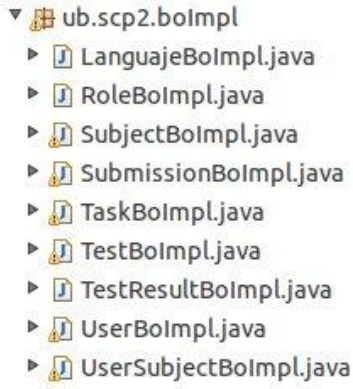


Ilustración 17: Ficheros asociados al package boImpl

- ***dao:*** interfaces de la capa de acceso a Base de Datos. Estas serán utilizadas en la implementación de la lógica de negocio (*boImpl).*

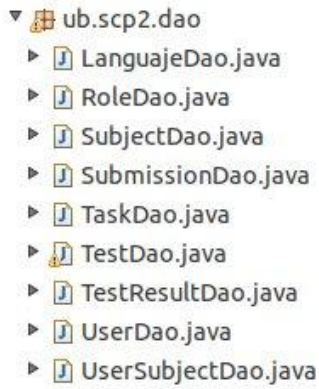


Ilustración 18: Ficheros asociados al package dao

- ***daoImpl:*** implementaciones de la capa de acceso a Base de Datos. Estas serán las encargadas de utilizar Hibernate para el trabajo con la Base de Datos.

Ilustración 19: Ficheros asociados al package daoimpl

- ***entities:*** Son los nombrados *POJO, Plain Old Java Object.* Reflejan las clases que tienen una relación directa entre el objeto y la tabla de la base de datos. Por ello se almacena también los ficheros de *mapping de* Hibernate.

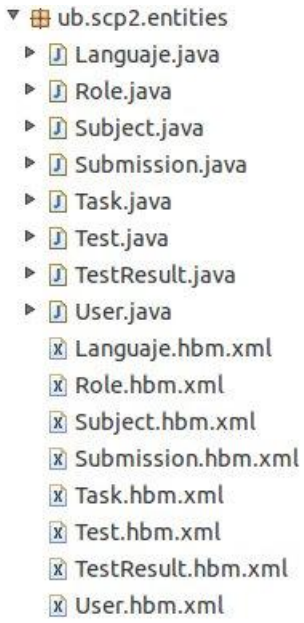


Ilustración 20: Ficheros asociados al package entities

- ***ManagedBean:*** Se trata de los *beans* de la capa de vista, interfaz gráfica. Son las clases que controlan los elementos de la interfaz gráfica.

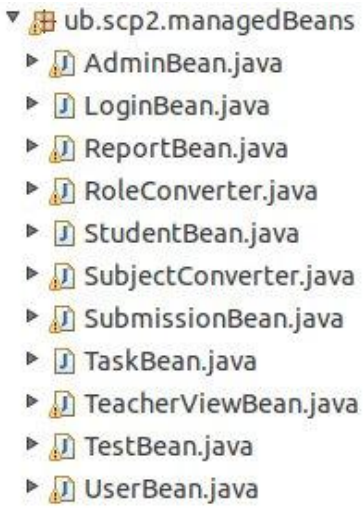


Ilustración 21: Ficheros asociados al package managedBean

5.2 Diagrama de Clases

A continuación se detallaran los diagramas de clases, cada uno de ellos pertenece a una vista diferente, vista Administrador, vista Alumno y realizar entrega, vista Profesor y nueva tarea. Estos diagramas nos definen las distintas clases que participan internamente para el funcionamiento de esa vista, junto sus atributos y métodos implementados.

Se ha decidido crear cinco diagramas de clases, cada uno orientado desde su respectivo controlador de la vista. Dado que al ser una aplicación con un gran número de clases colocadas a distinto nivel, por la estructura utilizada, explicada en el apartado *4.1*. Se dispone de clases básicas entidad, *managedBean* (controladores de la vista), *Bussines Objects* (BO) y *Data Acces Object* (DAO).

5.2.1 Diagrama de Clases Administrador

Diagrama de Clases resultante de la vista Administrador, *Ilustración 22*. En esta se le permite al administrador todas sus funcionalidades.

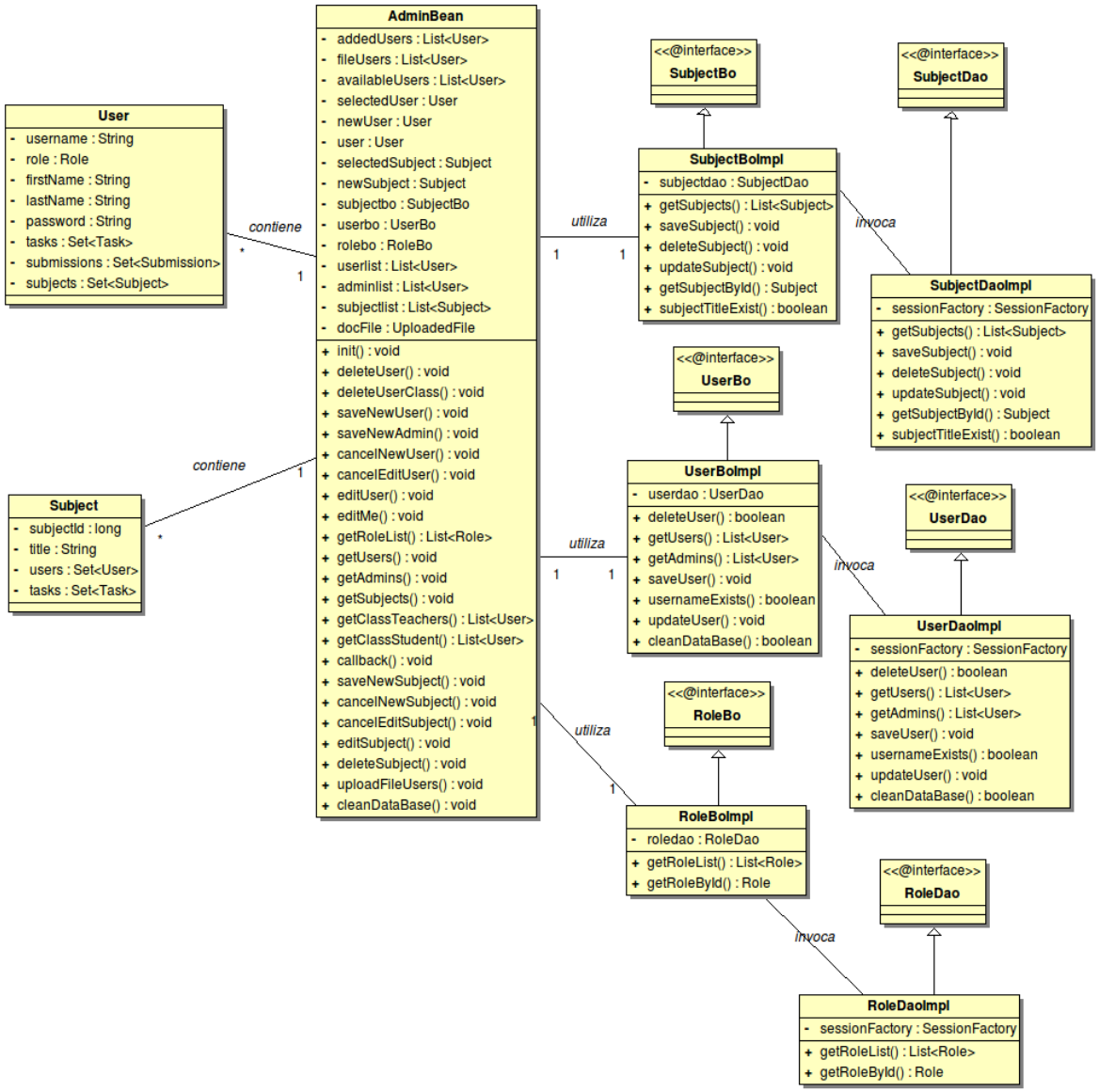


Ilustración 22: Diagrama de Clases Administrador

5.2.2 Diagrama de Clases Alumno

Diagrama de Clases resultante de la vista Alumno, *Ilustración 23,* donde se muestran todas las funcionalidades de las que dispone el alumno en su página principal.

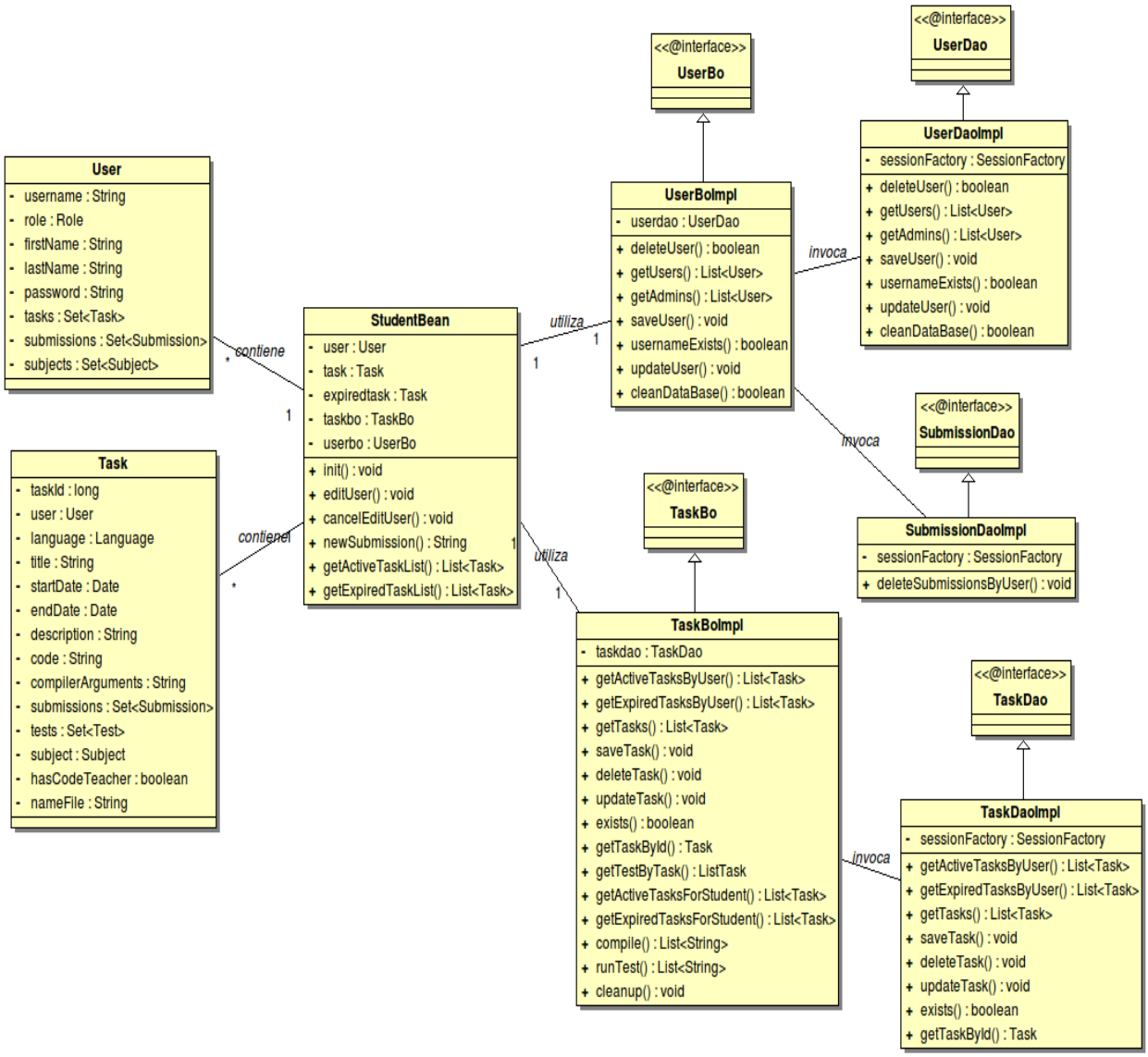


Ilustración 23: Diagrama de Clases Alumno

La funcionalidad de realizar la entrega no está reflejada en este diagrama de clases, dado que esta se realiza a través de otra vista, por lo tanto con otro *managedBean.* En el diagrama de clases siguiente se muestra las funcionalidades en la vista de realizar entrega, *Ilustración 24*.

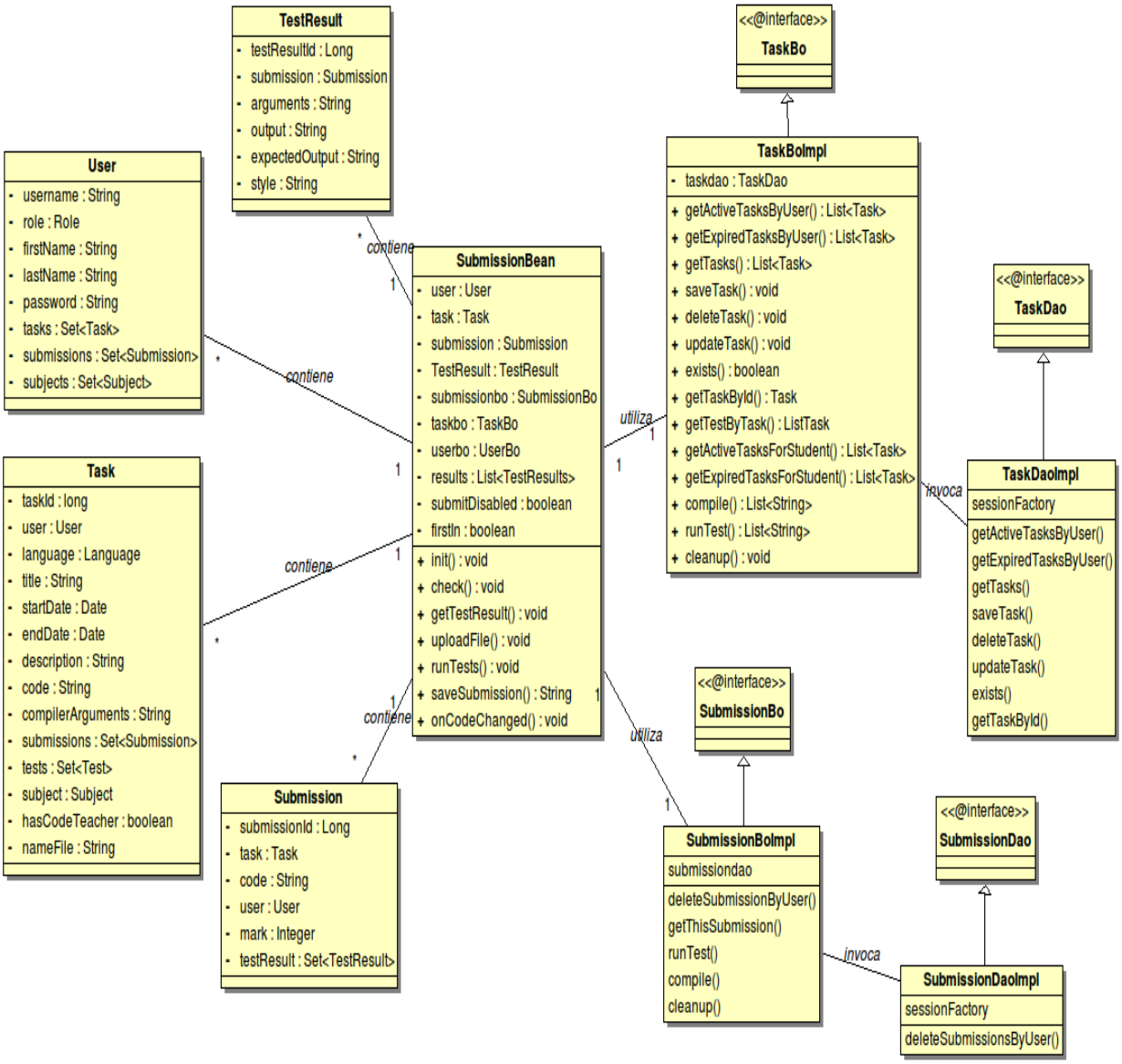


Ilustración 24: Diagrama de Clases Alumno-Realizar Entrega

5.2.3 Diagrama de Clases Profesor

Diagrama de Clases de la vista principal de profesor, *Ilustración 25,* en el encontramos todas las funcionalidades que se permiten al profesor.

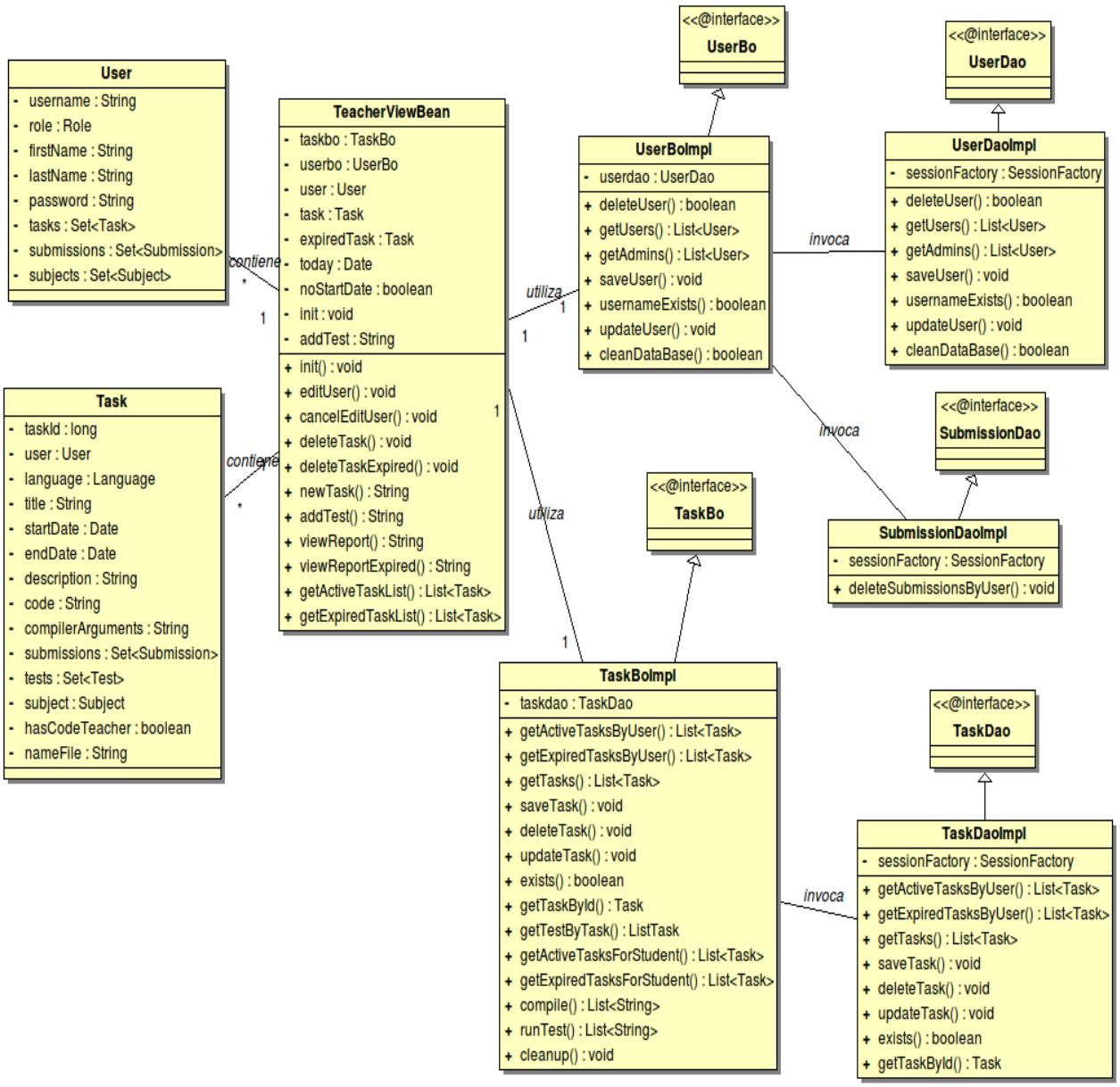


Ilustración 25: Diagrama de Clases Profesor

Hay funcionalidades del profesor que requieren otras vistas, controladas por otro *managedBean*, estas vistas son: creación de nueva tasca, diagrama en la *Ilustración 26,* añadir *Test* o ver el *Report.* Se muestra el diagrama de clases de la vista nueva tasca porque en comparación a las otras dos esta es más compleja.

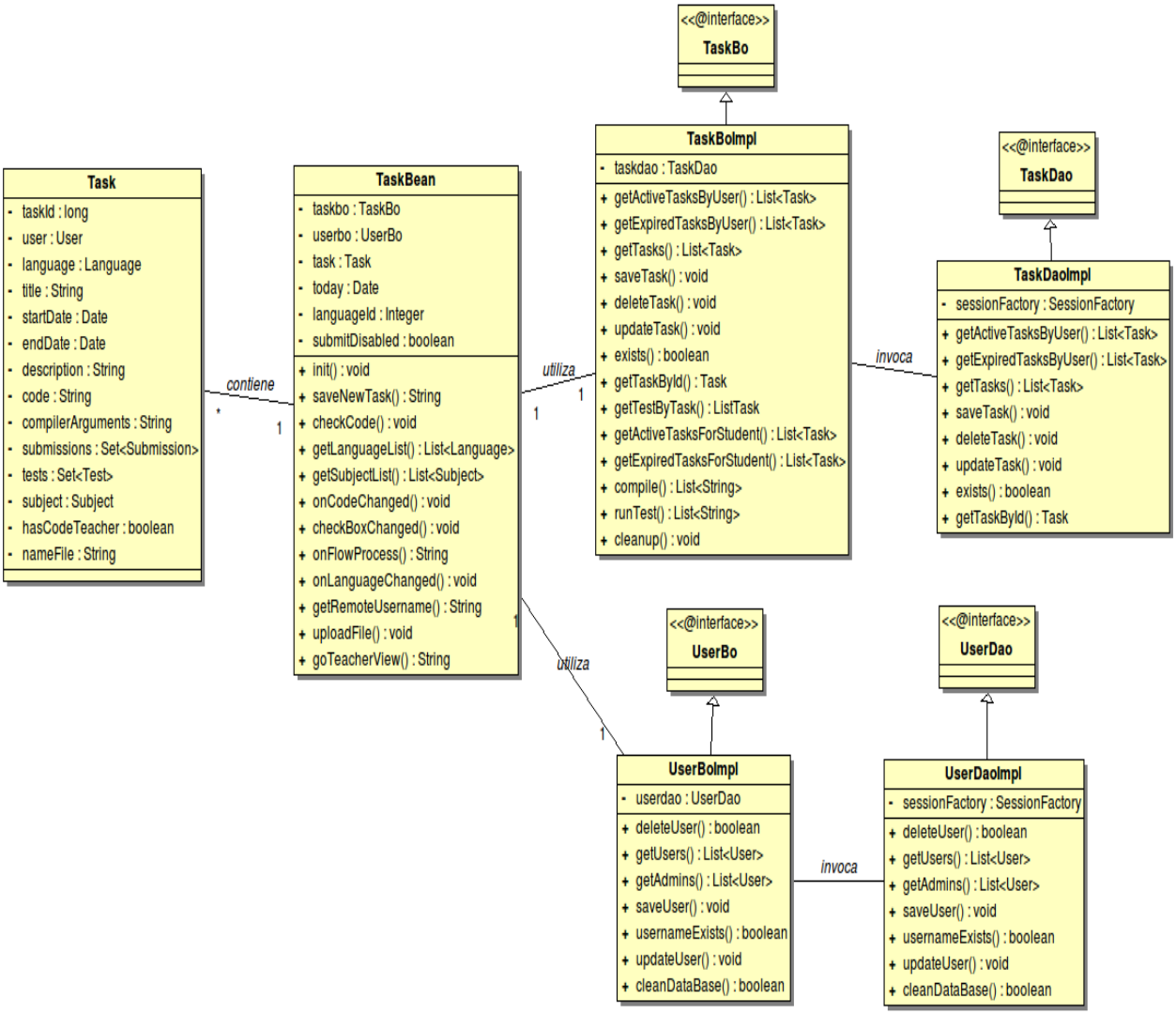


Ilustración 26: Diagrama de Clases Profesor-Nueva Tarea

5.3 Base de Datos

A continuación se detallara la estructura de la Base de Datos, mostrándose el diagrama Entidad-Relación (*Ilustración 27)* de dicha Base de Datos, esta será la utilizada en el proyecto anterior, analizado en el apartado *2.*, con algunas modificaciones que se comentaran.

5.3.1 Modelo E-R

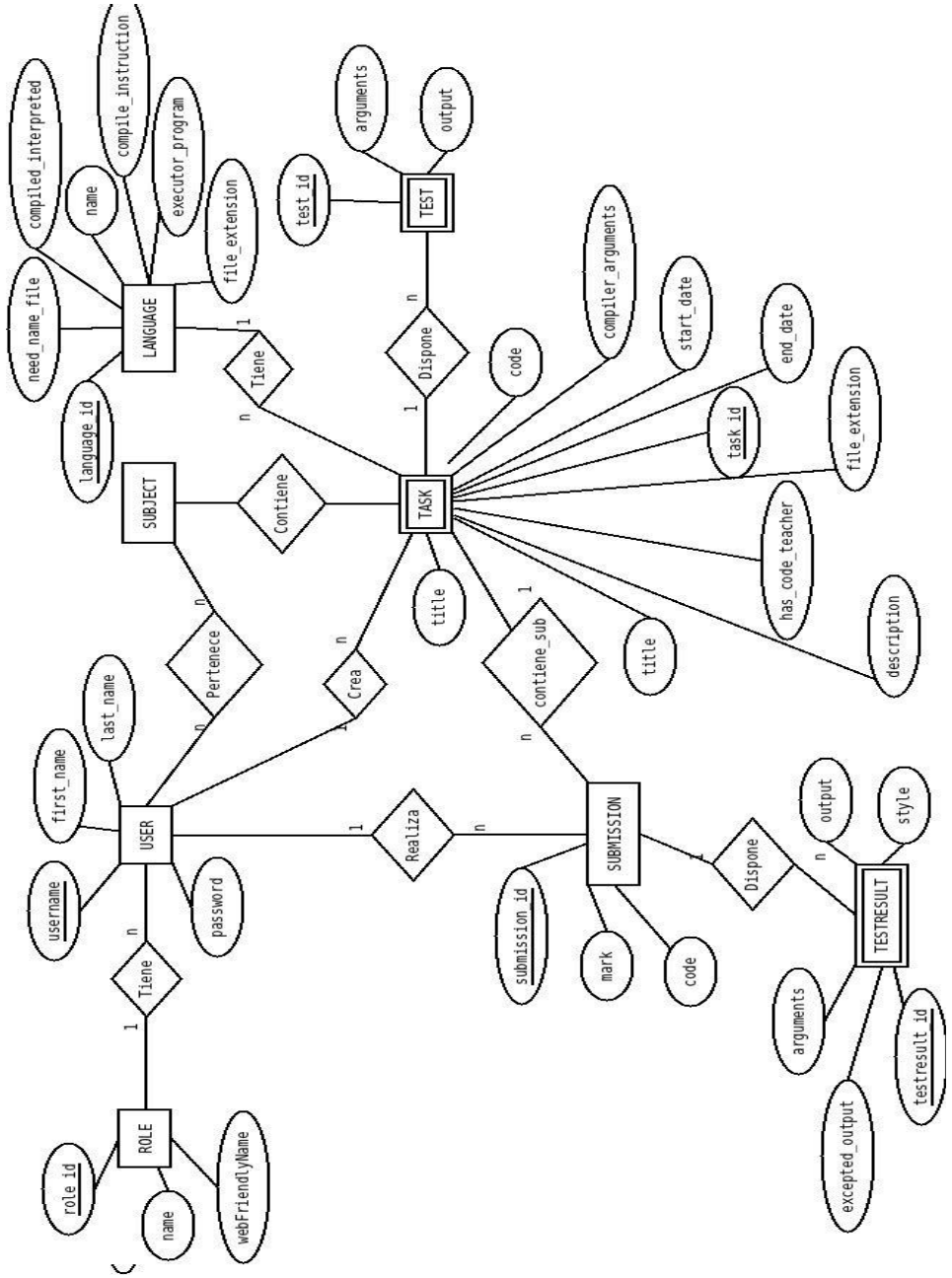


Ilustración 27: Diagrama ER del SCP Versión Actual

5.3.2 Comentarios.

Con tal de mejorar la aplicación se ha seguido utilizando la Base de Datos anterior, pero se han realizado cambios. Estos se comentaran a continuación.

Nueva entidad TestResult:

TESTRESULT: Entidad que almacenara los resultados de las entregas del Alumno.

- *testresult_id:* Campo que identifica una entrada en la entidad Testresult. Está representado como un entero y no pude ser nulo.
- arguments: Argumentos de entrada para la ejecución de los tests.
- output: Resultado de la ejecución de los tests.
- style: Campo para identificar si una entrega a sido correcta o incorrecta.
- *excepted_output:* Resultado esperado de la ejecución de los *tests.* Este campo se utiliza para la comparación de los resultados obtenidos con los esperados.

Modificación en la entidad Language, se han añadido nuevos campos para facilitar el registro de nuevos lenguajes. Además esta se ha relacionado con la entidad Task.

LANGUAGE:

- *language_id*: Campo que identifica una entrada en la entidad Language. Está representado como un entero y no puede ser nulo.
- *name*: Nombre del Language. Está representado con un Varchar de máximo 255 caracteres.
- file_extension: Campo que identifica la extensión de los ficheros para ese lenguaje
- compiled_interpreted: Campo que nos indicara si el lenguaje es compilado o interpretado. Se representa con un boolean.
- compiled_instruction: Instrucción para compilar el código para este lenguaje. Puede ser nulo dependiendo del campo compiled_interpreted.
- executor_program: Intrucción de ejecución del código para este lenguaje.
- *need_name_file:* Campo que nos indicara si el fichero a compilar necesita un nombre en especifico, por ejemplo en el caso del lenguaje Java.

Nuevos campos en la entidad Task:

TASK:

- *has_code_teacher:* Campo que nos indica si el profesor ha decidido subir código o no.
- *name_file:* Campo relacionado con el campo *need_name_file* de la entidad Language, dado que si esta está activada se deberá introducir el nombre del fichero a compilar.

5.4 Mapa de Navegación

A continuación, en la *Ilustración 28,* se muestra el diagrama de mapa de navegación, que nos muestra las distintas páginas a las que podemos acceder dependiendo de la acción que realizamos.

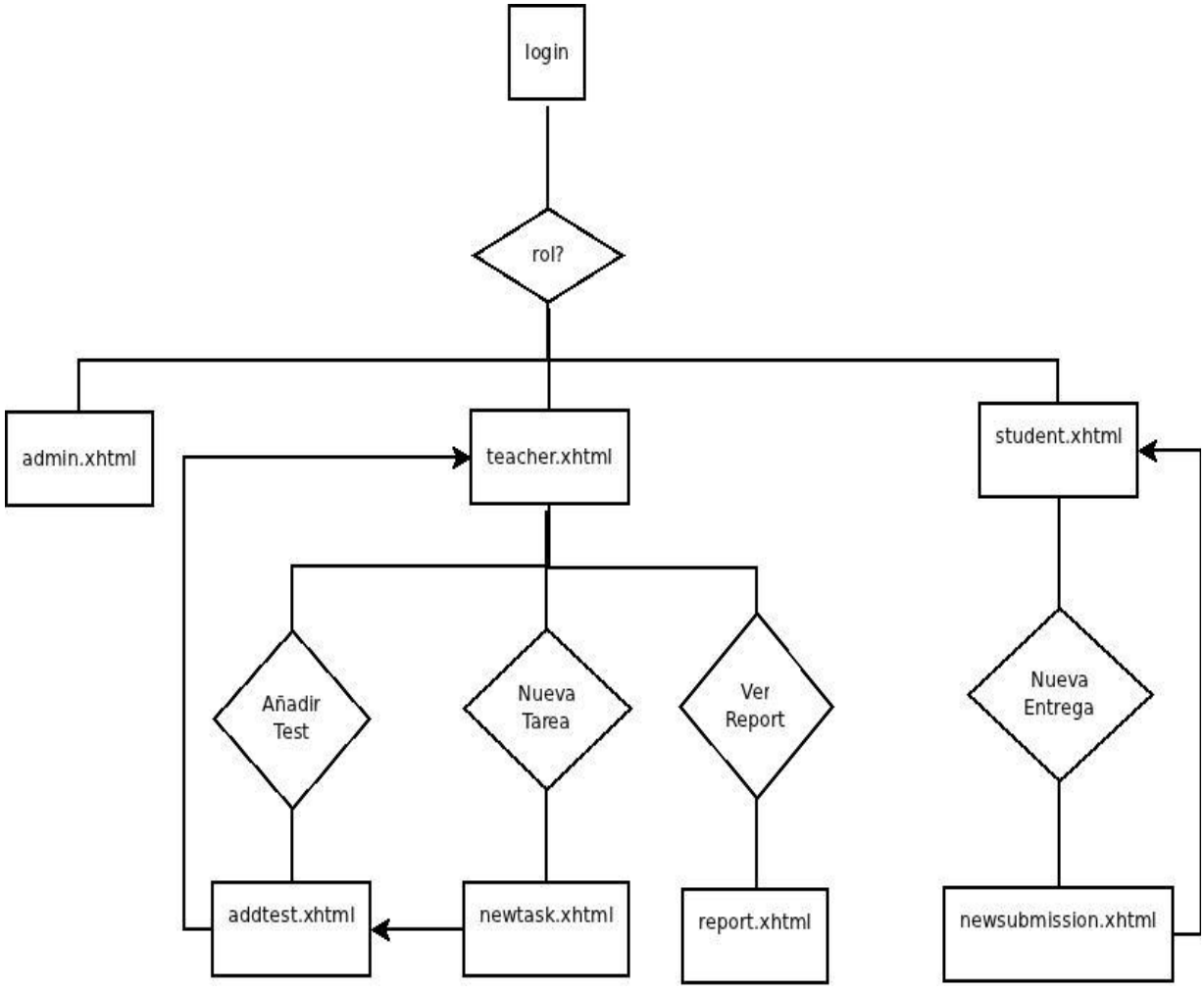


Ilustración 28: Mapa de Navegación

6. Pruebas y resultados

6.1 Pruebas con distintos navegadores

Para comprobar que la visualización de la web es correcta, se han realizado pruebas en varios navegadores conocidos.

A continuación se muestran la vista del administrador en cada uno de los distintos navegadores. Hay que comentar que se ha probado para todas las vistas de la aplicación y no se han visto diferencias apreciables entre ellos. Tan solo en el caso de Internet Explorer, tal como se puede comprobar en la *Ilustración 29*, se ve una cierta diferencia en cuanto las proporciones.

6.1.1 Internet Explorer

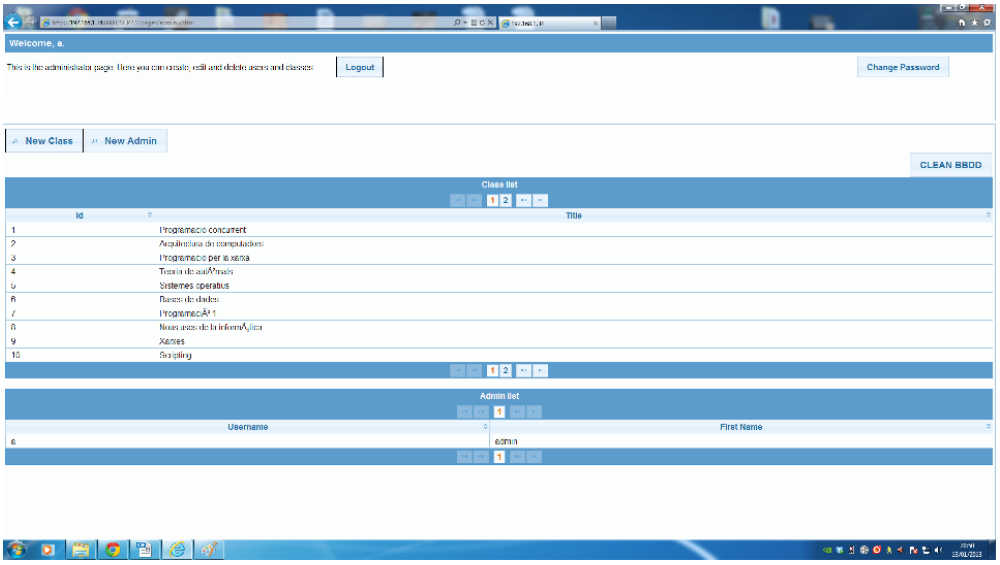


Ilustración 29: Vista Administrador desde Internet Explorer

6.1.2 Google Chrome

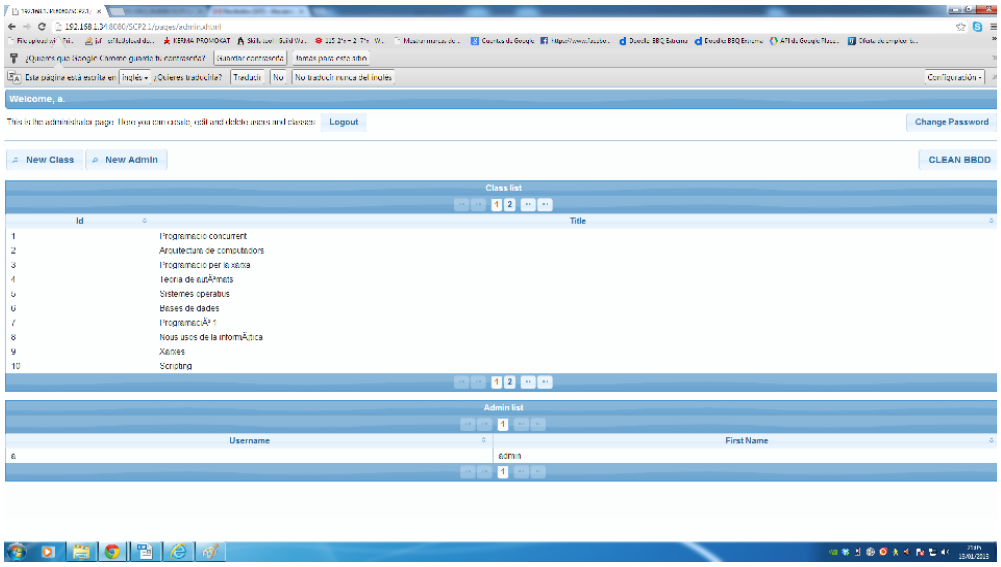


Ilustración 30: Vista Administrador desde Google Chrome

6.1.3 Mozilla Firefox

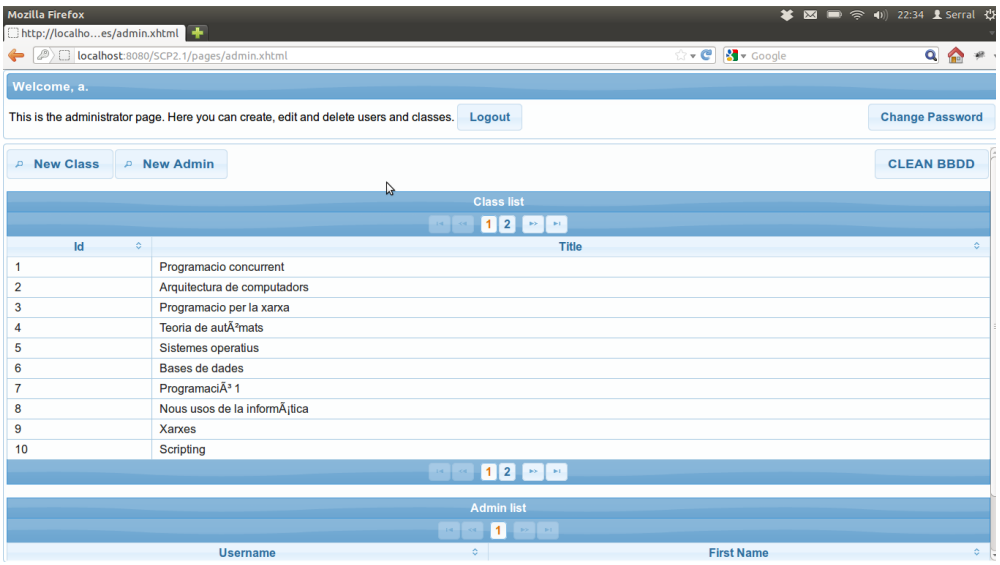


Ilustración 31: Vista Administrador desde Mozilla Firefox

6.2 Pruebas de vistas

Como se describió, la aplicación dispone de tres vistas distintas dependiendo del rol del usuario. Para estas pruebas se decidió comprobar el funcionamiento de las distintas acciones de las que dispone cada usuario en su vista.

6.2.1 *Login*

Prueba1. Realizar el *Login* con un usuario correcto

- Situación esperada: Se espera que tras introducir el nombre de usuario y la contraseña correcta, se nos redirija a la vista correspondiente al rol de nuestro usuario.
- Situación obtenida: Se nos permite el acceso al sistema, se redirige a la vista correspondiente.

Prueba2. Realizar el *Login* con un usuario incorrecto

- Situación esperada: Se espera que tras introducir el nombre de usuario correcto con una contraseña incorrecta, el sistema no nos permita el acceso.
- Situación obtenida: El sistema no nos permite el acceso.

6.2.2 Vista Administrador

Prueba1. Crear una nueva Asignatura

- Situación esperada: Se espera que tras introducir el titulo y el fichero con los usuarios, el sistema almacene en la base de datos la nueva asignatura y se nos informe.
- Situación obtenida: El sistema realiza correctamente la creación de la asignatura y de los usuarios que aun no estuvieran registrados. Por el contrario no se nos informo de lo sucedido.

  Este error fue solucionado tras añadir un mensaje informativo.

Prueba2. Crear una nueva Asignatura sin titulo

- Situación esperada: Se espera que no se registre la asignatura en la base de datos y se nos informe conforme falta el titulo de la asignatura.
- Situación obtenida: El sistema nos muestra un error conforme el titulo es obligatorio, sin realizar el registro de la asignatura.

Prueba3. Crear un nuevo Administrador

- Situación esperada: Se espera que tras introducir todos los datos necesarios, el sistema almacene en la base de datos el nuevo usuario administrador.
- Situación obtenida: El sistema realiza correctamente la creación y registro del nuevo administrador en la base de datos.

Prueba4. Crear un nuevo Administrador sin introducir todo los datos.

- Situación esperada: Se espera que tras intentar registrar un usuario administrador sin haber introducido todos sus datos, el sistema nos avise de la falta de uno de ellos.
- Situación obtenida: El sistema nos avisa del campo que hemos dejado vacío.

Prueba5. Editar Asignatura

- Situación esperada: Se espera que se muestre una lista con los usuarios que pertenecen a esa asignatura, así como las acciones de crear nuevo usuario, edita, eliminar o remover de la asignatura al usuario.

  En esta prueba comprobamos que las acciones nombradas también funcionan.
- Situación obtenida: Se nos muestra la lista de usuarios de la asignatura, así como la opción de añadir nuevo usuario. Si seleccionamos a algún usuario existente se nos permite realizar las acciones de eliminar, remover o editar a dicho usuario.

  Se comprueba que funcionan correctamente las funciones con los usuarios.

Prueba6. Editar Usuario Administrador, de la lista

- Situación esperada: Se espera que se nos permita modificar atributos del usuario administrador. Únicamente no se nos debe permitir el rol ni el *username*. Estas modificaciones se han de actualizar en la base de datos.
- Situación obtenida: Se permite la modificación de los atributos del usuario, pero sin la opción de modificar el atributo *username.* Los cambios son actualizados correctamente en la base de datos.

Prueba7. Limpiar Base de Datos

- Situación esperada: Se espera que se muestre un mensaje de confirmación, una vez aceptado el sistema limpiara la base de datos dejando únicamente, los usuarios administrador y los lenguajes de programación.
- Situación obtenida: El sistema nos muestra el mensaje de confirmación, una vez

aceptado se elimina todos los registros de la base de datos. Se deja únicamente los usuarios con rol administrador y los registros de la tabla lenguaje.

6.2.3 Vista Alumno

Prueba1. Realizar entregados

- Situación esperada: Se espera que se nos redirija a una nueva vista donde realizar la entrega. Si en esta entrega ya se había subido algún código, este debe aparecer con sus resultados.
- Situación obtenida: Se nos redirige a la nueva vista donde se puede realizar la entrega. Y si previamente ya se había entregado se nos muestra el código anterior y sus resultados.

Prueba2. Subir código en un fichero

- Situación esperada: Se debe actualizar el cuadro de texto con el contenido del fichero.
- Situación obtenida: Se actualiza el cuadro de texto con el contenido del fichero. Pero al realizar un refresco de la página este aparece en blanco.

  Este error fue solucionado con un control sobre el acceso a la página. Para controlar si era la primera vez y debíamos cargar el código antiguo o por el contrario se accedía tras la subida de fichero por lo que debíamos quedarnos el código actual.

6.2.4 Vista Profesor

Prueba1. Crear una nueva tarea

- Situación esperada: Se espera que el sistema nos envié a una nueva vista donde realizar la nueva tarea.
- Situación obtenida: Se nos envió a una nueva vista en la que se dispone de los datos para crear una nueva tarea.

Prueba2. Registrar una nueva tarea con algún atributo en blanco.

- Situación esperada: Se espera que el sistema no nos deje avanzar si no se han rellenado todos los campos.
- Situación obtenida: El sistema nos informa con un mensaje los campos vacíos, sin dejarnos avanzar.

Prueba3. Registrar una nueva tarea con código

- Situación esperada: Se espera que el sistema nos permita la subida del código por fichero o introducido manualmente, tras esto y comprobar que el código no contiene errores de compilación. Se debería poder acceder a registrar la tarea. Una vez registrada se debería redirigir a la vista de añadir test.

  Si existieran errores de compilación se nos debería de informar.

- Situación obtenida: El sistema nos permite la subida del código ya sea desde fichero como introducido manualmente. La comprobación del código nos informa si éste tiene errores o si esta correcto, una vez nos comunica que es correcto se nos habilita el botón de *Submit* con el que realizaremos el guardado en la Base de datos. Una vez registrado accedemos a la vista añadir *test.*

Prueba4. Añadir *test* en tarea no empezada.

- Situación esperada: Se espera que el sistema nos muestre 4 opciones distintas para asignar argumentos de ejecución. Según si se ha subido código o no, se nos debería permitir ejecutar los argumentos con el código o guardar los argumentos automáticamente.

- Situación obtenida: El sistema nos muestra diferentes tipos de argumentos, si hemos subido código en esta tarea se nos permite ejecutar los argumentos con el código y obtener un resultado, con el cual luego podemos comparar al del alumno. Si no hemos subido código guardamos en la base de datos el *test.*

Prueba5. Visualizar el *report* de una tarea.

- Situación esperada: Se espera que el sistema nos redirija a una vista, en la que podremos comprobar todas las entregas de los Alumnos con sus respectivos códigos y resultados a nuestros *tests.*

- Situación obtenida: El sistema nos muestra esta vista para que se pueda ver las entregas de los alumnos. Estas contienen los códigos y los resultados, aunque los resultados no se está marcando correctamente los correctos y los incorrectos.

  Se soluciono el error de correctos y incorrectos asignando el atributo *style* en esta tabla para que muestre el color correspondiente.

7. Dificultades

Con el desarrollo de las nuevas funcionalidades de la aplicación web, se han adquirido una serie de conocimientos, que inicialmente, presentaron una curva de aprendizaje bastante profundizada ya que de repente se tuvieron que utilizar una serie de tecnologías distintas y totalmente nuevas para mí.

La primera barrera que se encontró fue a la hora de realizar el análisis y comprensión del código del proyecto anterior. Para conseguir avanzar en las nuevas funcionalidades se requería entender completamente el funcionamiento de otras, así como su diseño. Esto era una tarea complicada por la poca documentación de la que disponía el código.

La siguiente barrera fue la comprensión y aprendizaje de las tecnologías utilizadas. Esto propicio un beneficio para mi persona, dado que estas eran tecnologías apenas conocidas por mi pero con un amplio uso en el mundo empresarial. Pese a ser una tarea complicada, el hecho de que estas herramientas fueran muy utilizadas me ayudo a la hora de realizar la búsqueda de información. Esto proporciono un aprendizaje relativamente más rápido de lo esperado.

Una vez superadas estas barreras la parte más complicada fue la implementación de las nuevas funcionalidades, así como analizar cuáles de estas eran necesarias.

8. Conclusiones

Al inicio del proyecto se declararon unos objetivos a los cuales llegar una vez terminado. Estos objetivos principalmente centrados en la mejora de la versión anterior del proyecto, englobando los aspectos a mejorar del proyecto anterior, así como nuevas funcionalidades que añadir. Si nos centramos en los distintos tipos de usuario:

- **Administrador:** Como usuario administrador, se encontró que existía un gran problema a la hora de gestionar a los usuarios. Tanto en el momento de darlos de alta, se debía efectuar de usuario a usuario, cosa que en una aplicación con un número de usuario muy elevado es un verdadero inconveniente. Esto se soluciono permitiendo al administrador crear una asignatura con la subida de un fichero, el cual se puede extraer del Moodle, facilitando así tanto la creación de nuevos usuarios, como la asignación de estos a las asignaturas en las que están matriculados.

  Otro de los problemas encontrados, también relacionados con la gestión de usuarios, fue el hecho de que en el diseño de la página del administrador, se nos mostraba toda una lista de usuarios. Esto podía llegar a provocar que con un gran número de usuarios resultara difícil encontrar y administrar cualquiera de ellos. Para ello se re-diseño la página del administrador. Actualmente se muestra una lista de asignaturas y una lista de administradores. Al seleccionar una de las asignaturas se nos mostrara todos los alumnos asignados a esta, facilitando así el acceso a los usuarios.

  Además se añadió una nueva funcionalidad para facilitar al administrador la limpieza de la base de datos. A través de un simple botón se permite el *reset* de toda la base de datos. Se deja únicamente los registros necesarios, como por ejemplo los lenguajes.

- **Alumno:** Desde el usuario alumno, tras realizar el análisis, se encontró con un único problema, pero a la vez molesto. Este principalmente erradicaba a la hora de realizar una entrega donde el alumno debía introducir manualmente el código en un cuadro de texto. Para ello se habilito una opción de subida de fichero, en la cual se accedía al fichero y se actualizaba el cuadro de texto con el contenido de este, permitiendo también la edición de éste desde el propio cuadro de texto.

  También se añadió una nueva funcionalidad, se habilitara al alumno poder modificar su *password*. Anteriormente el alumno debía pedir al administrador que modificara su *password*, dando esto una faena extra e innecesaria al administrador.

- **Profesor:** Tras el análisis realizado respecto al profesor, se encontraron bastantes problemas de funcionalidad. Uno de los principales problemas era a la hora de poder ver las entregas de los alumnos, de estos tan solo podíamos ver el tanto por ciento de

aciertos de los *tests*, así como el código subido por el alumno. Pero no se podía visualizar los resultados de los *tests*. Para ello se añadió en la base de datos un nueva entidad que registrara estos resultados, y se habilito el acceso a estos desde la opción del profesor ver *report*.

Si seguimos con el tema de los *reports* uno de los problemas que solucionar, era el no poder visualizar los *reports* de las entregas que ya habían finalizado. Cabe comentar que normalmente el profesor visualizará los resultados de los alumnos una vez ya no se puedan realizar más entregas. Para ello se habilito la opción de ver *report* dentro de las tareas expiradas.

Otro de los principales problemas del profesor era en la creación de una nueva tarea. En la cual el profesor debía subir el código siempre, esto obligaba a que los resultados de los alumnos estuvieran o no correcto. Se pensó que se podría dar el caso en que el profesor no quisiera subir el código y únicamente quiera ver los resultados que le dan al alumno los *tests.* Por ello se habilito una opción en donde el profesor puede decidir si subir o no el código. Esta introducción obligó a modificar todos los aspectos relacionado con las entregas de los alumnos y sus resultados.

También relacionado con la creación de nueva tarea, encontramos que a la hora de añadir los *tests* para la tarea que se está editando, se nos restringía a un solo tipo de argumento. Es por ello que se habilitó la opción de seleccionar entre diferentes tipos de entrada de argumentos, ya fuera argumentos por teclado, argumentos por teclado con fichero etc.

Para conseguir ampliar el funcionamiento de la aplicación con nuevos lenguajes, se amplió la entidad *Language* con nuevos atributos que nos faciliten la incorporación y el uso de un nuevo lenguaje. Por ello también se modificó el diseño de la vista de nueva tarea, mostrando u ocultando campos que rellenar que dependen del lenguaje escogido.

Otro problema solucionado, menos importante, pero a la vez necesario para la realización de una versión más robusta. Fue el hecho de poder añadir *tests* a tareas ya empezadas. Para ello se inhabilito esta opción una vez empezada la tarea, momento en el que los alumnos pueden comenzar a realizar sus entregas.

A finales de este proyecto, la aplicación funciona cumpliendo los objetivos específicos que se definieron inicialmente. Al mirar un poco más allá, se le pueden aplicar mejoras a la aplicación, mejoras que no se plantearon en los objetivos iniciales por el tiempo establecido para la realización de este proyecto, estas mejoras pueden ser:

- El aspecto de la seguridad del presente proyecto, se está desarrollando en un proyecto paralelo. En él está trabajando un compañero de la universidad.

  Para mejorar la seguridad, se está implementando las funciones que realizan el compilado y ejecución del código en un servidor aparte. Estas funciones están pensadas para desarrollarlas con la tecnología RMI, y así poder realizar una correcta comunicación entre ambos servidores.

- Otra posible mejora, es el comportamiento de los resultados. Actualmente tan solo se dispone de la posibilidad de que los resultados del alumno sean capturados de la salida estándar del programa. La mejora consistiría en ofrecer la posibilidad de que el resultado del alumno fuera un fichero, ya fueran ficheros de texto, imágenes u otros contenidos. Para ello se debería almacenar el fichero en el servidor y ofrecer un *link* donde el usuario se pueda descargar el fichero.

- Otra funcionalidad, pequeña pero muy importante, es el soporte de la aplicación para diferentes idiomas. Para ello se debería modificar los ficheros *.xhtml*. En lugar de disponer de texto *harcoded* se debería utilizar una referencia a un fichero con las *strings* de cada idioma. Así como un *widget* con el cual seleccionar el idioma.

9. Bibliografía

- http://docs.jboss.org/hibernate/orm/3.3/reference/en/html/index.html
- http://www.hibernate.org/
- http://www.javaserverfaces.org/get-started
- http://primefaces.org/
- http://stackoverflow.com/
- http://www.springsource.org/get-started
- http://www.javabeat.net/2007/05/introduction-to-java-server-faces/
- http://en.wikipedia.org

10. Anexos

10.1 Manual de instalación

Para la correcta instalación del SCP, serán necesarios los siguientes requisitos software:

- Java 1.6+
- Tomcat 7+
- Compiladores o Interpretes de los lenguajes que se vayan a utilizar, de momento solo está en funcionamiento C. Por lo tanto necesitaremos gcc 4.6+
- bash 4.2+
- MySQL 5.5+

Una vez disponemos de los requisitos de software, hay que seguir los siguientes pasos:

- **Base de Datos**

El primer paso es la creación de la base de datos. Para ello se ha de crear un usuario con todos sus permisos sobre una base de datos y ejecutar el script create_db.sql, ubicado en el directorio */SCP2.2/WebContent/WEB-INF/conf.* Esto se realiza con las siguientes instrucciones en MySQL:

```
mysql > create user 'user '@'localhost ' identified by ' useruseruser ';
mysql > grant all privileges on scp2 .* to 'user '@'localhost ';
mysql > source create_db.sql
```

Una vez se tenga configurada la base de datos, debemos editar el fichero *applicationContext.xml,* ubicado en el directorio */SCP2.2/WebContent/WEB-INF/applicationContext.xml:*

```
< bean id =" dataSource " class =" org . springframework . jdbc . datasource .
DriverManagerDataSource ">
< property name =" driverClassName " value =" com . mysql . jdbc . Driver "
/ >
< property name ="url " value =" jdbc:mysql: // localhost:3306 / scp2 " / >
< property name =" username " value =" user " / >
< property name =" password " value =" useruseruser " / >
</ bean >
```

Se ha de configurar las propiedades ***url, username*** *y* ***password.*** Se debe asignar el valor del usuario que se creó en la base de datos.

- **Desplegar la Aplicación**

Se ha de copiar el fichero war, fichero.war, en el directorio de Tomcat, webapps. Habitualmente éste se encuentra en */opt/tomcat/webapps.*

Además, se tendrá que comprobar que el fichero de configuración de Tomcat, server.xml, tenga activadas las opciones ***unpackWars=true*** y ***autoDeploy=true***.

En el fichero de creación de la base de datos, incluye la creación de un usuario con perfil administrador. Este usuario se podrá utilizar para la creación de los demás usuarios, asignaturas etc.

10.2 Manual Administrador

A continuación encontrara el manual de administrador que le ayudara a entender la aplicación y que operación realizar en cada momento.

Página Principal.

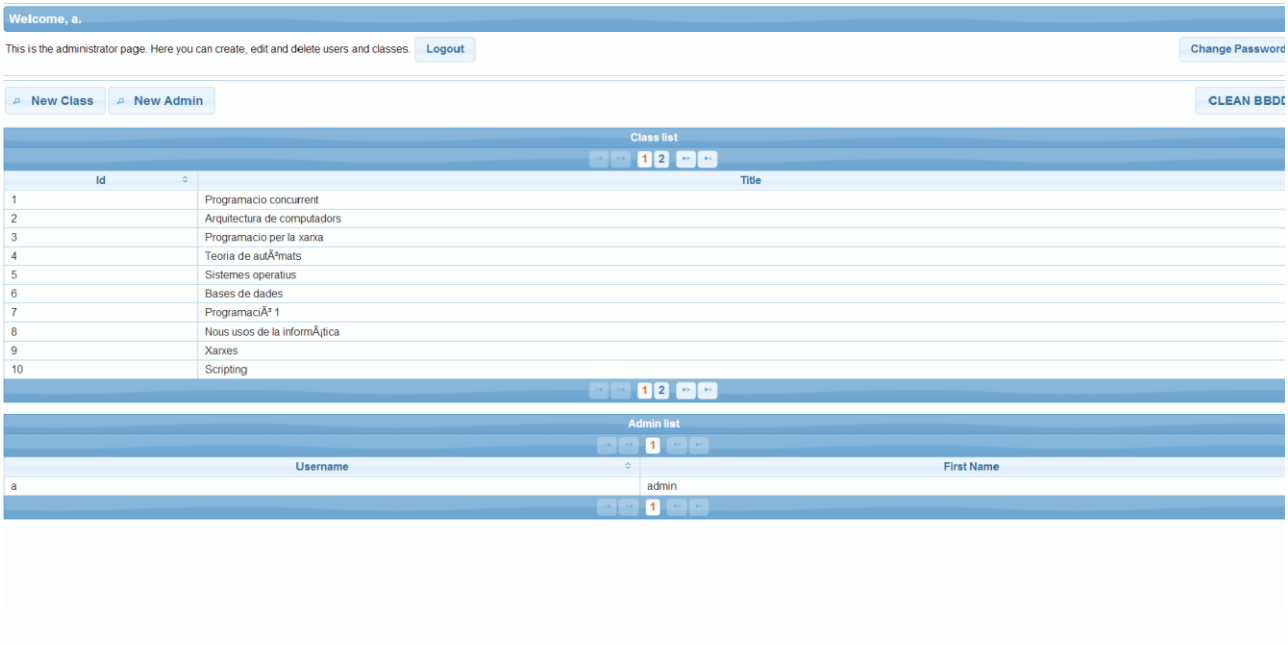


Ilustración 1: Página Principal Administrador

Acciones Posibles:

- **New Class**: Lanza un dialogo en el que deberemos introducir el nombre de la asignatura y elegir un fichero con los alumnos que se quieran crear y añadir a la asignatura. El fichero se puede extraer del Moodle y debe tener el siguiente formato, *Ilustración 2*:

```
Nom,Cognoms,Número ID,Institució,Departament,Correu electrònic,Total del curs
Pablo,Almirante Ruiz,niub38293049,Universitat de Barcelona,ENGINYERIA TÈCNICA EN INFORMÀT,paalruiz7@alumnes.ub.edu,-
Alberto,Santos Plazas,niub39283948,Universitat de Barcelona,ENGINYERIA TECNICA EN INFORMAT,alsaplaz9@alumnes.ub.edu,-
Juan Pedro,Olmos Tejedora,niub39182738,Universitat de Barcelona,ENGINYERIA TECNICA EN INFORMAT,jpolteje4@alumnes.ub.edu,-
Estefania,Moreno Pereira,niub39203920,UB,ENG.TEC.INFORM.SISTE,esmopere3@alumnes.ub.edu,-
Alex,Garcia Lopez,niub58493826,Universitat de Barcelona,,algarlope1@alumnes.ub.edu,-
Eudald,Perez Montes,niub19328593,Universitat de Barcelona,ENGINYERIA TÈCNICA EN INFORMÀT,eupemont2@alumnes.ub.edu,-
Jose Jaime,Luque Robles,nub5623,Universitat de Barcelona,Matemàtica Aplicada i Anàlisi,jjaime.luque@ub.edu,-
```

Ilustración 2: Ejemplo de formato fichero de usuarios

- **New Admin:** Lanza un dialogo para la creación de un nuevo usuario administrador. En éste se deberá rellenar todos los datos solicitados y seleccionar la opción *Create* para realizar el registro de este nuevo usuario.
- **Clean BBDD:** Acción para realizar una limpieza de la base de datos. Se dejara la base de datos por defecto, con los usuarios administrador y los lenguajes de programación.
- **Change Password:** Acción para modificar el password de nuestro usuario.

Junto a estas acciones aparecen dos listas:

- Lista de Asignaturas
- Lista de Administradores

Con la selección de una asignatura con el botón derecho del ratón, se nos abrirá un submenú en el cual se dispondrá de:

1. **View:** Para visualizar la información de la asignatura, en ella se muestran los usuarios registrados. Por una banda los profesores y por otra los alumnos.
2. **Edit Class:** Se nos muestra un dialogo con una lista de los usuarios registrados en esta. En éste podremos realizar las acciones de **New User,** así como hacer un clic encima de uno de ellos nos aparece un submenú con las opciones:
    1. **View:** Visualiza la información del usuario.
    2. **Edit User:** Se permite la edición de la información del usuario.
    3. **Remove User from Class:** Remueve el usuario de la asignatura.
    4. **Delete:** Elimina el usuario de la aplicación.
3. **Delete:** Elimina la asignatura de la aplicación.

Con la selección de un usuario administrador de la lista con el botón derecho del ratón, dispondremos de un submenú con las acciones:

1. **View:** Visualiza la información del usuario administrador.
2. **Edit User:** Se permite la edición de la información del usuario.
3. **Delete:** Elimina el usuario de la aplicación.

10.3 Manual Alumno

A continuación encontrara el manual de alumno que le ayudara a entender la aplicación y que operación realizar en cada momento.

Página Principal

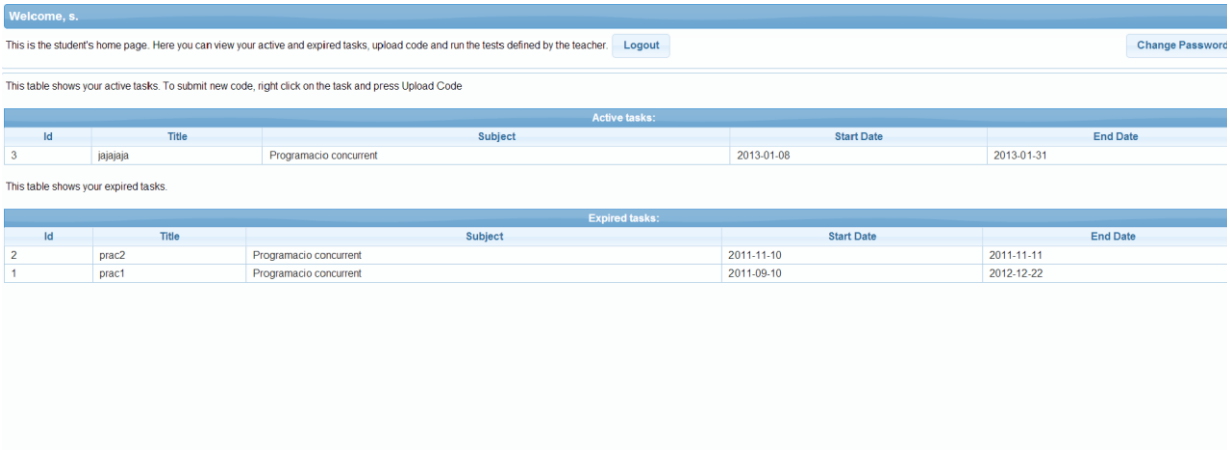


Ilustración 3: Página Principal del Alumno

En la página principal se encuentran dos listas:

1. **Active Task:** Lista de tareas activas.
2. **Expired Task:** Lista de tareas expiradas, las cuales ja no se puede realizar la entrega.

Si se selecciona una tarea de la lista A**ctive Task** se dispone de las opciones:

1. **View:** Visualiza la información de la tarea.
2. **Upload Code:** Acción que nos llevara a una nueva página en la cual se realizara la entrega para la tarea, que se puede ver en la *Ilustración 4*.

Página de Entrega

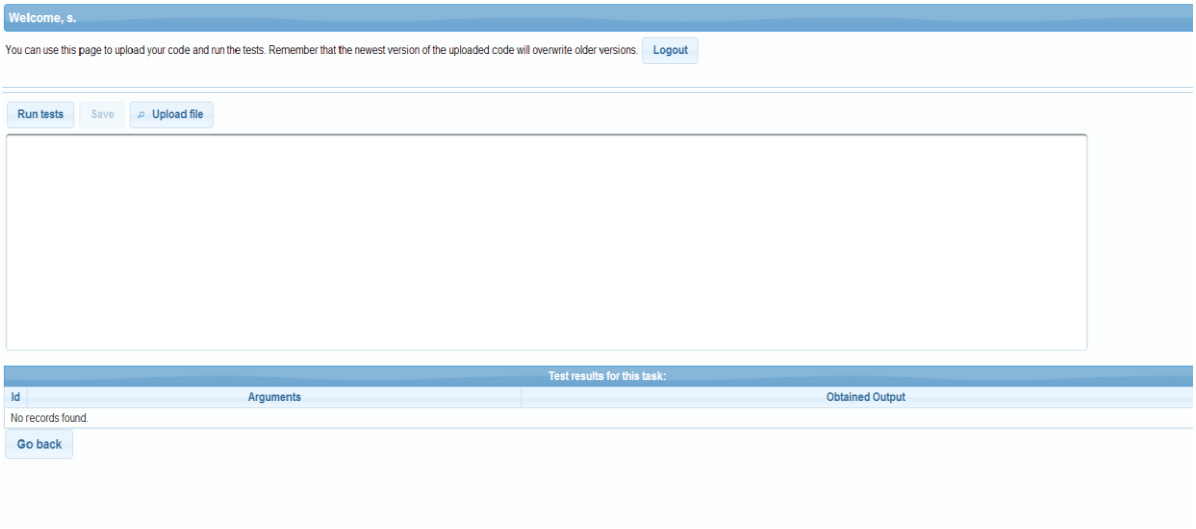


Ilustración 4: Página de Upload Code

Acciones posibles:

- **Run Test:** Ejecuta el código subido con los juegos de prueba que tiene definio la tarea.
- **Save:** Realiza el guardado de la entrega. Esta no estara habilitado hasta que no se haya ejecutado los juegos de prueba.
- **Upload File:** Opción de subida del fichero con el código. El contenido de este se podrá visualizar en el cuadro de texto, donde se podrá modificar.

Debajo del cuadro de texto se mostraran en una tabla los resultados de los juegos de prueba.

10.4 Manual Profesor

A continuación encontrara el manual de profesor que le ayudara a entender la aplicación y que operación realizar en cada momento.

Página Principal

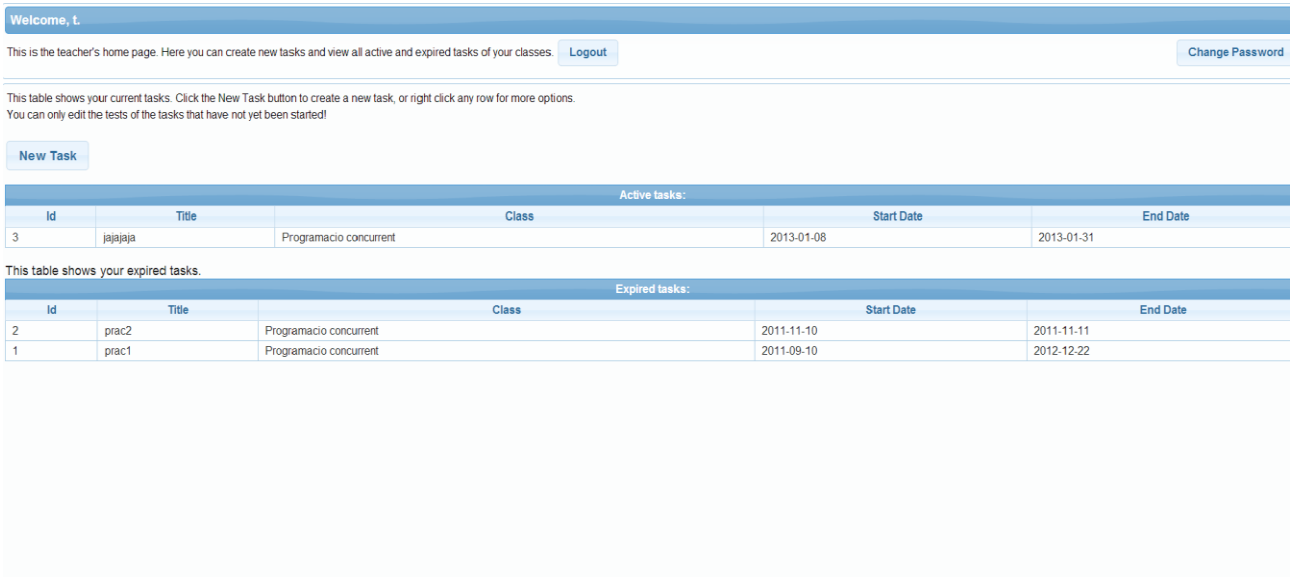


Ilustración 5: Página Principal del Profesor

Acciones Posibles

- **New Task:** En esta acción realizaremos la creación de una nueva tarea. Ésta nos enviara a una nueva página que se explicara más adelante, *Ilustraciones 6,7,8*.

Además en esta página se mostraran dos listas:

1. **Active Task:** Lista de tareas activas.
2. **Expired Task:** Lista de tareas expiradas, las cuales ja no se puede realizar la entrega.

Si se selecciona una tarea de la lista **Active Task** se nos mostrará el siguiente submenú:

1. **View:** Visualiza la información de la tarea.
2. **View Report:** Está acción nos enviara a la página para ver los resultados de las entregas realizadas por los usuarios. Esta página se mostrara detalladamente más adelante.
3. **Edit Test:** Acción que nos permite el añadir y visualizar los juegos de prueba.

4. **Delete:** Eliminar la tarea.

Por contra en las tareas de la lista **Expired Task** dispondremos de las opciones del menú **1, 2, 4.**

Página New Task

Está página está compuesta por cuatro tabuladores consecutivos. Es necesario rellenar los datos obligatorios para poder ir al siguiente.

- **General:** Este tabulador pide introducir los datos generales de la tarea, *Ilustración 6*:
  - **Title**: Titulo de la tarea
  - **Class:** Asignatura a la que pertenece la tarea.
  - **Start Date & End Date:** Marca el periodo de la tarea.
  - **Description:** Descripción extensa de la tarea.

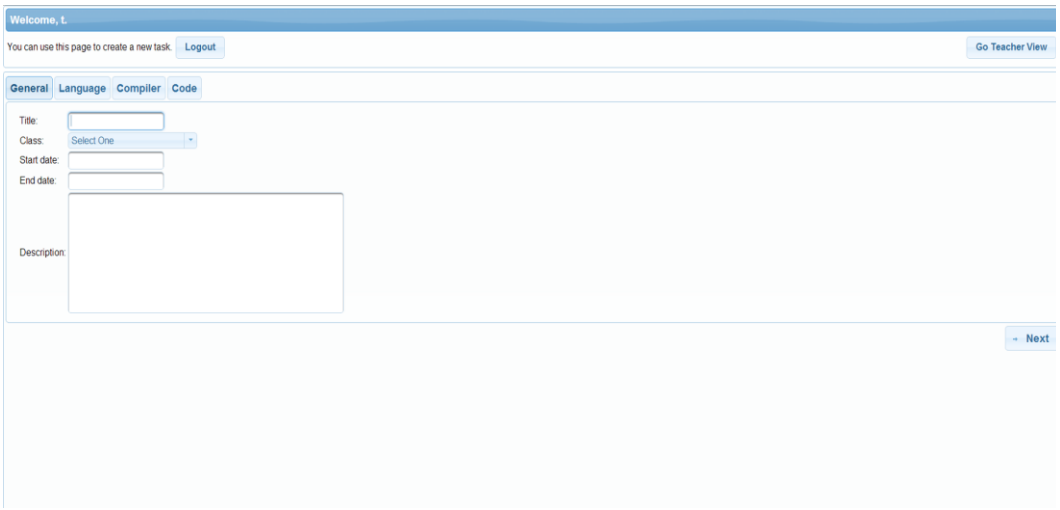


Ilustración 6: Tabulador General

- **Language:** Éste se ocupara del idioma, *Ilustración 7*:
  - **Language:** Desplegable con los lengaujes de programación disponibles. Anotacion: De momento tan solo esta funcional el lenguaje C.

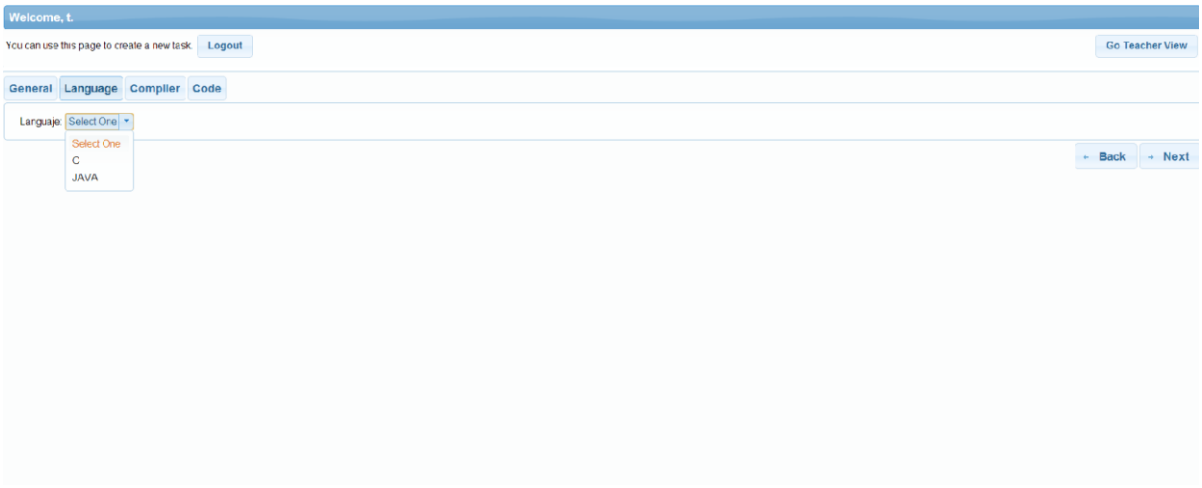


Ilustración 7: Tabulador Language

- **Compiler:** Se ocupará de las opciones de compilación de los programas, *Ilustración 8*.
  - **Upload Code:** Checkbox que nos permite seleccionar si se desea subir código o no.
  - **Compiler Arguments:** Argumentos de compilación.

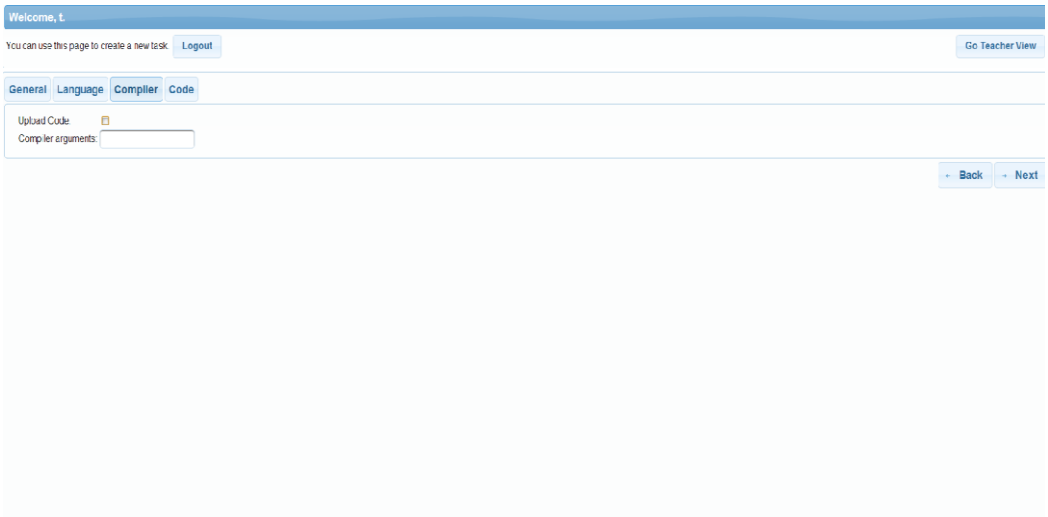


Ilustración 8: Tabulador Compiler

- **Code:** Según si se ha escogido subir código o no se mostrara un estilo u otro.

Si se ha escogido la opción de no subir se mostrara, *Ilustración 9*:

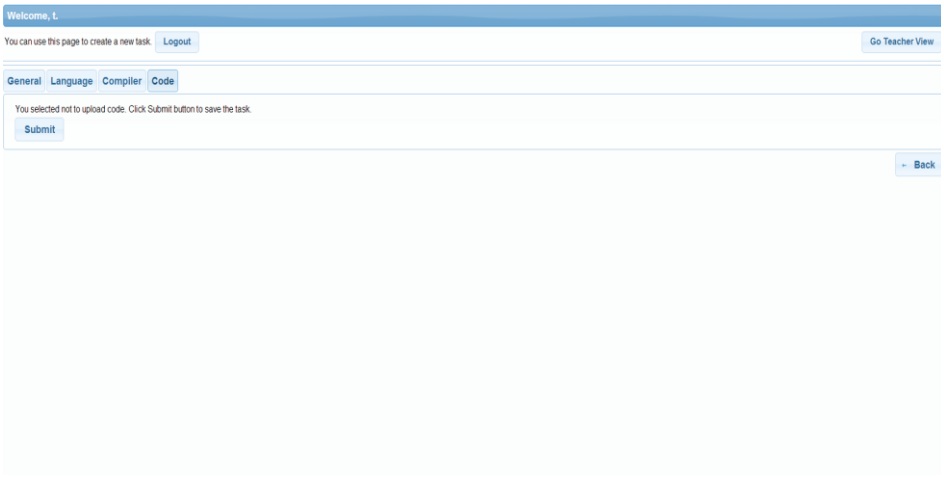


Ilustración 9: Tabulador Code sin subida de código

- **Submit:** Realizar la creación de la tarea.

Si por el contrario se escogió la opción de subir código se mostrara un cuadro de texto con las siguientes acciones, *Ilustración 10*:

- **Check Code:** Para realizar una comprobación del código.
- **Submit:** Realizar la creación de la tarea. Éste no se habilitara si no se dispone de un código correcto.

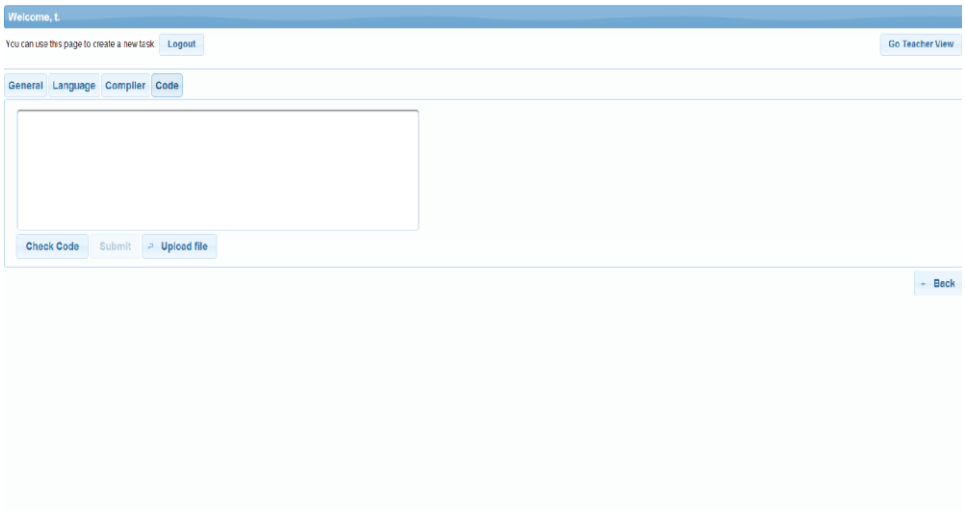


Ilustración 10: Tabulador Code con subida de código

- **Upload File:** Opción para la subida del código mediante un fichero.

Página Edit Test

En esta página se realizara la opción de añadir nuevos test. Ésta está compuesta por un desplegable donde dependiendo la opción escogida se mostrara un dialogo u otro. Es en este dialogo se deberán subir los ficheros o escribir los argumentos de entrada. Todos estos diálogos tendrán dos opciones:

- **Save:** Se almacenan los argumentos
- **Cancel:** Se descartan los argumentos

Una vez almacenados los argumentos, se dispone de las opciones siguientes:

- **Save:** Se almacena el test**.**
- **Run Test:** Este campo tan solo estará activo si el profesor a subido el código. Éste nos permite realizar la ejecución del código con los argumentos introducidos.
- **Cancel:** Se descartan los argumentos y los resultados del test.

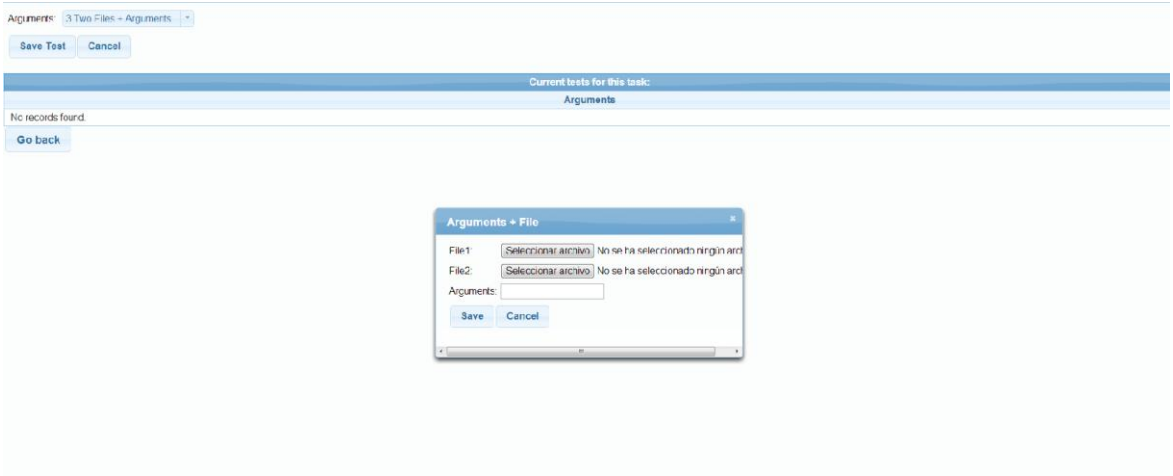


Ilustración 11: Página de Edit Test