Universitat de Barcelona

Treball de Fi de Grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques
Universitat de Barcelona

SISTEMA AUTOMATIZADO DE CORRECCIÓN DE PROGRAMAS: AMPLIACIÓN DE HERRAMIENTAS

Òscar Montañés Juanico

Director: Lluís Garrido Ostermann

Co-Director: Santi Seguí Mesquida

Realitzat a: Departament de Matemàtica Aplicada I Anàlisi. UB

Barcelona, 20 de juny de 2014

Resumen

El objetivo del proyecto es la implementación de una aplicación web con el nombre de Sistema Corrector de Programas, capaz de corregir código de programas de forma automática. El escenario donde se sitúa esta aplicación es en la comunicación entre profesores y alumnos a la hora de crear tareas de programación en alguna asignatura concreta.

Los profesores adjudicados a una asignatura podrán configurar diferentes tareas y tests de evaluación que los alumnos deberán de superar subministrando su código. Una vez la tarea se haya finalizado, los profesores verán los resultados de los tests de los alumnos para corregir con comodidad, o también podrán descargar el código si así lo desean. Los alumnos, por su lado, podrán entregar su código y ver a tiempo real si este cumple los tests configurados por el profesor. De esta manera se busca incentivar el aprendizaje autónomo y la motivación.

Este trabajo final de grado es la continuación de cuatro otros proyectos, todos realizados con la finalidad de construir una plataforma segura y cómoda de usar que en un futuro pueda ser implementada en el Grado de Ingeniería Informática de la UB y usada por todos los alumnos y profesores.

El presente proyecto se centra en resolver problemas aportados de la mano de profesores los cuales han probado la aplicación y han aportado un listado con mejoras y problemas que había previamente.

En este documento encontraremos información sobre el análisis de la versión anterior y el procedimiento que se ha seguido para implementar las nuevas herramientas y mejorar las existentes.

Resum

L'objectiu del projecte és la implementació d'una aplicació web amb el nom de Sistema Corrector de Programes, capaç de corregir codi de programes de forma automàtica. L'escenari on es situa aquesta aplicació és en la comunicació entre professors i alumnes a l'hora de crear tasques de programació en alguna assignatura concreta.

Els professors adjudicats a una assignatura podran configurar diferents tasques i tests d'avaluació que els alumnes hauran de superar subministrant el seu codi. Un cop la tasca es finalitza, els professors veuran els resultats dels tests dels alumnes per a corregir amb comoditat, o també podran descarregar el codi si ho desitgen. El alumnes, per la seva banda, podran entregar el seu codi i veure a temps real si aquest compleix els tests configurats pel professor. D'aquesta manera es busca incentivar l'aprenentatge autònom i la motivació.

Aquest treball final de grau és la continuació de quatre altres projectes, tots realitzats amb la finalitat de construir una plataforma segura i còmode d'utilitzar que en un futur pugui ser implementada en el Grau d'Enginyeria Informàtica de la UB i utilitzada per tots els alumnes i professors.

El present projecte es centra en resoldre problemes aportats de la mà de professors els quals han provat l'aplicació i han aportat un llistat amb millores i problemes que hi havia prèviament.

En el document trobarem informació sobre l'anàlisi de la versió anterior i el procediment que s'ha seguit per a implementar les noves eines i millorar les existents.

Abstract

The goal of this project is the development of a website named Program Corrector System. This website must be able to automatically validate the output of the software programs. The application scenario where this is located is the communication between professors and students in creating programming tasks of some particular subjects.

Professors from different subjects define different evaluation rules and tests. The software delivered by the students must pass these tests. Once the deadline from a test is over, the professors of the corresponding subject task can see the results from all students, and download all codes. Students, in the other hand, can submit their own code and see in real time if it accomplishes all tests defined by the professor. In this way, we're looking forward to encourage autonomous learning and motivation.

This final degree thesis is the continuation of four previous projects, all of them working together to build a safe and friendly platform. The aim of this platform is to be implemented in different subjects from University of Barcelona in the future.

My role in this project consists in resolving and improving several issues provided by professors who tested the last versión and made a list of them that needs to be improved.

In this document, we'll find information about the review of the last versión and the procedural method used to improve the existing features and implement new ones.

Índice de contenido

1. Introducción

El Sistema Corrector de Programas (SCP) es una herramienta orientada para los primeros cursos de programación, en los cuales los alumnos deben hacer un número elevado de entregas de pequeños programas mientras se van familiarizando con el mundo de la programación.

Es por eso, que una herramienta que facilite a los profesores la corrección de entregas, así como a los alumnos la comprobación de los resultados, se puede considerar indispensable en una facultad de estudios informáticos.

A través de esta herramienta, un alumno puede hacer entregas de su código, y a la vez comprobar si éste cumple todos los tests que el profesor había designado para esa tarea. A la vez, el profesor puede comprobar el código de todos los alumnos que han hecho una entrega y revisar el porcentaje de tests superados que el profesor había configurado.

1.1 Antecedentes

Este proyecto es la continuación de otros cuatro Trabajos Finales de Gradp, todos ellos dirigidos por Lluís Garrido.

El proyecto original, fue realizado por Jordi Salvatella el primer semestre de 2011. El SCP se trataba de una web realizada con Java Servlets en el que un usuario podía subir un código comprimido el cual el sistema descomprime, compila, ejecuta y escribe el resultado de compilación y sus ejecuciones en un fichero. Más tarde SCP lee este fichero y lo muestra en la página web.

Este proyecto fue continuado por Daniel Gil, que, partiendo del anterior, desarrolló un programa que intenta conseguir una mayor perspectiva empresarial usando un código más robusto y con mayor facilidad para su extensión. Para ello se introdujeron herramientas y frameworks con una amplia documentación, como por ejemplo MySQL, Hibernate, Primefaces y otros.

Más adelante vino la continuación de la mano de Jordi Serral, que analizó el proyecto en busca de mejoras en el apartado de la interfície web, para que se pudiera trabajar cómodamente con un número elevado de usuarios. En general, se adaptó la web haciendo SCP una aplicación más cómoda y fácil de usar para los usuarios finales, los cuales pueden tomar los roles de administrador, profesor y alumno.

Finalmente, vino el proyecto de Enrique Muñóz, en el que se centraba en mejorar la parte que automatiza el proceso de compilación y ejecución de los programas. Se creó un nuevo programa llamado SecuritySCP que se comunica vía RMI con la aplicación web y agrupa todas esas funcionalidades encargadas de compilar y ejecutar códigos además de ofrecer un sistema de resultados que se envían a la aplicación web para que se muestren. En este punto, el nombre del proyecto era SCP 3.0.

Este proyecto empieza el SCP 4.0 añadiendo muchas más opciones en la definición de tareas, solucionando bugs y aumentando la seguridad en la ejecución de programas. Durante este documento veremos todos los cambios realizados con más detalle.

1.2 Objetivos

El objetivo principal de todo este proyecto es el de disponer de una herramienta fácil de usar y cómoda, tanto para profesores como para alumnos, el cual cumpla todos los requisitos necesarios para que algún día se pueda instalar en el Grado de Ingeniería Informática de la UB y para que se use con normalidad durante toda la carrera. De este modo se facilitaría y motivaría el aprendizaje autónomo, dando al alumno la posibilidad de comprobar si su código cumple los tests de la entrega programados por el profesor, que a su vez, también se aprovecha de la aplicación ahorrándole tiempo para la corrección de sus practicas.

Es por eso que llegados a este punto, el presente proyecto se centra en arreglar inexactitudes de la interfaz y añadir más opciones para la creación de tareas y entregas para poder ser más personalizables por parte del profesor. Además, se han revisado gestiones de seguridad para evitar el uso indebido de la aplicación y se ha añadido políticas de uso en las aplicaciones java.

1.3 Planificación

Este proyecto se ha realizado en el segundo semestre del curso 2013-2014 teniendo una duración aproximada de 6 meses empezando en enero y acabando en junio.

A continuación se muestra el diagrama de Gantt del procedimiento efectuado.

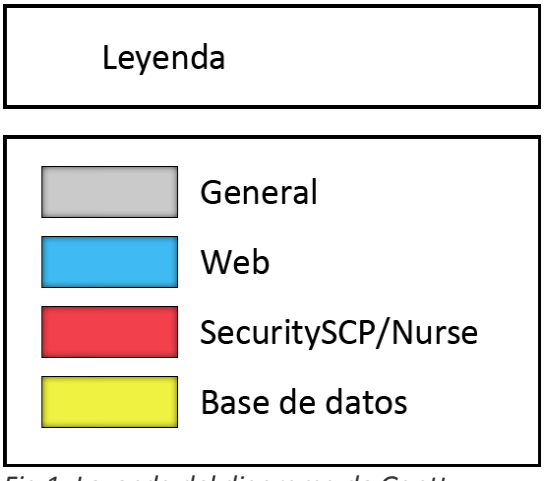

Fig 1: Leyenda del diagrama de Gantt

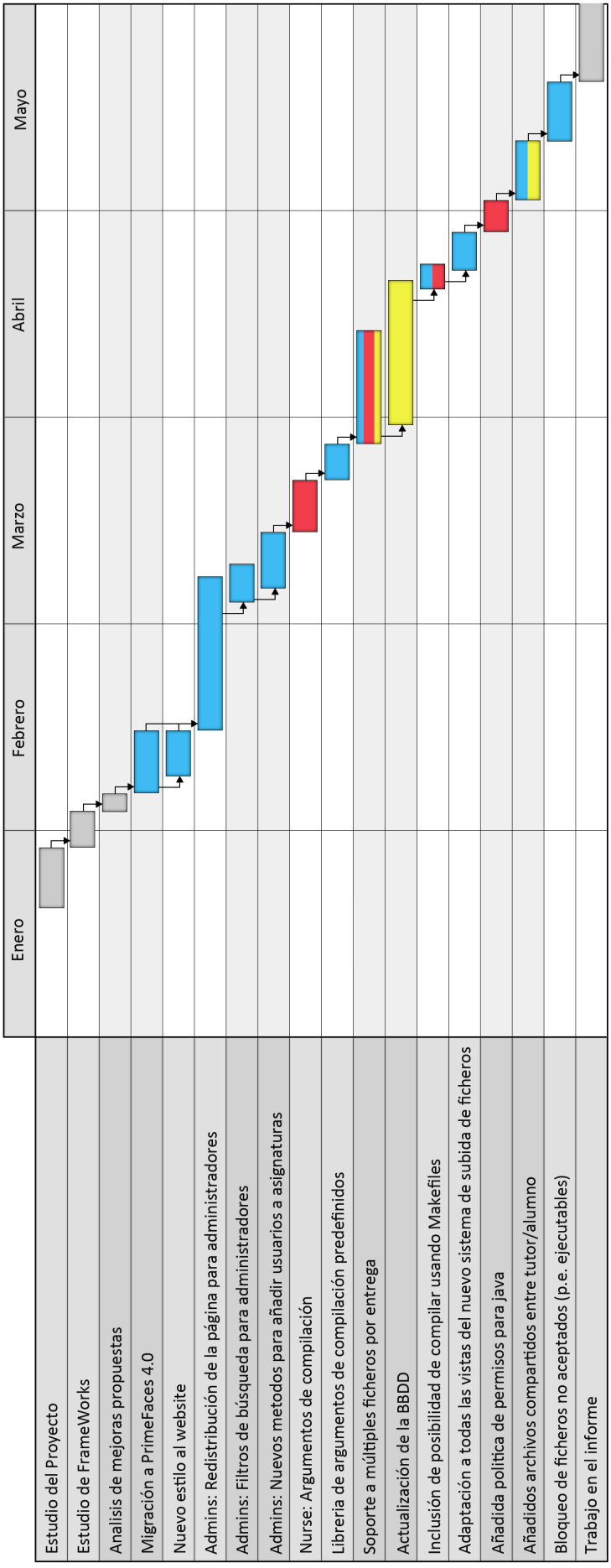

Fig 2: Diagrama de Gantt con el proceso seguido

2. Análisis del TFG

2.1 Tecnologías usadas

Las tecnologías utilizadas en este proyecto son las mismas que en las versiones anteriores. Ha continuación repasaremos y explicaremos brevemente cada una de ellas.

· **Java** es el lenguaje principal que se ha usado en este proyecto. Toda la aplicación web y SecuritySCP están escritos en Java.

· **Apache Tomcat** es el servidor donde la aplicación web se sitúa.

· **Java Server Faces (JSF)** es la tecnología usada para escribir el código de la página web. JSF acepta extensiones o frameworks adicionales para facilitar su uso. Una de estas extensiones usadas para la interfaz gráfica ha sido **PrimeFaces.**

· **Spring** es un framework de inyección de dependencias con un amplio uso en el mundo empresarial por los desarrolladores de Java. Se utiliza para mantener el código limpio y de fácil mantenimiento. Nos permite declarar objetos sin necesidad de instanciarlos dentro del código. Además, también se usa **Spring Security 3**, que es un framework derivado de Sprint y se encarga de la seguridad en la identificación y autorización de acceso a las diferentes páginas, sesiones, roles, etc.

· **RMI** es la tecnología usada para comunicar dos nodos principales de nuestro proyecto: La aplicación web, y el sistema de compilación y devolución de resultados SecuritySCP. Usando RMI es posible que estos dos nodos puedan distribuirse en maquinas distintas.

· **MySQL**. La base de datos usada ha sido MySQL, y para facilitar la comunicación con ella usamos el framework **Hibernate** para crear una abstracción del nivel SQL. Usando ficheros de configuración y unas clases de Java, esta librería nos permitirá, llamar a funciones de objetos sin hacer directamente la consulta SQL en la base de datos.

Fig 3: Tecnologías usadas

2.2 Bases de datos

2.2.1: Análisis de los trabajos anteriores

El proyecto usa una base de datos MySQL donde se guarda toda la información de usuarios, tareas y entregas. Únicamente tiene acceso a ella el código de la aplicación web.

El modelo ER de la versión SCP3.0 del proyecto era el siguiente:

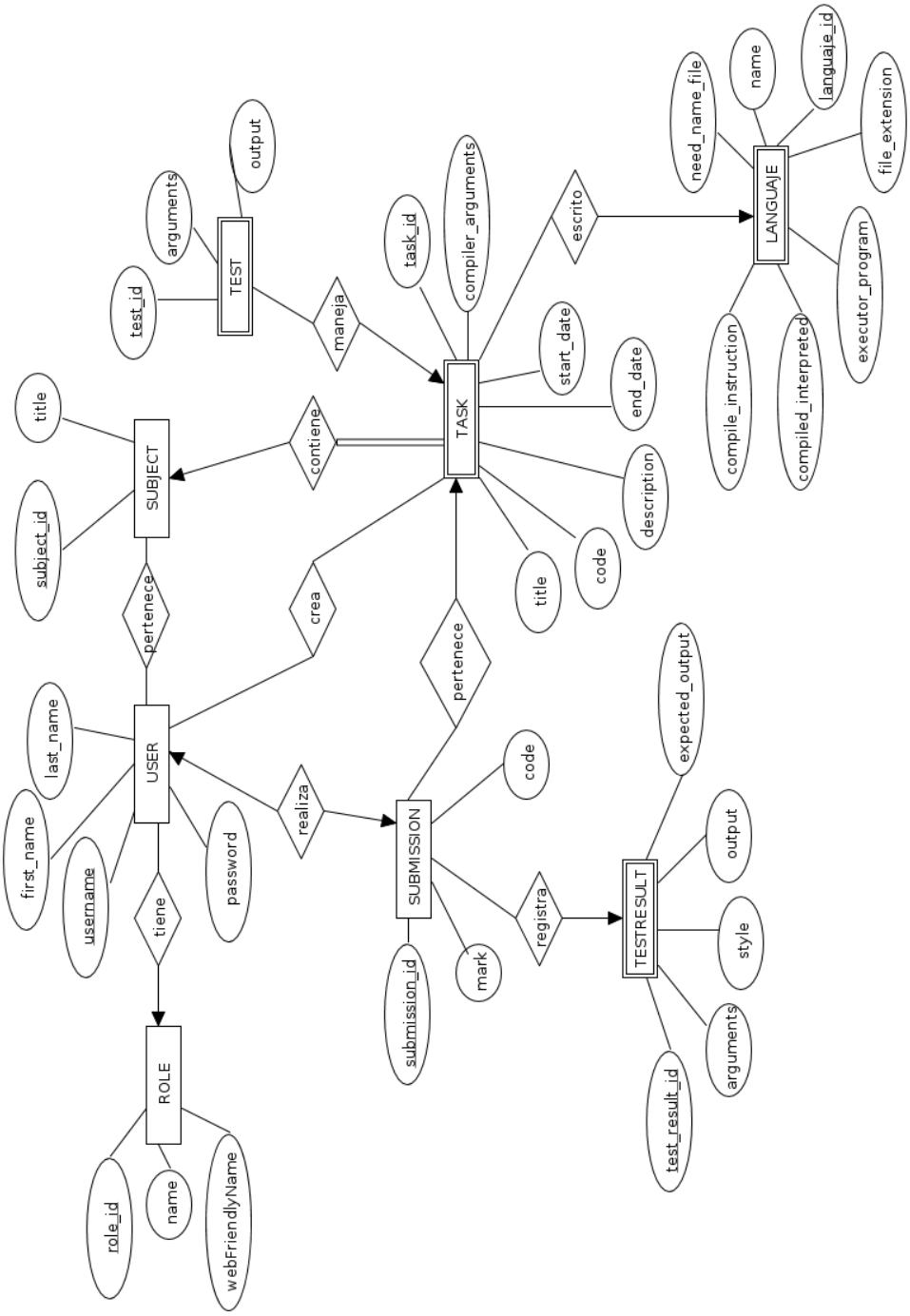

Fig 4: Base de datos en la versión SCP3.0 del proyecto

Análisis de entidades

ROLE: Entidad que contiene la información necesaria de los posibles roles de un usuario: Administrador, profesor o alumno.

- role_id: Campo que identifica inequívocamente una entrada en la entidad Role.

  1 - Profesor 2 - Alumno 3 - Administrador.

- name: Nombre del Role.
- webFriendlyName: Nombre del Role con un formato más familiar para un usuario corriente. Este es el nombre que se mostrará, por ejemplo, en la web.

USER: Es la persona que utilizará la aplicación, ya sea como administrador, profesor o alumno.

- username: Campo identificador de la tabla. Este campo se usará para el login.
- password: Contraseña del usuario. Campo obligatorio para el login del usuario en la aplicación.
- role: Campo relacional que define el role del usuario.
- first_name: Nombre del usuario.
- last_name: Apellidos 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.
- title: Campo que nos indica el titulo de la asignatura.

LANGUAGE: Entidad que contiene la información necesaria de un lenguaje de programación.

- language_id: Campo que identifica una entrada en la entidad Language.
- name: Nombre del Language.
- file_extension: Campo que contiene la extensión de los ficheros para ese lenguaje .
- compile_instruction: Instrucción para compilar código en ese lenguaje.
- compiled_interpreted: Booleano que indica la necesidad de que el código sea compilado. (En python este valor es false).
- executor_program: Instrucción para ejecutar el programa una vez compilado. (Vacío en C)
- need_name_file: Booleano que indica la necesidad de un lenguaje de requerir información sobre el nombre del fichero compilado. (En java este valor es true).

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

- task_id: Campo que identifica una entrada en la entidad Task.
- title: Nombre de la tarea.
- start_date: Campo que representa la fecha de inicio de la Tarea, fecha a partir de la cual se pueden realizar las entregas.
- end_date: Campo que representa la fecha de fin de la Tarea, fecha a partir de la cual ya no se pueden efectuar más entregas.
- description: Campo que almacena un texto descriptivo de la Tarea.
- owner: Campo que relaciona la Tarea con el profesor propietario de ella.
- code: Código subido por el creador de la Tarea.
- compiler_arguments: Campo con los argumentos para compilar correctamente el código de esta tarea.
- subject_id: Campo que relaciona a que asignatura pertenece la tarea.

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.
- task_id: Campo que relaciona los tests creados con la tarea a la cual pertenecen.
- arguments: Argumentos de entrada de ejecución para efectuar los tests.
- output: Salida obtenida al ejecutar el código con el test.

SUBMISSION: Representa una entrega de código del alumno a una tarea del profesor.

- submission_id: Campo que identifica una entrada en la entidad Submission.
- task_id: Campo que relaciona una entrega con la task.
- mark: Representa el porcentaje de tests correctos de esa entrega.
- user: Campo que relaciona una entrega con el usuario que la realiza.
- code: Código subido por el usuario que realiza la entrega.

TESTRESULT: Son los resultados de los test que los alumnos obtienen tras hacer una entrega (submission). En ellos se especifica el resultado de cada uno de los test realizados.

- test_result_id: Campo que identifica una entrada en la entidad Testresult.
- submission_id: Campo que relaciona los testsresult creados con la entrega a la cual pertenecen.
- arguments: Argumentos de entrada para los tests.
- output: Salida al ejecutar el código con el test.
- excepted_output: Salida esperada para que el test hubiera sido correcto.
- style: Campo que indica si el resultado del test ha sido correcto o incorrecto.

2.2.2 Cambios añadidos en el proyecto

Aunque esta base de datos es totalmente funcional para la versión anterior del proyecto, no lo será para esta. La base de datos se ha modificado, y se han solucionado algunos puntos que veremos a continuación:

1. El código de los programas se guarda en la base de datos. A priori esta opción puede parecer buena, pero hemos de tener en cuenta que un código puede ser muy extenso, y no es el propósito de una base de datos el guardar mucha información de este estilo.

   Por lo que en este proyecto, se ha eliminado el campo “codigo” de la base de datos, y ahora los ficheros se guardaran dentro del disco duro. A través del nombre de la tarea y del usuario, podremos acceder a ellos con facilidad.

2. Hasta el momento, la base de datos no ha estado pensada para subida de múltiples ficheros, por lo que solo se permite un código por entrega.

   Para ello se ha creado una nueva entidad a la base de datos en la que se registran los ficheros que contiene tanto una entrega como una tarea.

3. Una vez realizada adaptada la base de datos para múltiples ficheros, detectamos que únicamente se permite realizar una tarea o entrega usando código. Es decir, no podemos subir una imagen o vídeo, lo cual puede ser útil si el programa necesita leer datos de ellos.

   Como el contenido de los ficheros ya no se encuentran en la base de datos sino en nuestro disco, no tenemos la limitación de no poder guardar binarios, sin embargo, hemos creado un filtro para que no todos los ficheros sean aceptados: no nos interesa que se puedan subir ejecutables maliciosos en nuestro servidor.

Para realizar todos estos cambios, ha hecho falta añadir nuevas entidades y modificar algunas existentes que a continuación explicaremos con detalle.

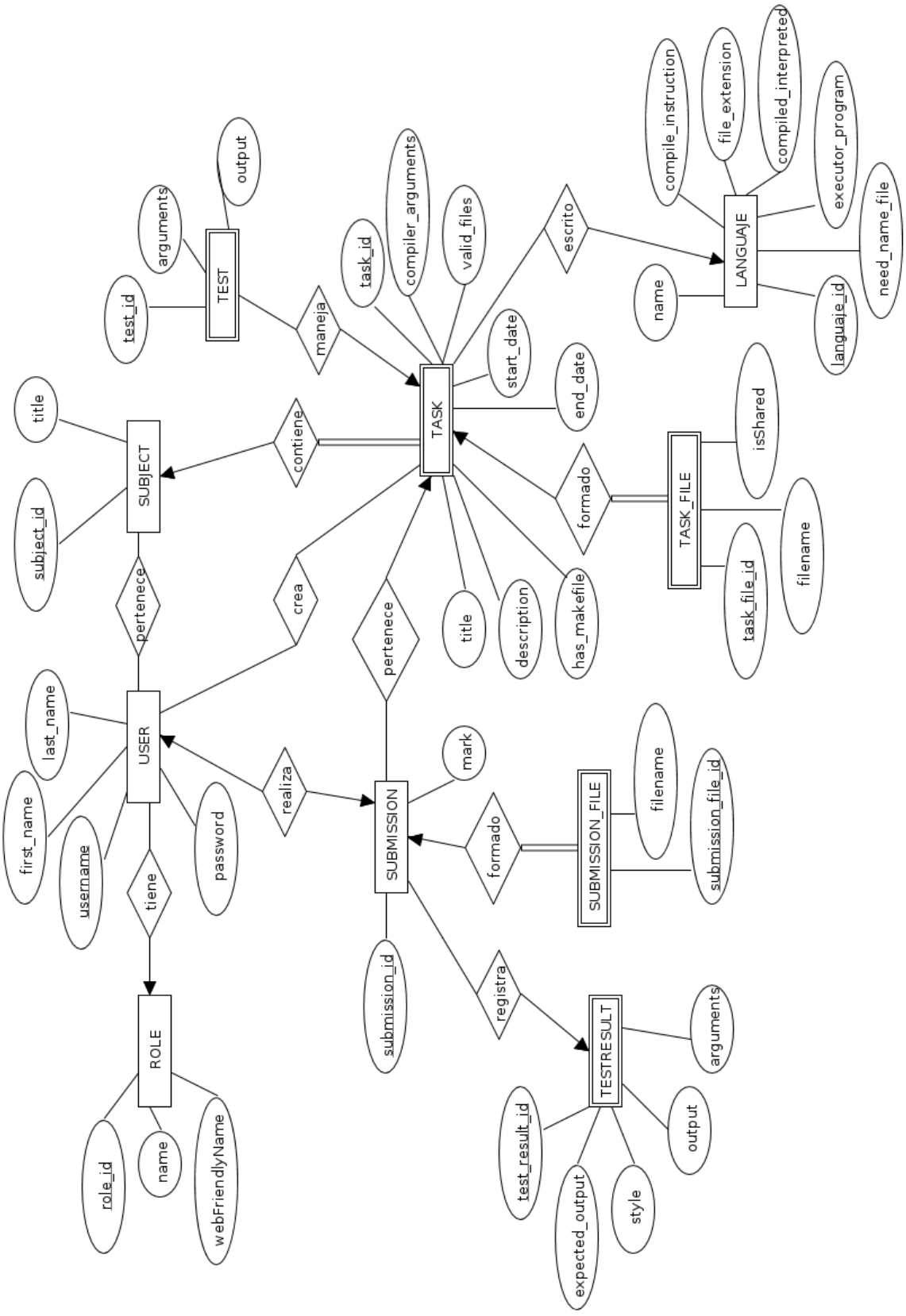

Fig 5: Nueva base de datos implementada en este proyecto del SCP4.0

TASK: Cambios efectuados en la entidad de tareas.

- ✗ Se ha eliminado el atributo **code**.
- • Modificación: El campo **description** puede ser NULL y estar vacío ahora.
- + has_makefile: Campo que nos indica si el código dispone de archivo un makefile para compilar.
- + valid_files: Campo que indica que archivos aceptaremos. Se trata de un entero en el que, cada bit corresponde a un tipo de fichero. Si ese bit es 1, aceptaremos ese tipo de archivo y si es 0 no.

  Los bits que se relacionan con los tipos de ficheros son los siguientes (*ordenados de bit de menor orden a bit de mayor orden)*:

  1ʳ bit: PDF's
  2º bit: Imágenes
  3ʳ bit: Audios
  4º bit: Vídeos

  En todos los casos los ejecutables estarán prohibidos y los ficheros de texto permitidos. Por ejemplo, con el entero 5 (0101 en binario) aceptaríamos ficheros PDF's y audios.

TASK_FILE: Entidad añadida. Dentro de esta entidad registraremos todos los ficheros que los profesores han añadido para comparar los tests de una tarea.

- + task_file_id: Campo que identifica una entrada en la entidad TaskFile.
- + filename: Campo que indica el nombre del fichero.
- + shared: Campo booleano que indica si este fichero será compartido con las entregas de los alumnos. Es decir, todos los ficheros que tengan activo este atributo, se copiaran junto a los ficheros que los alumnos presenten. Los archivos makefile siempre deben tener shared activo. Los alumnos nunca elijen cómo compilar el código.

SUBMISSION: Cambios efectuados en la entidad de entregas.

- ✗ Se ha eliminado el atributo **code**.

SUBMISSION_FILE: Entidad añadida. Dentro de esta entidad registraremos todos los ficheros de las entregas de los alumnos.

- + submission_file_id: Campo que identifica una entrada en la entidad SubmissionFile.
- + filename: Campo que indica el nombre del fichero.

2.2 Web SCP4.0

2.3.1 Análisis de los trabajos anteriores

La web es un apartado importante en nuestro proyecto, será la vía por la que el usuario podrá usar nuestro sistema y se encargará de enviar las ordenes al SecuritySCP que veremos a continuación.

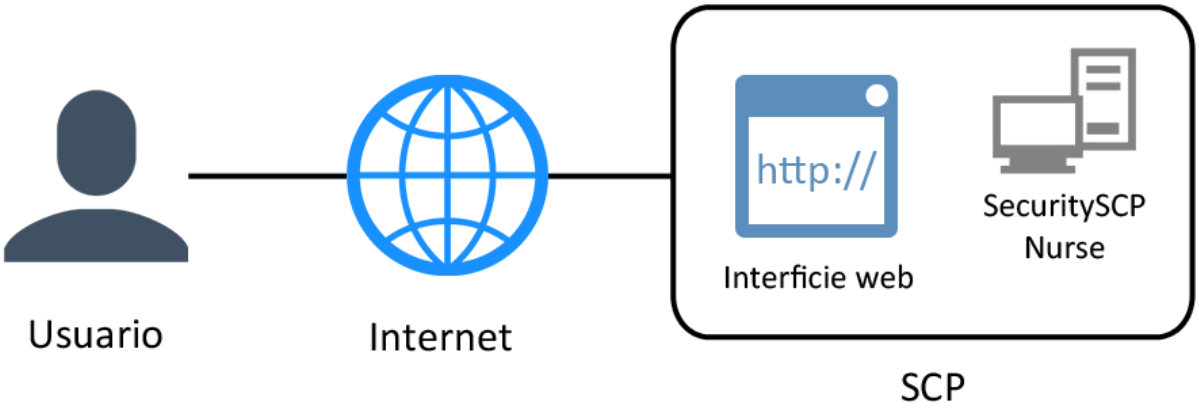

Fig 6: Esquema de nuestro sistema SCP

Login

La página web inicialmente te presenta una pantalla de login, en la cual debemos entrar un nombre de usuario y una contraseña. En ningún momento podremos acceder a una pagina de registro para obtener una cuenta, sino que, únicamente los administradores deben dar acceso a los alumnos y ofrecerles una.

Fig 7: Mensaje de Login al entrar en la web por primera vez

Inicialmente, el único usuario existente es un administrador con nombre admin y contraseña 1234.

Administrador

El administrador del sistema tiene una vista de la web especial para la creación de nuevos usuarios incluyendo a otros administradores, para la creación de asignaturas, y para hacer un limpiado de la base de datos eliminando todo excepto los administradores.

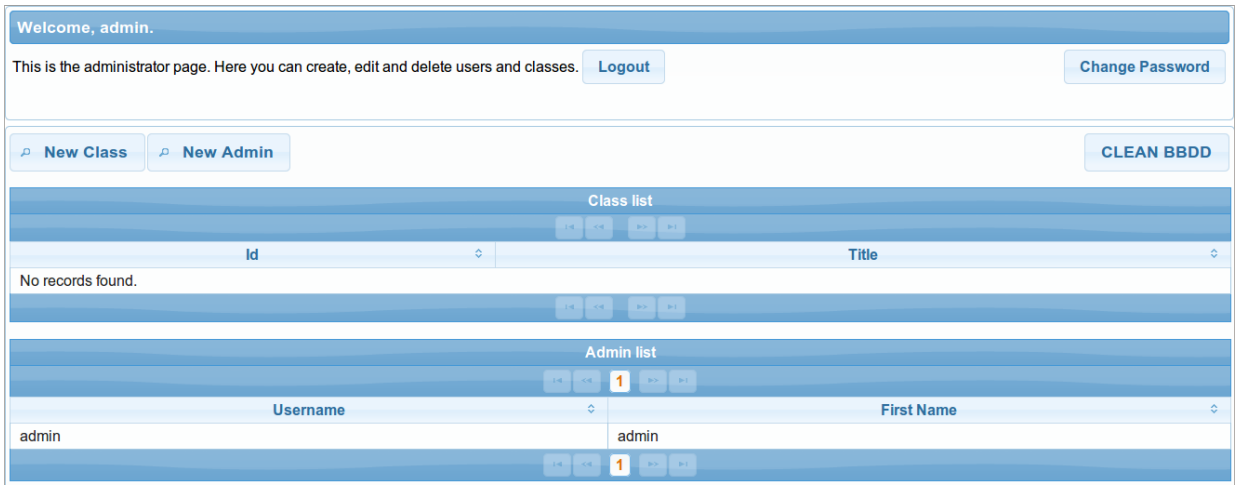

Fig 8: Vista principal de la web siendo administrador

Además, en el momento de añadir una asignatura podremos seleccionar un fichero extraído directamente del Campus Virtual de la UB en el cual se listan todos los alumnos de una asignatura para , de esta manera, poder añadirlos rápidamente. Además, si alguno de los alumnos no se encuentran en la base de datos, se añadirán.

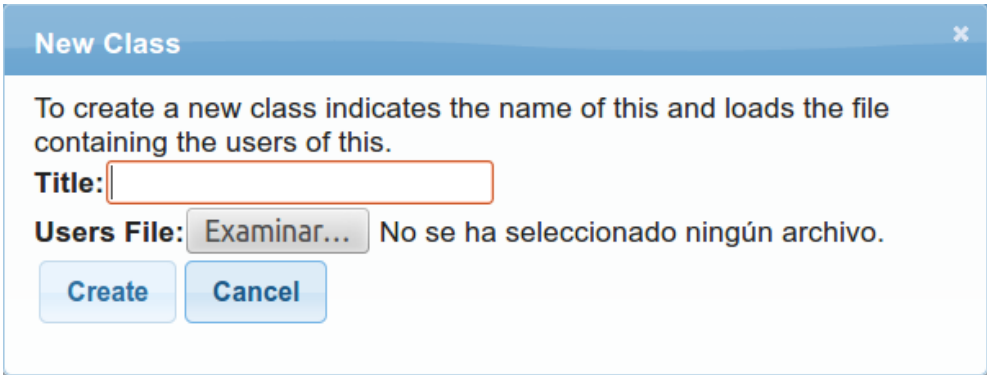

Fig 9: Ventana de creación de una nueva asignatura

De cualquier modo, siempre podremos añadir nuevos alumnos modificando la asignatura. De hecho, este será el único modo por el cual se podrán crear alumnos y no se podrán crear de otro modo sin que estén vinculados a ninguna asignatura.

Fig 10: Ventana de modificación de la asignatura

Observando la *Fig.10* podemos observar aquellos usuarios apuntados a una asignatura y añadir más si lo necesitamos. También podremos modificar a los usuarios o eliminarlos de la clase haciendo clic derecho sobre ellos.

Dejando de lado las asignaturas y alumnos, una de las tareas que puede hacer el administrador es la de crear otros administradores, de forma similar a como se crean los estudiantes o profesores.

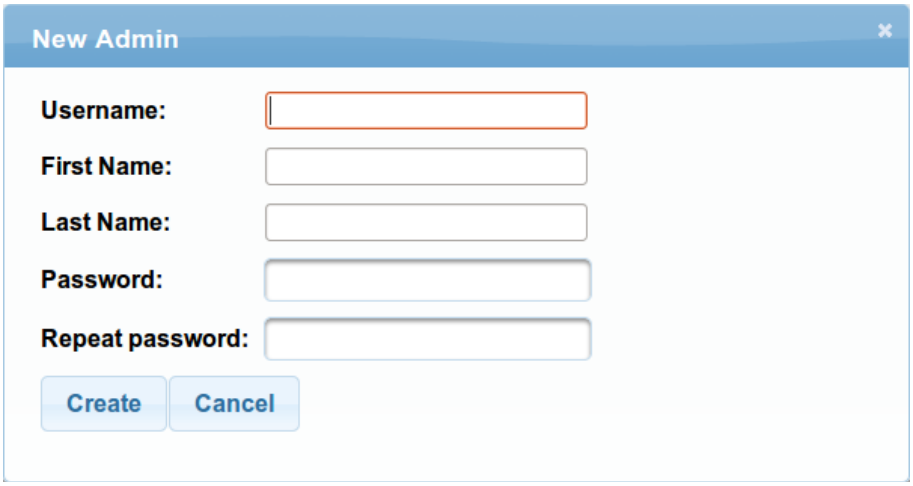

Fig 11: Ventana de creación de administradores

Finalmente, la última tarea que puede realizar un administrador es la de resetear la base de datos con sus valores por defecto (vacía, manteniendo únicamente los administradores).

Profesor

La vista del profesor será muy distinta a la del administrador, ya que en esta únicamente tendremos control sobre las tareas que hemos creado personalmente.

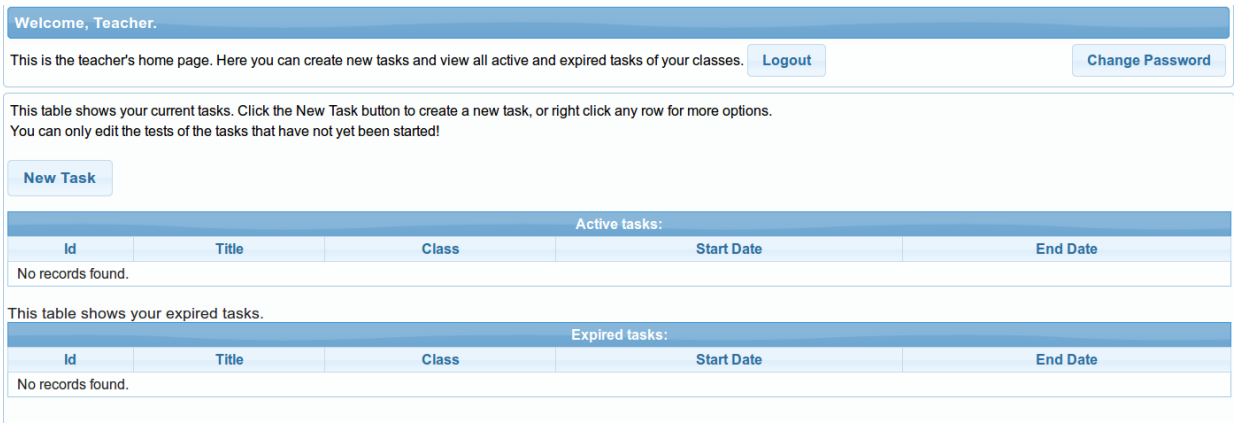

Fig 12: Vista principal de la web entrando como profesor

Al entrar como profesor nos aparecerá por pantalla un botón para crear una nueva tarea, y con dos listados para visualizarlas. La primera nos mostrará aquellas tareas que se encuentran activas, y en la segunda todas las tareas expiradas. En ambas podremos ver los resultados de entregas de los alumnos hasta el momento.

En el proceso de crear una nueva tarea, inicialmente se nos pedirá información básica de la asignatura así como el lenguaje en el que será escrito. De momento únicamente se soporta C, Java y Python y esto no ha cambiado durante el transcurso de este proyecto.

Las opciones de compilación son bastantes limitadas, y se reducen a decir que parámetros usaremos para compilar.

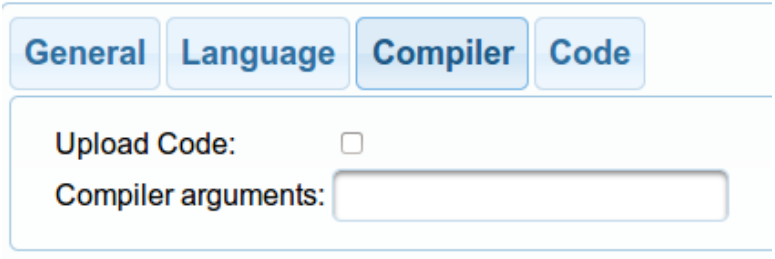

Fig 13: Datos pedidos para la compilación

Además, podemos decidir si queremos subir un código de referencia para comparar con el de los alumnos o no.

En caso de decidir subministrar un código, tenemos la opción de escribirlo o de subir un único fichero que contenga el código.

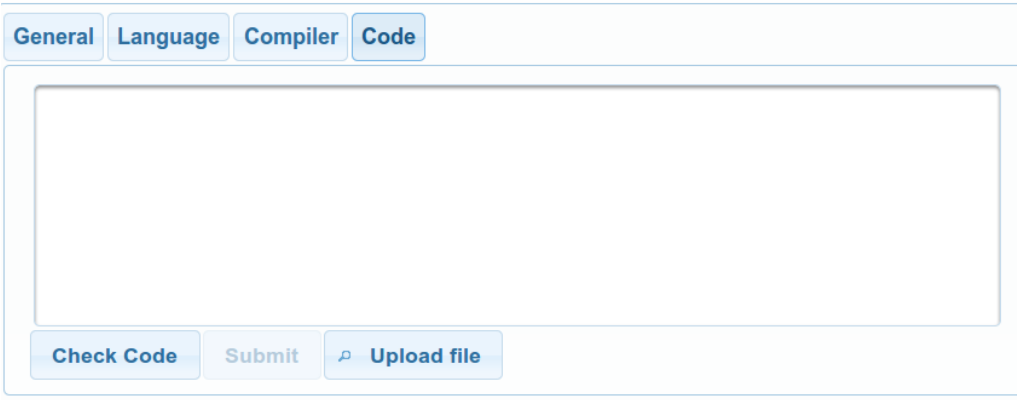

Fig 14: Última etapa de la creación de una tarea en la cual se nos pide el código a compilar

Una vez creada una tarea, se procederá a la creación de tests para comparar la salida de nuestro programa. En esta ventana podemos configurar distintos tests con diferentes argumentos de ejecución para comprobar la salida obtenida en cada uno de los casos.

Finalmente, la última función que puede realizar un profesor es la de revisar los resultados de los alumnos mediante la página de reportes. En ésta veremos el porcentaje de tests que cada alumno han pasado y su código si nos interesa verlo.

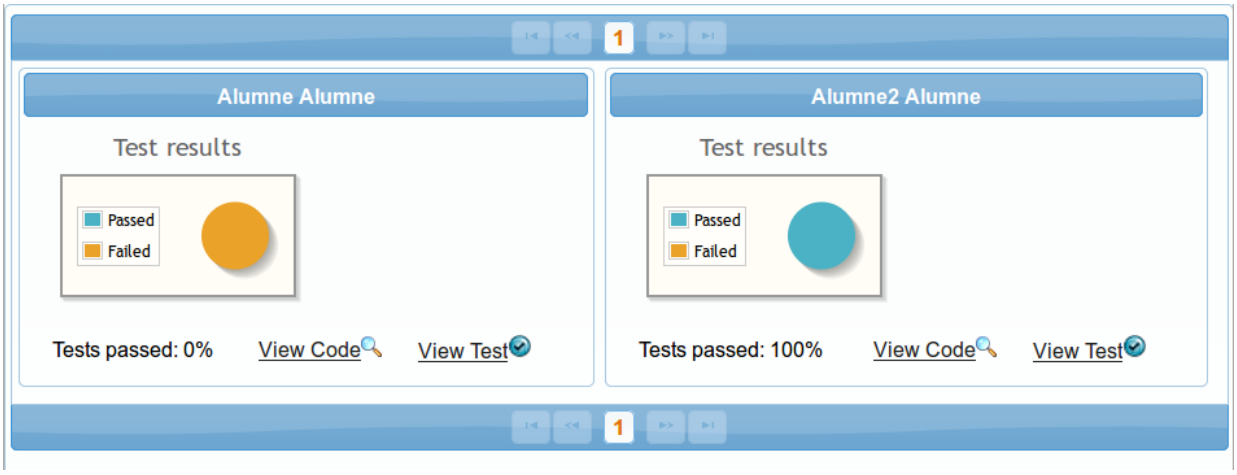

Fig 15: Vista de los resultados de las entregas de los alumnos

Alumno

A diferencia de los administradores o profesores, los alumnos no podrán crear asignaturas ni tareas, su función se limita a hacer entregas a las tareas que los profesores han creado.

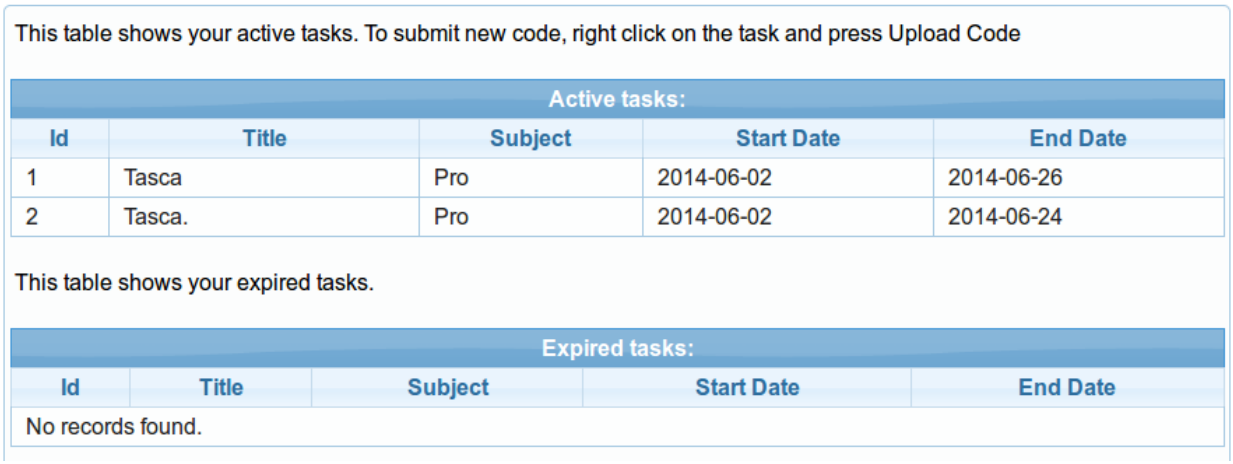

Fig 16: Vista principal de la web entrando como alumno

Su vista es similar a la del profesor, pudiendo ver las tareas activas y las expiradas. Sin embargo, el alumno sólo podrá subministrar su código de misma forma como lo hacia un profesor: mediante un fichero con código, o entrando el código manualmente en la web.

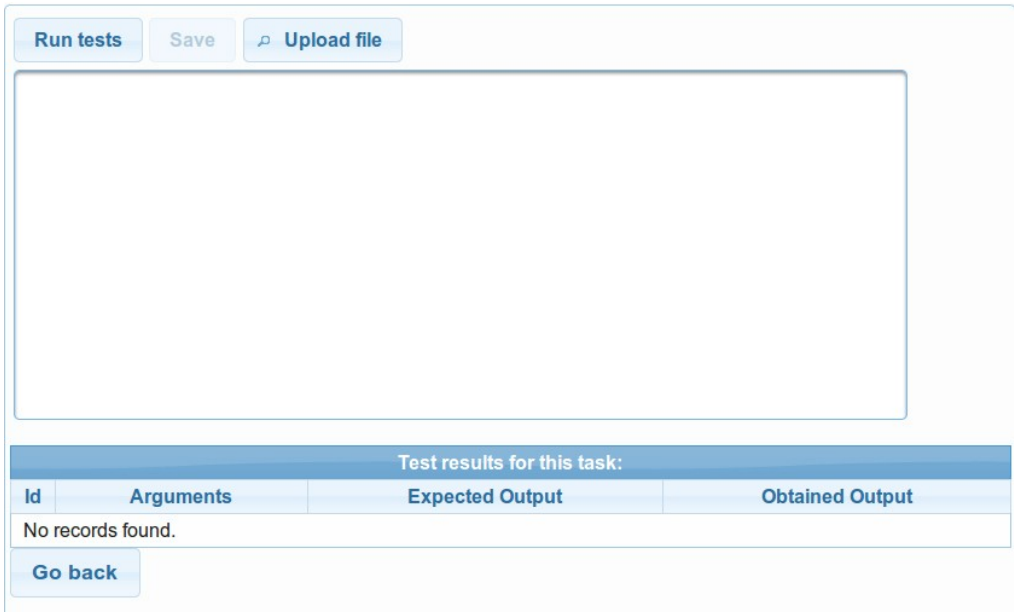

Fig 17: Diálogo de entrega del código por parte del alumno

2.3.2 Mejoras implementadas en el proyecto

Analizando la aplicación web hemos localizado ciertos puntos que no se actualizaban como debían al modificar datos, por ejemplo, ver el nuevo administrador después de crearlo. Otro aspecto que hemos podido observar es que se encuentra muy limitada en cuestiones de gestión.

En este proyecto se han arreglado esos pequeños fallos y se han aplicado medidas para facilitar el uso a los administradores y la creación de tareas a los profesores.

Además, se ha modificado el estilo visual de toda la interfaz web añadiendo un tono más amigable, un estilo de letra más reducido y añadiendo margenes a la pagina para centrar la vista y evitar que ésta ocupe todo el ancho de la ventana.

Vista del Administrador

Se ha dividido esta vista en cuatro grandes bloques: administradores, asignaturas, usuarios y base de datos.

El bloque de administradores es muy similar al anterior, apareciendo un listado con todos los administradores a los que podemos editar su configuración, y un botón para añadir nuevos.

El bloque de asignaturas también es parecido al anterior, pero en este caso, NO podremos añadir alumnos al crear la asignatura, sino que este procedimiento se divide en dos partes:

1. Creamos la asignatura vacía, sin alumnos ni profesores vinculados a ella
2. Añadimos los participantes editando la asignatura

Una novedad añadida es que, ahora no únicamente podemos añadir usuarios a una asignatura entrando sus datos a mano o mediante un fichero de datos extraído del campus virtual, sino que además, podemos seleccionar usuarios ya existentes en nuestra base de datos, y añadirlos.

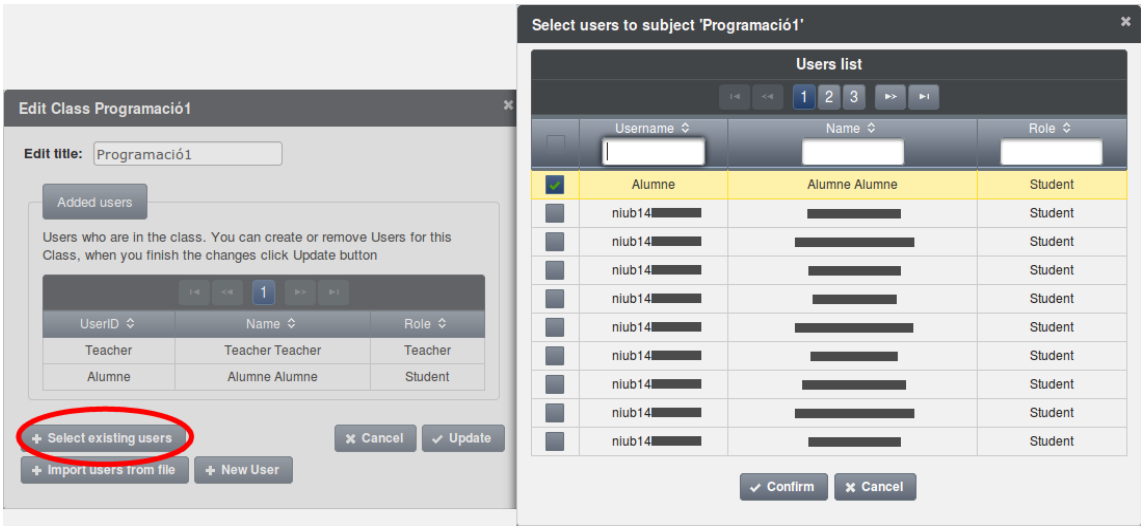

Fig 18: Añadiendo usuarios ya existentes a nuestra asignatura

Otro de los cambios importantes se encuentra en el bloque de alumnos y profesores. En el que se ha añadido el listado completo de todos los alumnos y profesores de nuestra base de datos.

Como sabemos que esta lista puede ser muy extensa, se ha añadido un buscador para poder filtrar cada uno de los campos según deseemos buscar.

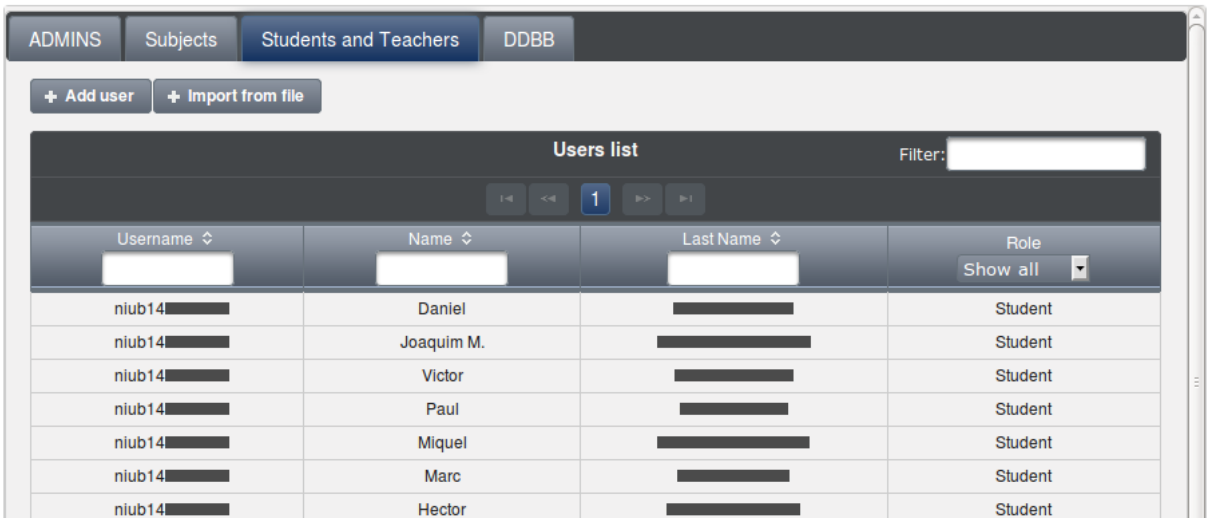

Fig 19: Bloque de estudiantes y profesores en la vista del administrador

Finalmente, el último bloque del administrador únicamente consta con un botón para limpiar todos los datos de la base de datos, excepto los datos de los administradores.

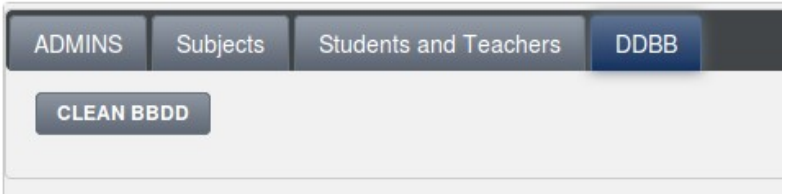

Fig 20: Bloque con botón para resetear la base de datos

Profesor

Este apartado se ha rediseñado completamente, añadiendo más posibilidades de configuración de las nuevas tareas.

Se ha eliminado la limitación de únicamente subir un sólo fichero por entrega, pudiendo extenderse a un número indefinido de ficheros. De esta manera se permite distribuir el código en tantos ficheros como el alumno necesite.

Este último cambio nos ofrece la posibilidad de compilar de otros métodos que no són usando "javac" y "gcc" para compilar Java y C respectivamente. Por ejemplo, ahora podemos, además, usar ficheros makefile para realizar configuraciones de compilación mucho más complejas. Únicamente el profesor será el que decidirá como se compilará el código, y debe ser igual para él y para los alumnos, ya que puede ser irresponsable ejecutar un fichero makefile subministrado por ellos.

Finalmente, con la posibilidad de subir múltiples ficheros, corremos el riesgo que los alumnos suban ficheros peligrosos para el sistema. Por ese caso, se ha añadido la opción que los profesores puedan seleccionar que ficheros podrán subir sus alumnos. La lista actualmente se encuentra limitada en ficheros pdf, audio, vídeo o imágenes.
El tipo de los ficheros será revisado por la función "file -i" de Linux, ya que así se comprueban los llamados "magic numbers" para determinar el tipo del fichero, sin tener en cuenta la extensión de éste.

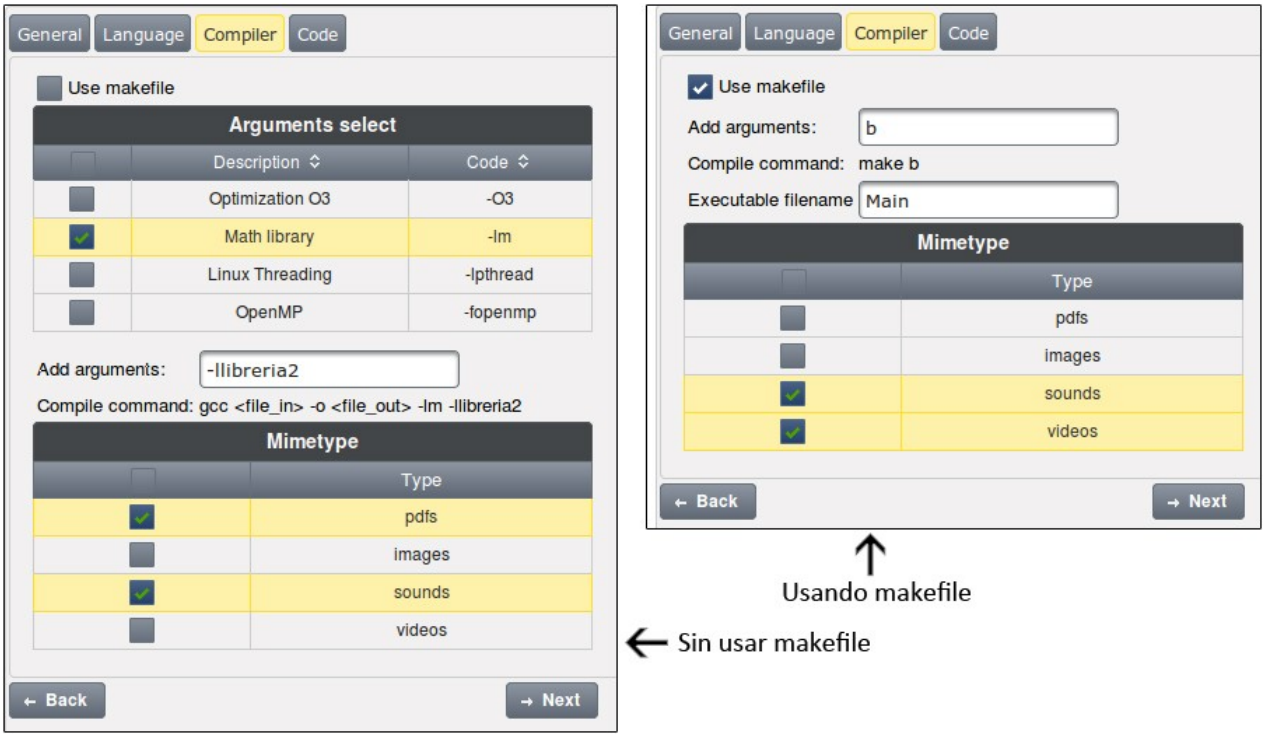

Fig 21: Configuración de la compilación de códigos

En la *Fig. 21* vemos como se pueden elegir un conjunto de librerías en una tabla predefinida para C, o añadir nuestros propios argumentos a mano. Además, veremos en tiempo real como quedaría el comando de compilación. También podremos escoger que tipo de ficheros los alumnos podrán subir.

Gracias a que los profesores subministran un makefile que los alumnos usarán, se ha podido extender esta opción de modo que sean los profesores quienes decidan que ficheros, además del makefile, quieren que se compartan con los alumnos.

Hay que tener en cuenta que estos ficheros compartidos, el alumno no tendrá acceso a ellos, ni los podrá ver. Simplemente, se añadirán junto a la entrega que éste realice.

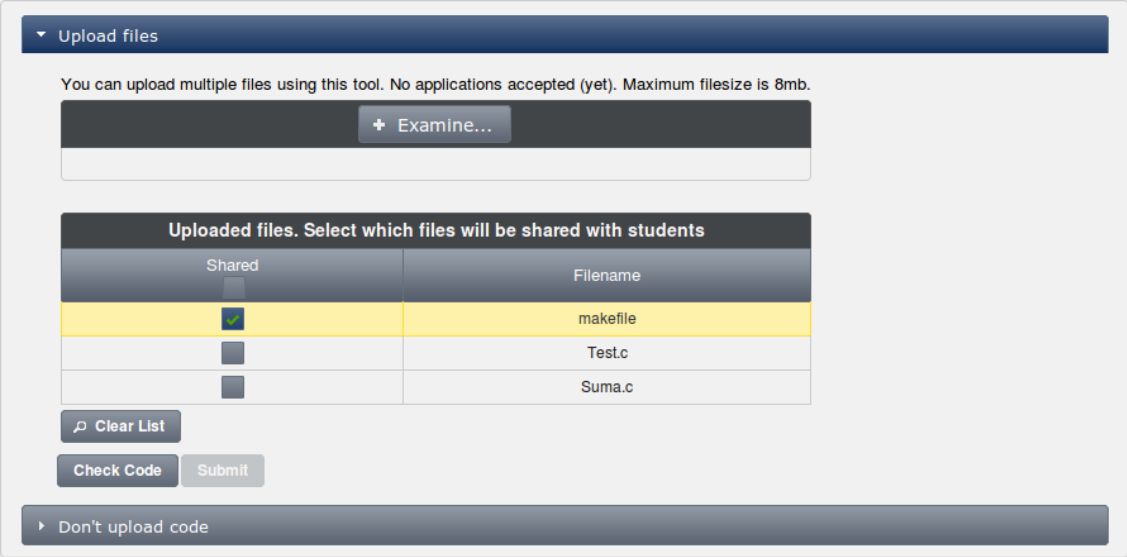

Fig 22: Tabla de elección de ficheros compartidos entre el alumno y el profesor

Alumno

Por parte del alumno, no se ha añadido ninguna funcionalidad importante, únicamente se ha adaptado el nuevo sistema de entregas con múltiples ficheros y se ha hecho funcional.

2.4 Análisis SecuritySCP

2.4.1 Análisis de los trabajos anteriores

Esta es la parte del proyecto que realmente hace el trabajo de compilación, corrección y comprobación de resultados.

SecuritySCP no interactuará con el usuario ni tendrá acceso a la base de datos a diferencia de la aplicación web. Por ese motivo, todos los datos que necesite este componente para trabajar, tendrá que ser enviado subministrado por la aplicación web.

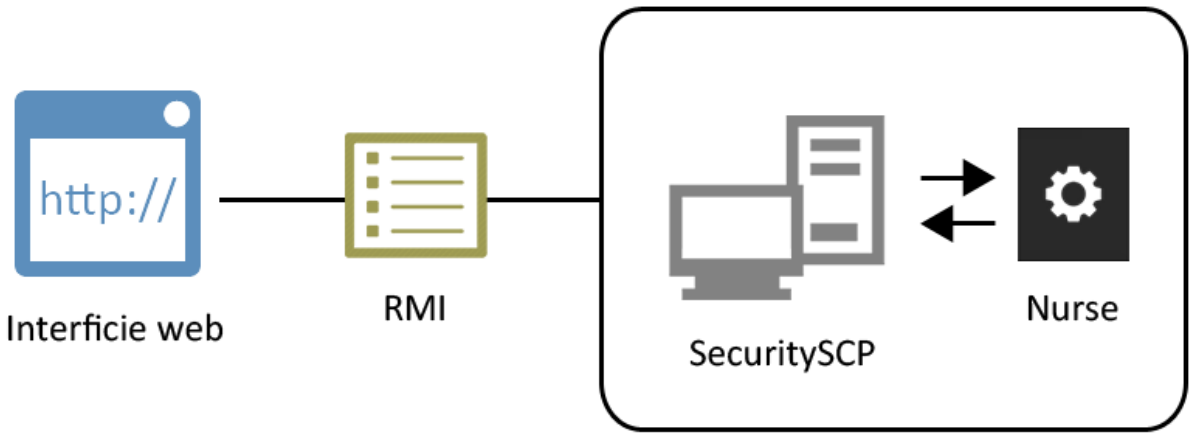

Fig 23: Esquema de los distintos nodos del sistema corrector de programas

Este componente del SCP se comunica con la aplicación web mediante objetos RMI definidos en ambas partes de la aplicación. La aplicación web se libra de la carga de corregir los programas de los alumnos y profesores, pudiendo derivar esta tarea.

SecuritySCP se ayuda de una pequeña herramienta llamada **Nurse** elaborada por antiguos proyectos de este trabajo, que se trata de un pequeño sandbox en el que se ejecutarán aquellos programas elaborados por los alumnos y profesores. Dicho sandbox obtiene la ruta del fichero ejecutable y, mediante unos ficheros de configuración, limitará los recursos que este tomará. En un principio únicamente se limita la memoria máxima de la que una aplicación dispone y, en el caso de C, se prohibirán ciertas librerías y llamadas a sistema, para evitar comportamientos indeseados.

Haciendo un resumen, las funciones generales del SecuritySCP serán las siguientes:

- **Compilar** usando los compiladores básicos del sistema Linux, javac y gcc para Java o C respectivamente. En el caso de Python no se compilará el código, únicamente le daremos el visto bueno en esta comprobación.

  javac para Java

  gcc para C

- **Ejecutar programas** previamente compilados usando el sandbox **Nurse**
- Hacer un **análisis** y **recoger los resultados** de ambos procedimientos y enviárselos a la aplicación web para ésta que modifique la base de datos o para actualizar contenidos de la propia web.

2.4.2 Mejoras implementadas en el proyecto

Esta parte del proyecto también se ha tenido que adaptar a los cambios generales sufridos en todo el proyecto, como es el caso de los múltiples ficheros.

Además, tras realizar el proyecto se ha observado y corregido el hecho que los argumentos de compilación, aunque estaban implementados en el apartado web, en la parte del SecuritySCP no se trataban.

En cuestiones de seguridad, se ha añadido también la limitación de recursos a los programas java, de forma similar a como los ejecutables C se limitaban gracias a Nurse.
Se trata de las Java Security Policies con las que, mediante un fichero con un patrón de configuración de políticas, se limitan los accesos del que un ejecutable de java dispone.

Según la configuración realizada, los únicos permisos especiales del que dispondrán los ejecutables java serán los de modificar los ficheros en su propio directorio. Todos los demás permisos como el de acceso a internet, o el de la reproducción de sonidos, están deshabilitados.

3. Diseño

3.1 Casos de uso

A continuación analizaremos los casos de uso de cada uno de los 3 tipos de usuarios que pueden navegar por nuestra aplicación. Aunque, además de esos casos de uso, todos los usuarios, además, tendrán las opciones de hacer logout y de cambiar su propia contraseña.

3.1.1 Administrador

Este diagrama de casos de uso de la Fig. 24 nos muestra las acciones que dispone el actor administrador.

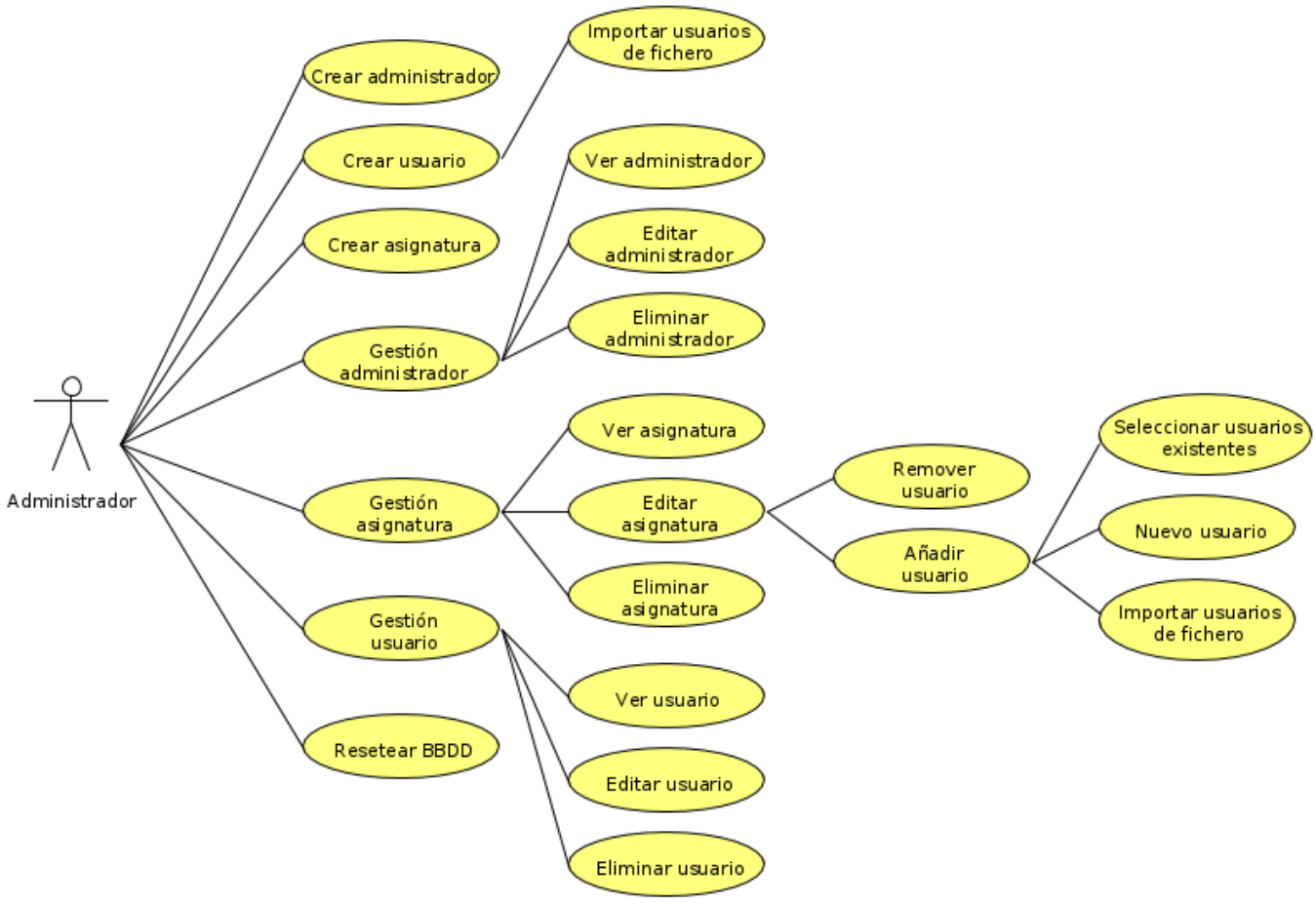

Fig 24: Diagrama de casos de uso de las acciones disponibles por el administrador

A continuación se procederá a dar una explicación detallada de cada uno de los casos de uso.

3.1.1.1 Crear Administrador

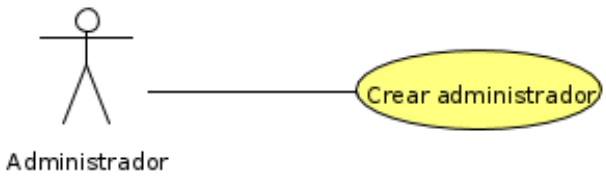

Fig 25: Caso de uso del administrador con la acción de creación de un administrador

Caso de Uso: Creación de un Administrador

Actor: Administrador

Flujo básico: Solicita al Administrador los datos necesarios para crear otro administrador. Entre esos datos encontramos un nombre de usuario, un nombre y apellidos, y la contraseña de acceso.

El nuevo administrador será registrado en la Base de Datos y será accesible para realizar las tareas de administrador.

Precondiciones: Ser administrador.

Flujo alternativo:

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

2. No completa todos los datos correctamente. La aplicación informará qué ha fallado.

3.1.1.2 Crear Usuario

Fig 26: Caso de uso del administrador con la acción de creación de un usuario

Caso de Uso: Creación de un Usuario

Actor: Administrador

Flujo básico: Solicita al Administrador los datos necesarios para crear otro usuario. Entre esos datos encontramos un nombre de usuario, un nombre y apellidos, la contraseña de acceso y el rol que desempeñará el usuario: profesor o alumno.

Además, se podrán importar múltiples alumnos seleccionando un fichero con los datos necesarios extraído del Campus Virtual.

Los nuevos usuarios serán registrados en la Base de Datos y serán accesibles a partir de ahora para realizar las tareas de profesor o alumno, según su rol indicado.

Precondiciones: Ser administrador.

Flujo alternativo:

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

2. El administrador no subministra un fichero válido para importar ficheros. La aplicación devuelve un aviso de datos inválidos.

3.1.1.3 Crear Asignatura

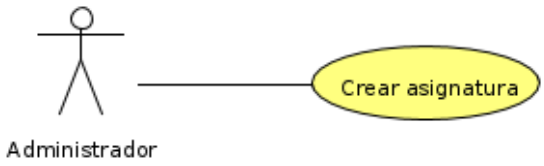

Fig 27: Caso de uso del administrador con la acción de creación de una asignatura

Caso de Uso: Creación de una asignatura

Actor: Administrador

Flujo básico: Solicita al Administrador el nombre de la asignatura.

La nueva asignatura estará vacía sin participantes, y será registrada en la Base de Datos.

Precondiciones: Ser administrador.

Flujo alternativo:

1. Nombre inválido para la asignatura. La aplicación devuelve un aviso de error.

3.1.1.3 Ver administrador

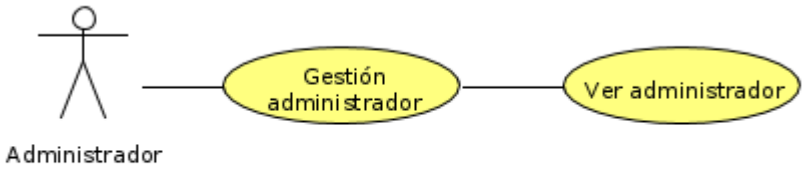

Fig 28: Caso de uso del administrador con la acción de ver un administrador

Caso de Uso: Visión deinformación de un administrador

Actor: Administrador

Flujo básico: El administrador solicita información sobre cierto administrador. La aplicación aporta información sobre el nombre de usuario, y el nombre real del administrador.

Precondiciones: Ser administrador.

3.1.1.4 Editar administrador

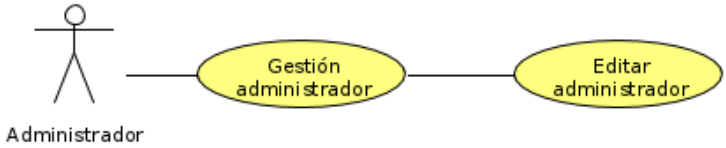

Fig 29: Caso de uso del administrador con la acción de editar un administrador

Caso de Uso: Editar datos de un administrador

Actor: Administrador

Flujo básico: El administrador solicita poder modificar los datos de información sobre cierto administrador.

La aplicación presenta los datos anteriores al administrador, y permite modificarlos.

Precondiciones: Ser administrador.

Flujo alternativo:

1. Datos modificados incorrectos. La aplicación devuelve un aviso de datos incorrectos.

3.1.1.5 Eliminar administrador

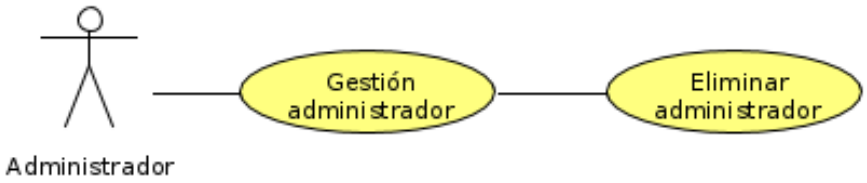

Fig 30: Caso de uso del administrador con la acción de eliminar un administrador

Caso de Uso: Eliminar administrador

Actor: Administrador

Flujo básico: El administrador selecciona eliminar administrador.

La aplicación elimina al administrador de la base de datos.

Precondiciones: Ser administrador.

3.1.1.6 Ver asignatura

Fig 31: Caso de uso del administrador con la acción de ver una asignatura

Caso de Uso: Visión de una asignatura

Actor: Administrador

Flujo básico: El administrador solicita información sobre cierta asignatura.

La aplicación aporta información sobre la asignatura.

Precondiciones: Ser administrador.

3.1.1.7 Insertar nuevo usuario en la asignatura

Fig 32: Caso de uso del administrador con la acción de insertar un nuevo usuario a una asignatura

Caso de Uso: Creación de un Usuario e insertarlo en una asignatura

Actor: Administrador

Flujo básico: Solicita al Administrador los datos necesarios para crear otro usuario. Entre esos datos encontramos un nombre de usuario, un nombre y apellidos, la contraseña de acceso y el rol que desempeñará el usuario: profesor o alumno.

El usuario introducido, si no existe, será registrado en la Base de Datos.

Finalmente, dicho usuario será añadido en la asignatura que estamos editando.

Precondiciones: Ser administrador.

Flujo alternativo:

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

2. El administrador completa los datos erróneamente. La aplicación devuelve un aviso de datos incorrectos.

3.1.1.8 Insertar usuario en la asignatura

Fig 33: Caso de uso del administrador con la acción de insertar un usuario existente a una asignatura

Caso de Uso: Insertar usuario a una asignatura

Actor: Administrador

Flujo básico: Se subministra un listado con todos los profesores/alumnos al administrador.

El administrador selecciona el/los usuario/s que formarán parte de la asignatura.

Tras aceptar, dichos usuarios serán añadidos en la asignatura que estamos editando.

Precondiciones: Ser administrador.

Flujo alternativo:

1. No selecciona ningún usuario para ser insertado. La aplicación no hará ningún cambio en la asignatura ni en sus actuales participantes.

2. El administrador des-seleccionar usuarios que ya participaban en la asignatura para eliminarlos. La aplicación eliminará dichos usuarios de la asignatura.

3.1.1.9 Importar usuarios en la asignatura

Fig 34: Caso de uso del administrador con la acción de importar usuarios de un fichero a una asignatura

Caso de Uso: Creación de un Usuario

Actor: Administrador

Flujo básico: Solicita al Administrador un fichero con datos extraídos del Campus Virtual de la Universidad de Barcelona, con la información sobre los alumnos inscritos a una asignatura.

Los nuevos usuarios serán registrados en la Base de Datos y serán añadidos a la asignatura.

Precondiciones: Ser administrador.

Flujo alternativo:

1. El administrador no subministra un fichero válido para importar ficheros. La aplicación devuelve un aviso de datos inválidos.

3.1.1.10 Eliminar usuario de una asignatura

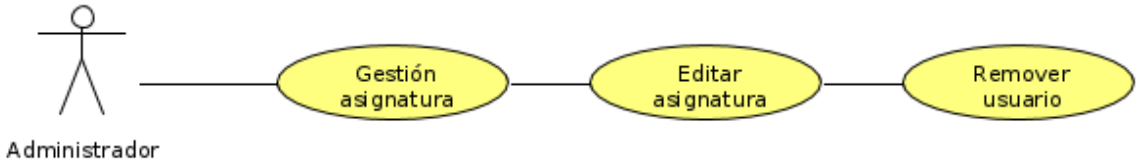

Fig 35: Caso de uso del administrador con la acción de remover un usuario de una asignatura

Caso de Uso: Remover usuario de la asignatura

Actor: Administrador

Flujo básico: El Administrador selecciona un usuario para ser removido.

La aplicación remueve al usuario de la asignatura, pero no lo eliminará completamente de la base de datos.

Precondiciones: Ser administrador.

3.1.1.11 Ver información de un usuario

Fig 36: Caso de uso del administrador con la acción de ver un usuario

Caso de Uso: Visión de un usuario

Actor: Administrador

Flujo básico: El administrador solicita información sobre cierto alumno o profesor.

La aplicación aporta información sobre el usuario.

Precondiciones: Ser administrador.

3.1.1.12 Editar usuario

Fig 37: Caso de uso del administrador con la acción de editar los datos de un usuario

Caso de Uso: Editar datos de un usuario alumno o profesor.

Actor: Administrador

Flujo básico: El administrador solicita poder modificar los datos de información sobre cierto alumno o profesor.

La aplicación presenta los datos anteriores al administrador, y permite modificarlos.

Precondiciones: Ser administrador y solicitar editar a un usuario existente.

Flujo alternativo:

1. Datos modificados incorrectos. La aplicación devuelve un aviso de datos incorrectos.

3.1.1.13 Ver información de un usuario

Fig 38: Caso de uso del administrador con la acción de eliminar un usuario

Caso de Uso: Eliminar usuario

Actor: Administrador

Flujo básico: El Administrador selecciona un usuario para ser eliminado.

La aplicación eliminará el usuario de la base de datos.

Precondiciones: Ser administrador.

3.1.1.14 Limpiar base de datos

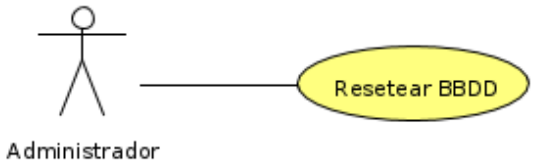

Fig 39: Caso de uso del administrador con la acción reiniciar la base de datos

Caso de Uso: Resetear Base de Datos

Actor: Administrador

Flujo básico: El Administrador selecciona la opción del reseteo de la base de datos.

La aplicación eliminará todas las entradas de la base de datos, a excepción de los administradores existentes.

Precondiciones: Ser administrador.

3.1.2 Profesor

Este diagrama de casos de uso de la Fig. 40 nos muestra las acciones específicas que dispone el actor profesor.

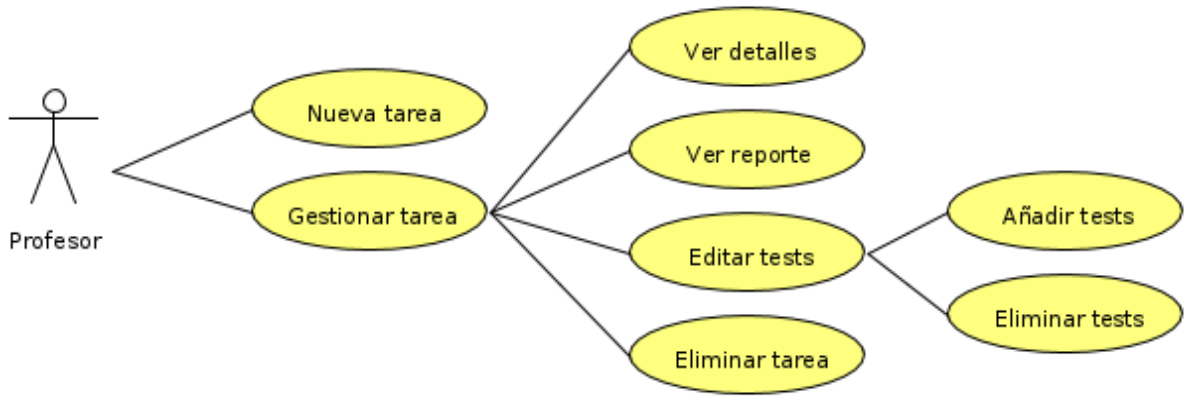

Fig 40: Diagrama de casos de uso completo del profesor

A continuación se procederá a dar una explicación detallada de cada uno de los casos de uso.

3.1.2.1 Creación de una nueva tarea

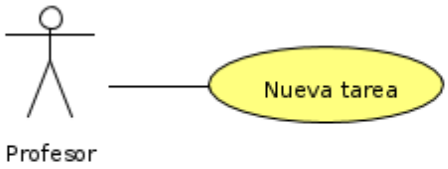

Fig 41: Caso de uso de profesor con la acción de crear nueva tarea

Caso de Uso: Creación de una tarea

Actor: Profesor

Flujo básico: Se solicitan a Profesor los datos necesarios para la creación de una nueva tarea.

La aplicación comprueba toda la información solicitada y si es correcta se registrará en la base de datos y se procederá a la creación de tests. Ver caso de uso del apartado 3.1.2.4 y 3.1.2.5.

Precondiciones: Ser profesor.

Flujo alternativo:

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

2.No completa todos los datos correctamente. La aplicación devuelve un aviso de datos incorrectos.

3. No se ha subministrado un código correcto compilable. La aplicación devuelve el aviso de error en la compilación.

4. Se ha programado el uso de archivos “makefile” pero el profesor no ha subministrado ninguno. La aplicación no dejará continuar al profesor hasta que se le subministre uno correctamente.

5. El profesor intenta subir un ejecutable. La aplicación prohíbe este tipo de archivos y se lo notifica.

3.1.2.2 Ver detalles de una tarea

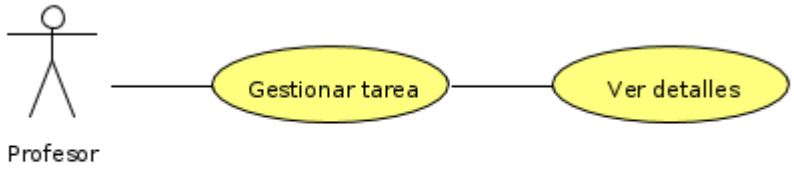

Fig 42: Caso de uso de profesor con la acción de ver detalles de una tarea

Caso de Uso: Ver tarea

Actor: Profesor

Flujo básico: El profesor solicita ver información sobre cierta tarea.

La aplicación aporta información sobre la tarea.

Precondiciones: Ser profesor con alguna tarea creada previamente.

3.1.2.3 Ver reporte de entregas en una tarea

Fig 43: Caso de uso de profesor con la acción de ver el reporte de estudiantes de una tarea

Caso de Uso: Ver reporte de entregas

Actor: Profesor

Flujo básico: El profesor solicita ver información sobre las entregas que han hecho los alumnos a cierta tarea.

La aplicación aporta información sobre la puntuación de los alumnos que han participado en la tarea.

Precondiciones: Ser profesor con alguna tarea creada previamente.

3.1.2.4 Añadir tests

Fig 44: Caso de uso de profesor con la acción de añadir tests a una tarea

Caso de Uso: Añadir tests a una tarea

Actor: Profesor

Descripción: El profesor pretende añadir tests y rellena los campos para ejecutar con argumentos o sin ellos.

La aplicación anota los tests realizados y los registra a la base de datos.

Precondiciones: Ser profesor con alguna tarea creada previamente.

Flujo alternativo:

1. No se rellenan correctamente los argumentos para la creación de un test. Se notifica al profesor que no se ha podido crear el test.

3.1.2.5 Eliminar test

Fig 45: Caso de uso de profesor con la acción de eliminar tests de una tarea

Caso de Uso: Eliminar test

Actor: Profesor

Flujo básico: El profesor solicita eliminar un test creado previamente sobre cierta tarea.

La aplicación elimina el registro de la base de datos del test eliminándolo completamente.

Precondiciones: Ser profesor con alguna tarea y test creados previamente.

3.1.2.5 Eliminar tarea

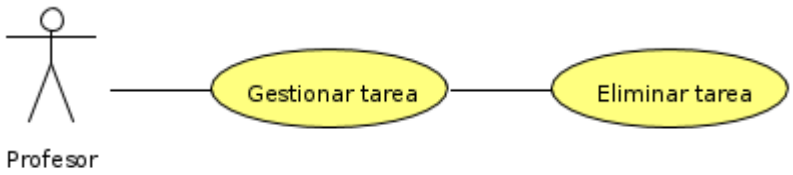

Fig 46: Caso de uso de profesor con la acción de eliminar tarea

Caso de Uso: Eliminar tarea

Actor: Profesor

Flujo básico: El profesor solicita eliminar una tarea creada previamente.

La aplicación elimina el registro de la base de datos de la tarea eliminándola completamente

Precondiciones: Ser profesor con alguna tarea creada previamente.

3.1.3 Alumno

Este diagrama de casos de uso de la Fig. 47 nos muestra las acciones que dispone el actor profesor.

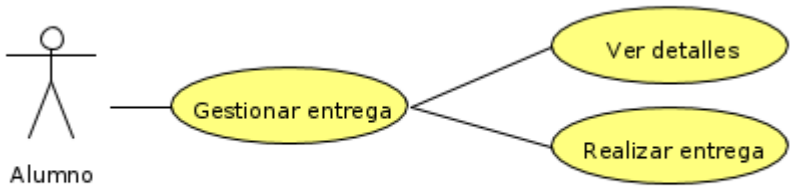

Fig 47: Diagrama de casos de uso completo del alumno

A continuación se procederá a dar una explicación detallada de cada uno de los casos de uso.

3.1.3.1 Entregar tarea

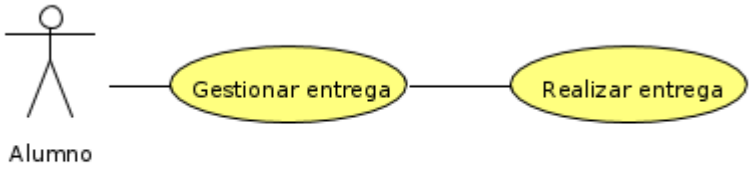

Fig 48: Caso de uso del alumno con la acción de entregar un código

Caso de Uso: Realizar entrega de una tarea

Actor: Alumno

Flujo básico: El alumno solicita subministrar el código para realizar la entrega de una tarea.

La aplicación lo recibe y le devuelve inmediatamente el resultado.

Precondiciones: Ser alumno con alguna tarea pendiente.

Flujo alternativo:

1. El código entregado no puede ser compilado según las opciones del profesor. La aplicación guardará como resultado del test el error obtenido tras la compilación, pero permitirá resubir el código nuevamente.

3.1.2.5 Ver detalles de una entrega

Fig 49: Caso de uso del alumno con la acción de ver detalles sobre una entrega

Caso de Uso: Ver detalles de una entrega

Actor: Alumno

Flujo básico: El alumno solicita ver información sobre una entrega realizada previamente.

La aplicación muestra información sobre la tarea.

Precondiciones: Ser alumno con alguna tarea pendiente.

4. Implementación

Como ya hemos comentado en otros apartados del informe, el sistema se ha dividido en dos aplicaciones: SCP4.0 y SecuritySCP. Siendo entre ellos completamente independientes y pudiendo estar ejecutándose en diferentes máquinas.

SCP4.0 se encarga de toda la interfaz gráfica de la web, y es la aporta al usuario acceso a la aplicación. Permite al usuario interactuar y ofrecerle los servicios vistos en el análisis. Además, esta es la única con acceso a la base de datos del sistema.

SecuritySCP, en cambio, es el encargado de las tareas de compilación, ejecución y recogida de resultados de los programas subidos por cualquier usuario, ya sea profesor o alumno.

La interacción entre ambas aplicaciones se realiza mediante Java RMI, y el procedimiento es el siguiente: el usuario se comunica mediante internet con la aplicación SCP4.0, la cual le muestra una página web. SCP4.0 conecta con SecuritySCP siempre que el usuario requiera de la comprobación de un código. SCP4.0 entonces, recoge los datos subministrados por SecuritySCP y modifica la base de datos si es necesario, o muestra al usuario dicha información.

En la Fig. 50 se muestra un diagrama de funcionalidad en conjunto del presente proyecto. En éste se ilustra la interacción entre el usuario y el sistema. Además se observa las interacciones internas de la aplicación mostrando que la interfície web y SecuritySCP pueden o bien correr en distinta máquinas unidas por Internet o por Red, o bien encontrarse ambas aplicaciones en una misma máquina.

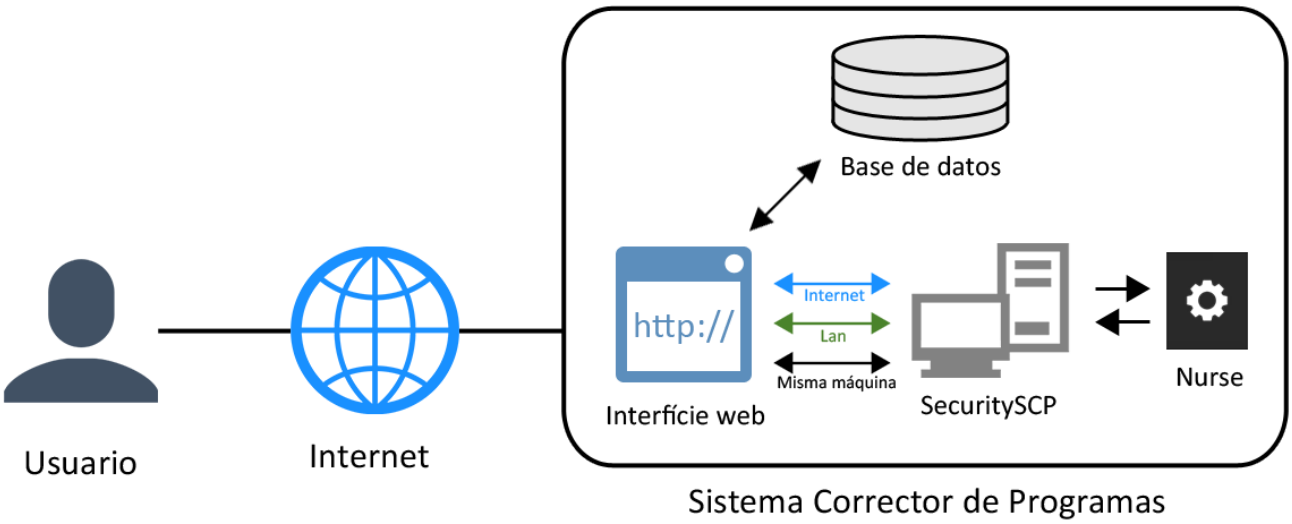

Fig 50: Diagrama de funcionamiento completo entre usuario, SCP4.0 y SecuritySCP

Linux

El proyecto debe ejecutarse bajo un sistema Linux, ya que, tanto la interfície web, SecuritySCP o la herramienta Nurse utilizan funciones especificas para este sistema operativo.

Además, Linux es un sistema operativo de software libre y ofrece un entorno adecuado para la programación. Linux ofrece también una buena seguridad y consume pocos recursos.

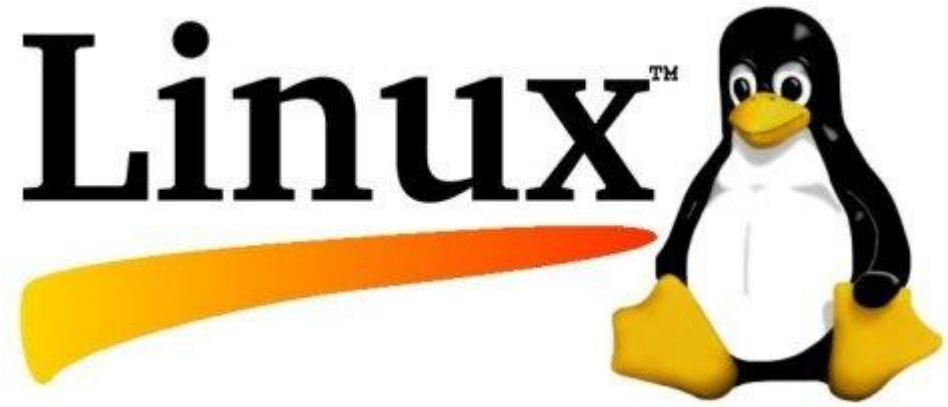

Fig 51: Logotipo de Linux

4.1 Web SCP4.0

4.1.1 Explicación

La aplicación web se aloja en un servidor Apache Tomcat, que se trata de una ligera estructura en java preparada para montar servidores web.

Apache Tomcat tiene una alta compatibilidad con el entorno de desarrollo integrado IDE Netbeans. En nuestro caso todo ha sido configurado para trabajar usando JavaServer Faces (JSF), un lenguaje de programación web, similar al PHP, pero orientado a JavaBeans, es decir, clases programadas en java que se comunicarán directamente con el código web y aportarán funciones a ésta.

Finalmente, la apariencia de la web ha sido aportada por la librería PrimeFaces, que inicialmente se encontraba en su versión 3.0.1, se intentó actualizar a la versión 4.0, pero debido a problemas con el javascript e Internet Explorer, se ha utilizado finalmente la versión 3.5. PrimeFaces aporta más funciones a la programación JSF, y además da un aspecto visual.

Podemos ver una pequeña demostración del funcionamiento de PrimeFaces con el siguiente código.

```
<p:commandButton value="Basic" type="button" onclick="PF('dlg1').show();" />

<p:dialog header="Basic Dialog" widgetVar="dlg1" minHeight="40">
        <h:outputText value="Resistance to PrimeFaces is futile!" />
</p:dialog>
```

Finalmente en la web veríamos un botón que al hacerle Click nos aparecería un diálogo

Fig 52: Ejemplo de diálogo usando PrimeFaces 4.0

Dejando de lado PrimeFaces y volviendo a los Beans con los que se ha programado la aplicación web, cada página de la web puede tener un número indefinido de Beans y cada uno está orientado a dar alguna funcionalidad a la web. En nuestro caso, cada una de las páginas accesibles se controla mediante un solo Managed Bean:

Página Web	**Nombre del Bean**	**Java Class**
login.xhtml	loginBean	LoginBean.java
admin.xhtml	adminBean	AdminBean.java
teacher.xhtml	teacherViewBean	TeacherViewBean.java
newtask.xhtml	taskBean	TaskBean.java
addtests.xhtml	testBean	TestBean.java
report.xhtml	reportBean	ReportBean.java
student.xhtml	studentBean	StudentBean.java
newsubmission.xhtml	submissionBean	SubmissionBean.java

Cada Bean o ManagedBean se inicializa cada vez que es llamado por primera vez, por lo tanto, se creará cada vez que accedamos a una nueva página y necesitemos cargar los recursos que el Bean nos aporte. Además, el Bean, se mantiene activo durante toda la permanencia en esa página, de este modo podremos llamar a métodos del Bean para realizar distintas funciones.

4.1.2 Diagramas de Flujo

Los diagramas de flujo que veremos a continuación son diagramas sobre el procedimiento del usuario al realizar las tareas. Dentro de la estructura del lenguaje usado para la página web (JavaServer Faces) no es posible hacer un diagrama de flujos de todas las llamadas que se hacen al servidor, y todo el procedimiento interno que siguen.

4.1.2.1 – Administrador: Crear Administrador

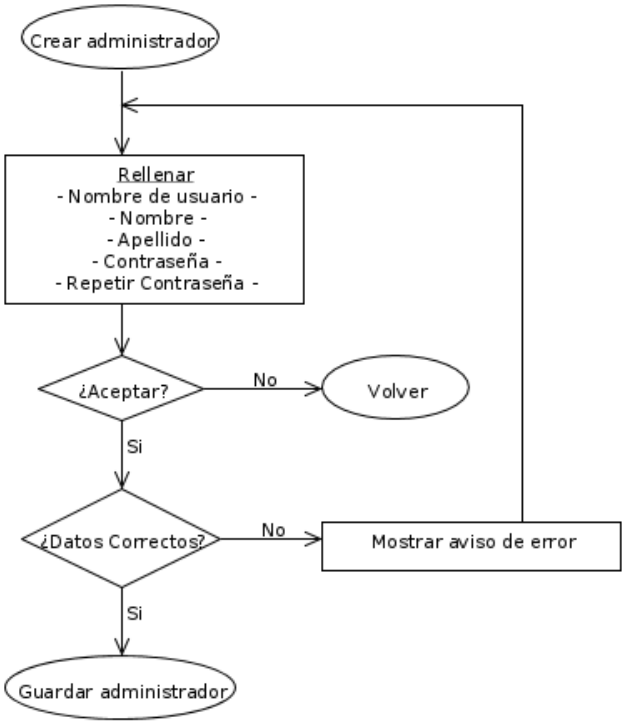

Fig 53: Diagrama de flujo de creación de un administrador

4.1.2.2 – Administrador: Crear asignatura

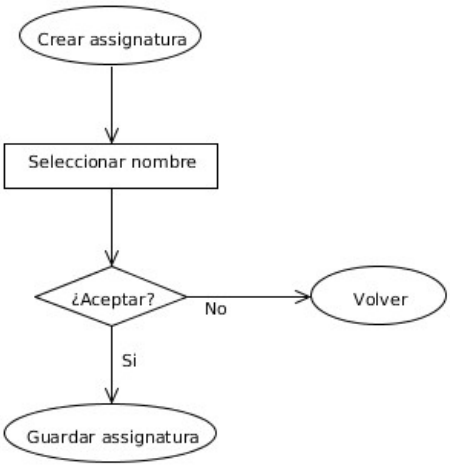

Fig 54: Diagrama de flujo de creación de una asignatura

4.1.2.3 – Administrador: Crear usuario

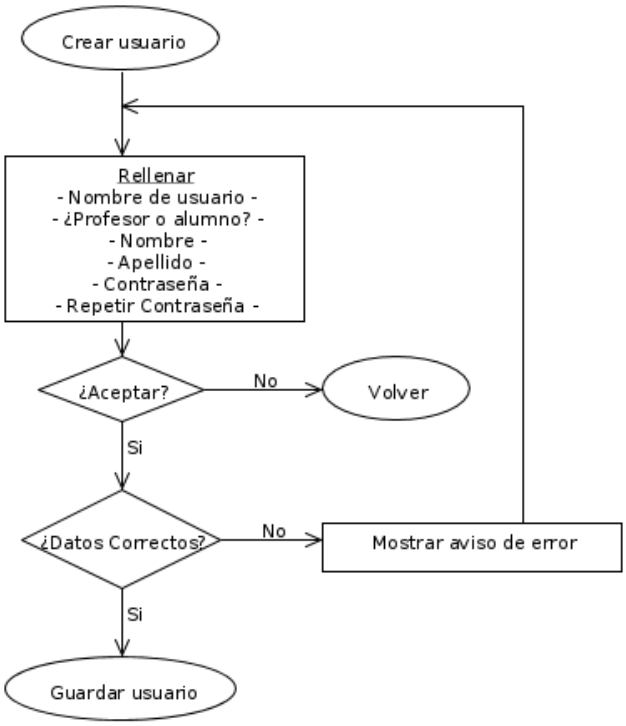

Fig 55: Diagrama de flujo de creación de un usuario

4.1.2.4 – Administrador: Importar usuarios de fichero

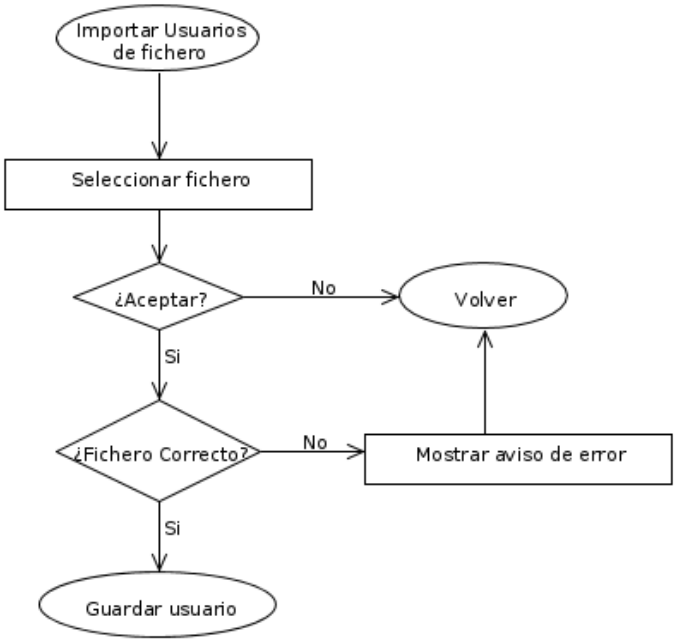

Fig 56: Diagrama de flujo del la importación de un fichero de usuarios

4.1.2.5 – Administrador: Resetear Base de Datos

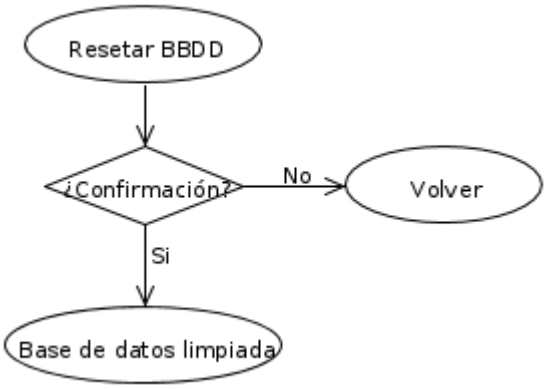

Fig 57: Diagrama de flujo para resetear la base de datos

4.1.2.6 – Administrador: Eliminar elementos

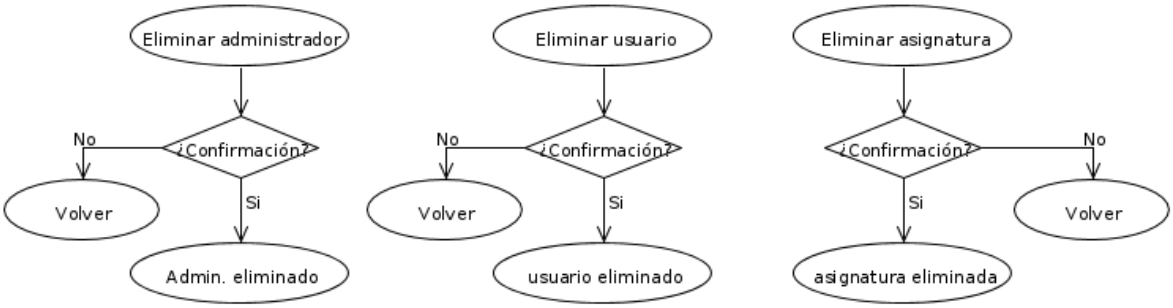

Fig 58: Diagramas de flujo para eliminar administradores, usuarios o asignaturas

4.1.2.7 – Profesor: Crear Tarea

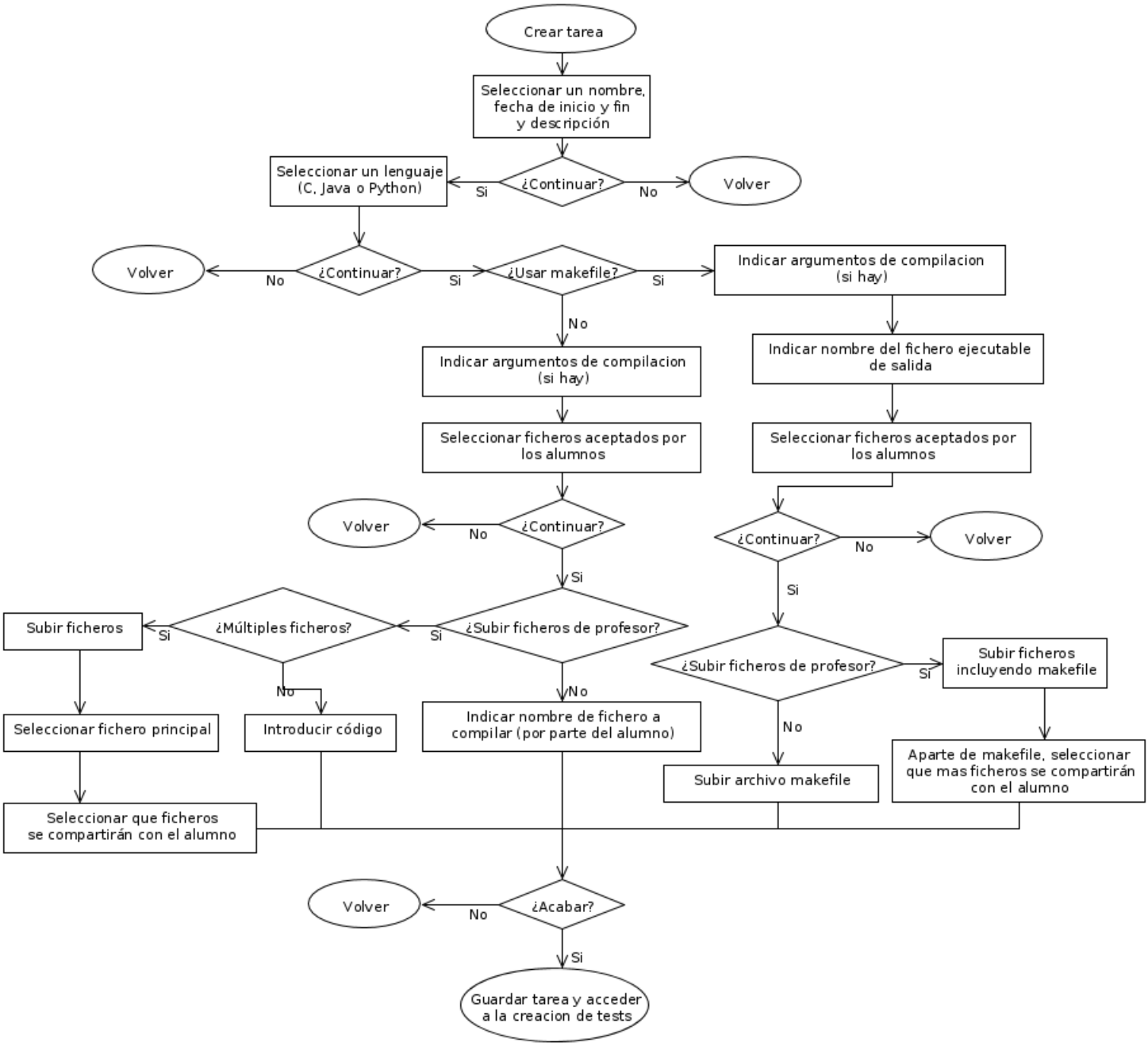

Fig 59: Diagrama de flujo de la creación de una tarea

4.1.2.8 – Alumno: Entregar código

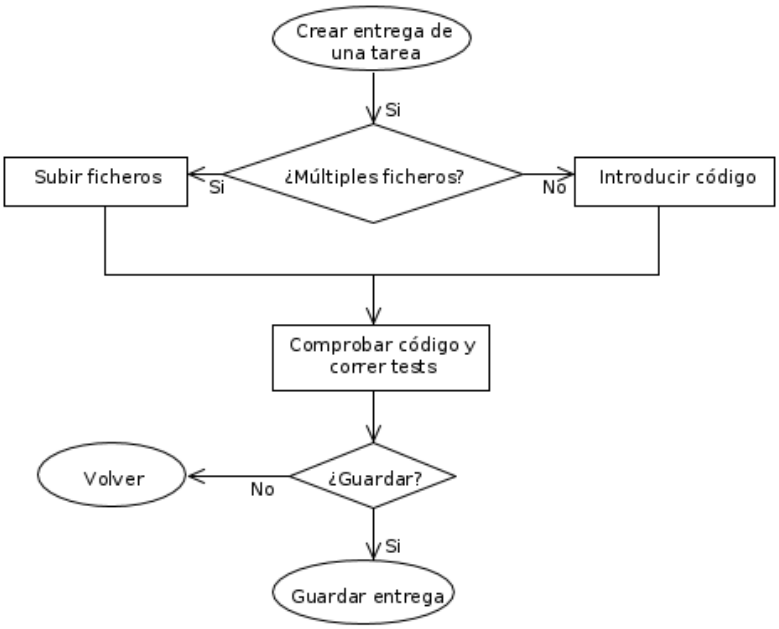

Fig 60: Diagrama de flujo de la creación de una tarea

4.1.3 Diagrama de clases

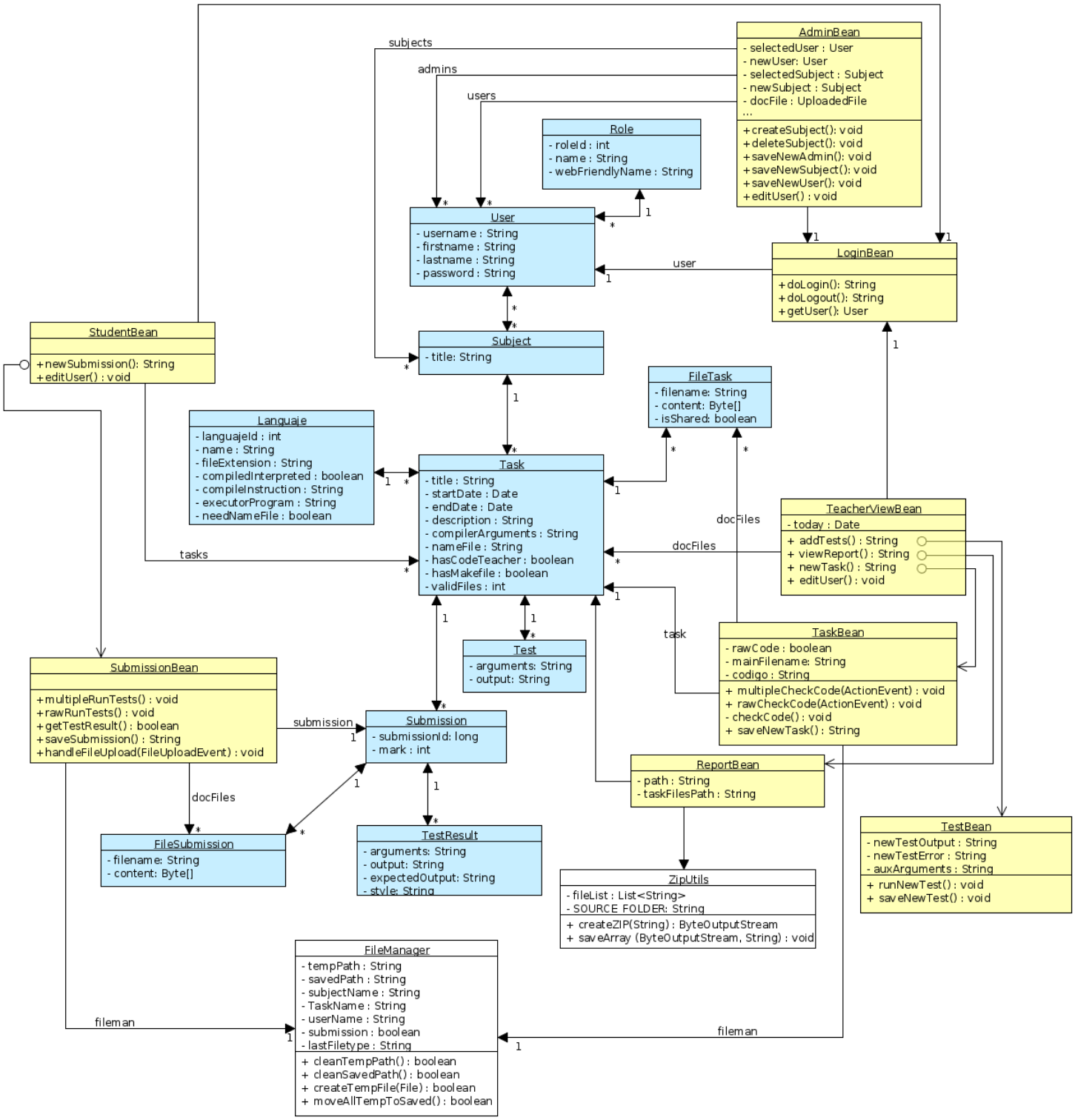

Fig 61: Diagrama de clases de la aplicación web sin tener en cuenta la Base de Datos ni conexión RMI

Leyenda

Amarillo: Los Beans, inicializados a medida que vamos avanzando por la web.

Azul: Clases correspondientes a Entidades de la Base de Datos SQL. Para leer de ellas se habrá de acceder a MySQL usando Hibernate.

Explicación del diagrama

Cuando trabajamos con JSF hacen falta muchas interacciones entre el servidor y el cliente. Por lo tanto, el servidor está continuamente enviando nueva información o incluso a veces modificando los elementos que el usuario ve por la pantalla. Todo eso implica tener un número muy elevado de funciones y atributos en el Bean activo si la página requiere de muchas interacciones.

Un ejemplo claro es, para visualizar una lista de elementos en el cliente web, hace falta un objeto "List<..> items" en el servidor. Si a esa lista le añadimos filtros, necesitaremos otro objeto "List<..> filteredItems" en el servidor para los valores filtrados. Si quisiéramos que se pudieran seleccionar elementos, necesitaríamos una tercera lista "List<..> selectedItems". Además, para todos estos objetos serían necesarias sus funciones get y set para que el cliente pudiera acceder a ellos. Cada input de nuestra web también necesita de un objeto java en el Bean correspondiente.

Por este motivo que el número de atributos y variables en el diagrama ha sido simplificado y reducido.

Como ya comentamos al principio, usamos la tecnología Spring para mantener el código limpio y de fácil mantenimiento, sin embargo, eso implica tener declaradas distintas clases java, una para cada capa de acceso a los datos. Se ha seguido el patrón de Business Object (BO) y Data Access Object (DAO) para identificar cada una de las capas de acceso a los datos de manera clara y evitar la confusión en el proyecto.

Hemos usado esta estructura junto a Hibernate creando una class BO y DAO para acceder a la base de datos. En el diagrama de clases no se han tenido en cuenta, y se han tratado las clases extraídas de la base de datos como clases de información convencionales.

Finalmente, hay que tener en cuenta que, al extraer los registros de la base de datos usando Hibernate, estos objetos están todos relacionados entre ellos, es decir, todas las relaciones son bidireccionales, lo cual se ha plasmado en el diagrama.

Se han excluido las clases relacionadas con RMI y el intercambio de información entre SCP4.0 y SecuritySCP, sin embargo las veremos en el siguiente punto 4.2.3.

4.2 SecuritySCP

4.2.1 Explicación

SecuritySCP, como ya hemos comentado previamente, es el encargado de compilar, ejecutar y devolver los resultados obtenidos tras aplicar los tests configurados.

La aplicación web se conectará a esta aplicación secundaria cada vez que requiera de sus servicios.

Clases de comunicación

Para transmitir información sobre las tareas, los tests y los resultados obtenidos se han definido una serie de clases serializables pensadas para el envío de paquetes de datos con toda la información necesaria.

- **InfoError**. Clase que recoge la información sobre cómo ha ido el proceso de compilación o ejecución, y en caso de error, indica qué tipo de error ha sucedido.
- **InfoResult**. Clase que guarda los resultados tras la ejecución de un único test.
- **InfoResults**. Clase que guarda información del resultado de correr el código del alumno con todos los tests asociados a la tarea que el profesor a creado.
- **InfoTarea**. Clase que guarda la información asociada a una tarea, así como los ficheros que la forman.
- **InfoTests**. Clase que guarda información asociada a un test.

RMI

Permite compartir interfícies entre diferentes aplicaciones, de manera que al llamar a una función concreta, se ejecutará en la aplicación correspondiente. Dichas aplicaciones pueden no encontrarse en una misma máquina y comunicarse usando la red.

SecuritySCP se encarga de montar el servidor RMI y de subministrar a ese servidor un nombre, y un puntero a un objeto a nuestra aplicación. La clase del objeto deberá implementar una interfície que será compartida con todos aquellos procesos que quieran usar sus servicios.

Por otra parte, cualquier otra aplicación (la aplicación web en nuestro caso) consulta dicho servidor RMI, y recoge el puntero correspondiente según el nombre entrado. Una vez obtenido el objeto, ya podremos usarlo con normalidad, teniendo en cuenta que cada vez que lo usemos, se estará ejecutando código en la aplicación SecuritySCP.

Estas funciones definidas en la interfície se envían y reciben únicamente atributos serializables, y en caso de enviar información especifica se usaran los objetos “Info” definidos anteriormente.

Todos los métodos nombrados en esa interfície son los siguientes:

- **InfoError securityCheckCode (InfoTarea infoTarea)**
  Método que compila el código contenido en el objeto 'infoTarea' en el caso de que éste esté escrito en C o Java.
- **InfoError securityNewTask (InfoTarea infoTarea)**
  Método que crea una nueva tarea con los datos de 'infoTarea' y la guarda en el disco.
- **InfoError securityDeleteTask (InfoTarea infoTarea)**
  Método que elimina del disco la tarea asociada a 'infoTarea'.
- **InfoError securitySaveTest (InfoTest infoTest)**
  Método que graba el test 'infoTest' en la tarea asociada a disco.
- **InfoError securityRunTest (InfoTest infoTest)**
  Método que ejecuta la tarea asociada de 'infoTest' con los argumentos de 'infoTest'.
- **InfoError securityDeleteTest (InfoTest infoTest)**
  Método que elimina del disco el test asociado a 'infoTest'.
- **InfoResults securityStudentRunTask (InfoTarea infoTarea)**
  Método que ejecuta el código asociado a 'infoTarea' del alumno con todos los test que el profesor ha introducido para esta tarea.
- **InfoError securityCleanData ()**
  Método que elimina todas las tareas y tests creados por SSCP en su disco.
- **void setShowHelpDebug (boolean show)**
  Método que permite activar y desactivar la ayuda de debug.

4.2.2 Diagrama de flujos

A continuación veremos los diagramas de flujos de esas funciones implementadas vía RMI que la aplicación inicia y SecuritySCP ejecuta.

4.2.2.1 Comprobar código (securityCheckCode)

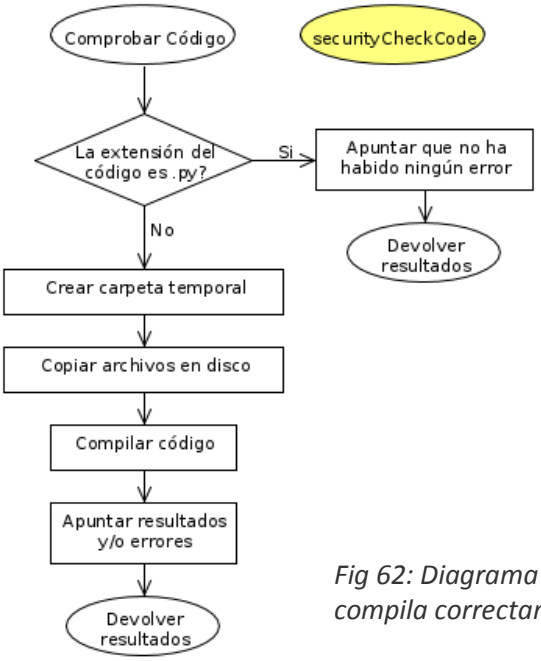

Fig 62: Diagrama de flujo para comprobar si el código de una tarea compila correctamente

4.2.2.2 Guardar nueva tarea a disco (securityNewTask)

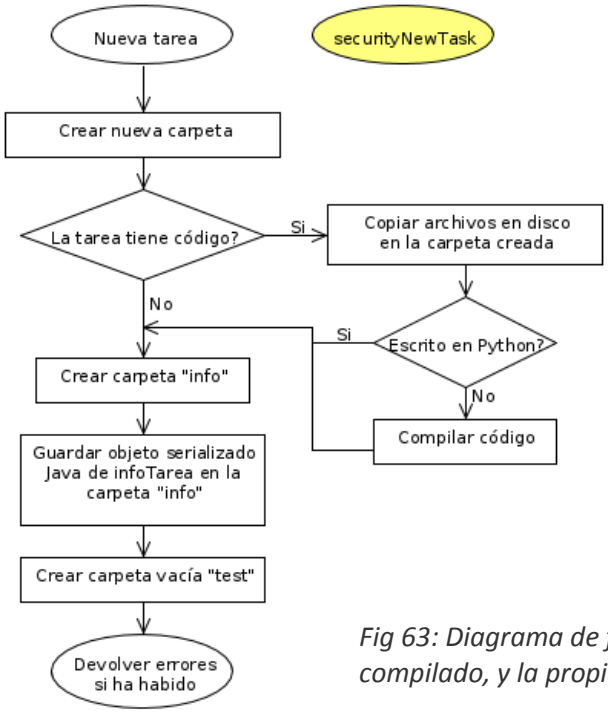

Fig 63: Diagrama de flujo para guardar los ficheros de una tarea, el código compilado, y la propia tarea en si serializada en una carpeta del sistema

4.2.2.3 Eliminar tarea (securityDeleteTask)

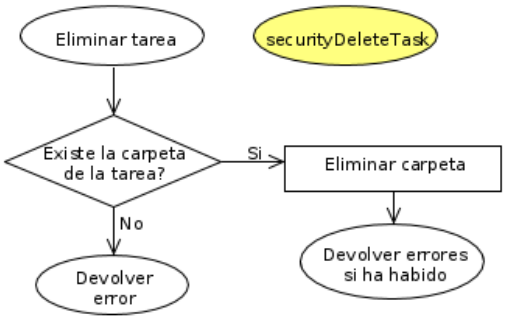

Fig 64: Diagrama de flujo para eliminar del disco una tarea

4.2.2.4 Guardar test a disco (securitySaveTest)

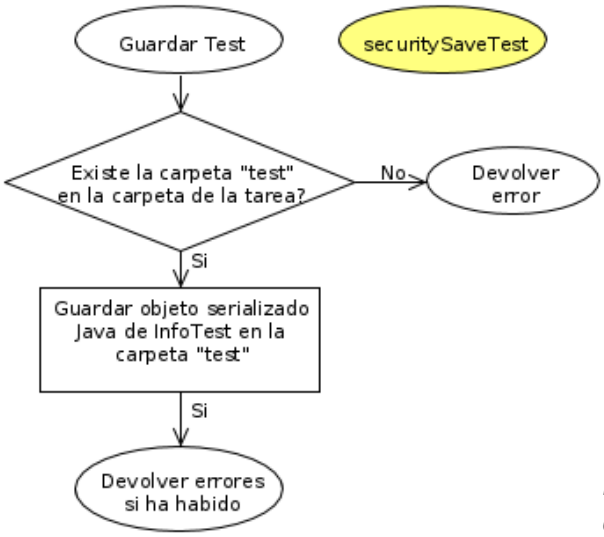

Fig 65: Diagrama de flujo para guardar un objeto de java con la información de un test a disco

4.2.2.5 Ejecutar test (securityRunTest)

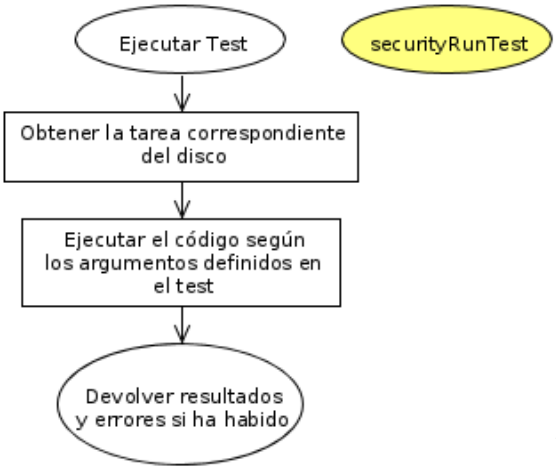

Fig 66: Diagrama de flujo para ejecutar un test sobre una tarea previamente guardada y devolver los resultados

4.2.2.6 Eliminar test (securityDeleteTest)

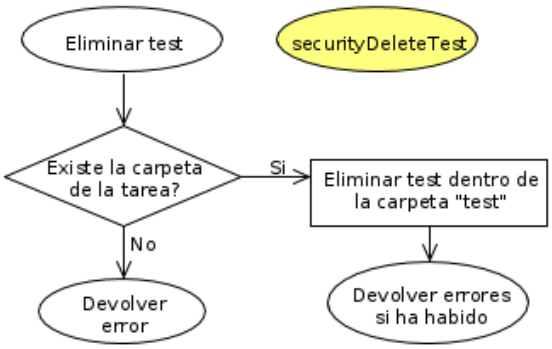

Fig 67: Diagrama de flujo para ejecutar un test sobre una tarea previamente guardada y devolver los resultados

4.2.2.7 Ejecutar código alumno (securityStudentRunTask)

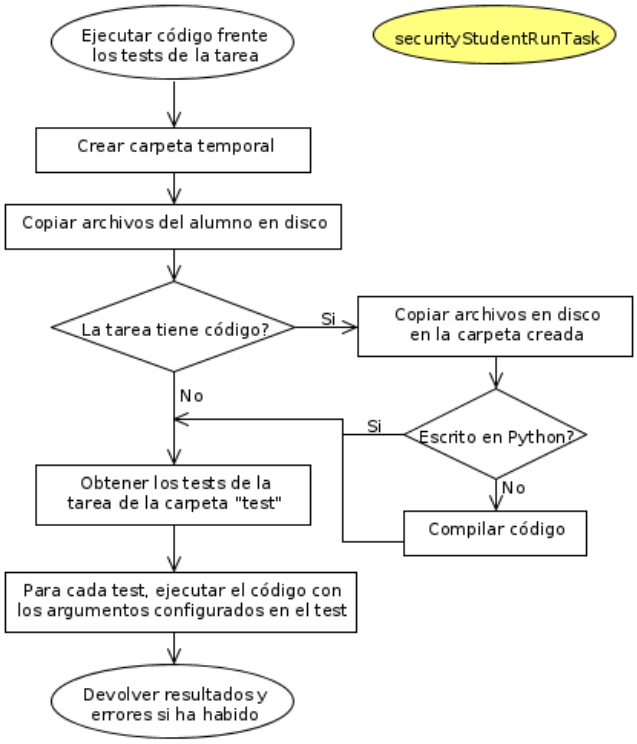

Fig 68: Diagrama de flujo para ejecutar el código de un alumno frente a todos los test configurados en la tarea que el alumno está ejecutando.

4.2.2.8 Limpiar datos (securityCleanData)

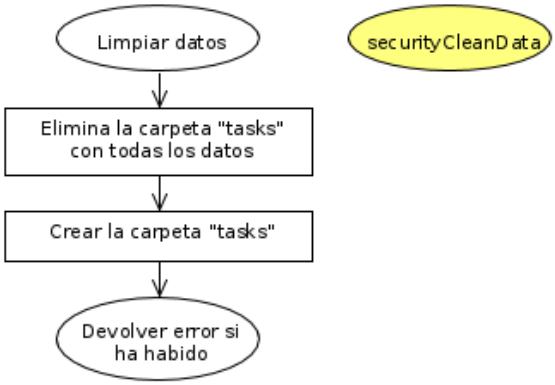

Fig 69: Diagrama de flujo para eliminar todo el contenido en el disco sobre las tareas y los tests.

4.2.3 Diagrama de clases

En el siguiente diagrama podemos ver marcada de color naranja la interfície utilizada para la conexión RMI con la aplicación Web.

Hay que tener en cuenta que todas las relaciones marcadas son relaciones usadas dentro de la funciones. Al tratarse, en su mayoría de clases estáticas, no necesitan inicialización de ningún tipo ni se disponen de atributos globales.

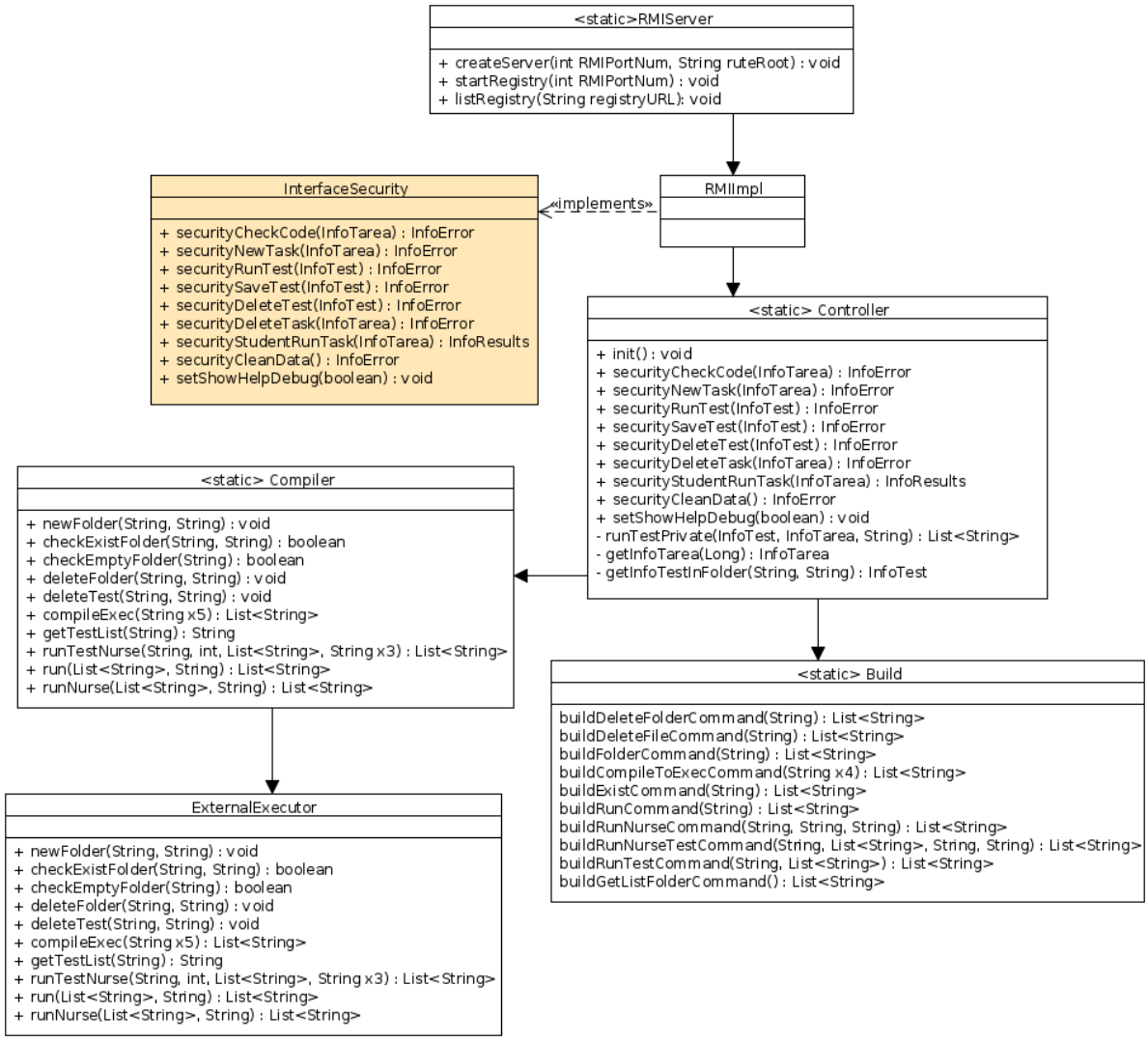

Fig 70: Diagrama de clases de SecuritySCP

4.3 Nurse

4.3.1 Explicación

Nurse es esa herramienta usada por SecuritySCP para limitar la ejecución de los programas previamente compilados.

Nurse hace de intermediario entre el sistema operativo y la aplicación, tal y como vemos en el siguiente esquema.

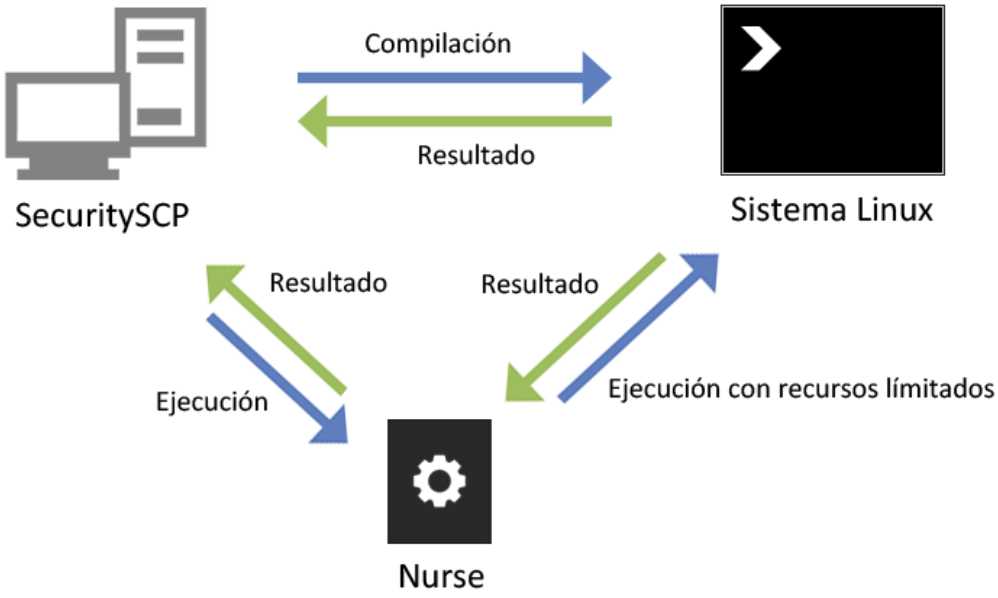

Fig 71: Esquema mostrando la relación entre SecuritySCP y las llamadas a sistema

Esta herramienta será llamada por SecuritySCP mediante el sistema operativo Linux, y el resultado que éste devuelva será el del resultado de la aplicación ejecutada.

La función de esta herramienta es la de limitar la memoria RAM disponible para la aplicación, y únicamente para C, también se limitarán algunas llamadas al sistema.

Para usar correctamente Nurse, debemos situarnos en la carpeta y escribir el siguiente comando:

$ src/sandbox <**0** ó **1**> <**comandos a ejecutar**>

Dónde el primer *<0 ó 1>* índica si va a haber una ejecución libre de los recursos. Como únicamente se prohibirán ciertas llamadas al ejecutarse en C, se deberá llamar con el valor de 0 cuando escribamos en este lenguaje, y con el valor de 1 en cualquier otro.

Finalmente, dentro los *<comandos a ejecutar>* debemos incluir todos los comandos que Nurse ejecutará, incluyendo "/usr/bin/java" y "/usr/bin/python" en el caso de java y python respectivamente.

4.3.2 Políticas Java

También se ha conseguido limitar las llamadas en el lenguaje Java usando lo conocido como *políticas* con las que podremos definir un fichero .policy y modificar los permisos con los que un programa se ejecutará.

En nuestra aplicación hemos limitado el acceso de lectura, escritura y ejecución a las aplicaciones usando las políticas. Únicamente tendremos acceso en la carpeta donde se encuentre ejecutable, todo tipo de conexión a internet o intento de cambiar el propietario de los ficheros, entre otras cosas, se encuentra bloqueado. Se han hecho todas las prohibiciones necesarias para que la ejecución del código sea segura.

Para ejecutar un código con este tipo de limitaciones se debe hacer mediante la consola con el siguiente comando

/usr/bin/java -Djava.security.policy=<**fichero de configuración de políticas**>
<**nombre ejecutable**> <**argumentos de ejecución**>

Nuestro fichero de políticas únicamente está configurado de este manera, de modo que todos los demás permisos especiales estarán prohibidos.

```
grant {
   permission java.io.FilePermission "./-", "write, read, delete";
};
```

Entonces, los comandos correctos, y los que usa SecuritySCP para ejecutar los programas de los alumnos y profesores usando Nurse en los distintos lenguajes serían:

En C	$ src/sandbox **0** <**ruta del ejecutable**> <**argumentos**>
En Java	$ src/sandbox **1** /usr/bin/java -cp <**ruta classpath**> <**configuración de políticas**> <**ruta del executable**> <**argumentos**>
En Python	$ src/sandbox **1** /usr/bin/python <**ruta del executable**> <**argumentos**>

5. Resultados

5.1 Distintos navegadores

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

Hay que comentar que todas las pruebas durante el transcurso del proyecto se han hecho con Firefox 28 y 29, y que tanto en Firefox como en Chrome no hay ningún tipo de problemas observables.

En Internet Explorer, en cambio, encontramos un problema en la versión 8 testeada. Se trata de un problema visual que hace que las dimensiones no sean las correctas, y se vean algunos elementos mal situados.

Actualmente con la versión de PrimeFaces 3.5 la aplicación es funcional en Internet Explorer. Se intentó en su momento usar PrimeFaces 4.0, pero al usarlo, encontrábamos una serie de errores con JavaScript y Ajax, por lo que finalmente decidimos usar 3.5.

A continuación mostraremos la vista del profesor en cada uno de los navegadores testeados.

5.1.1 Mozilla Firefox

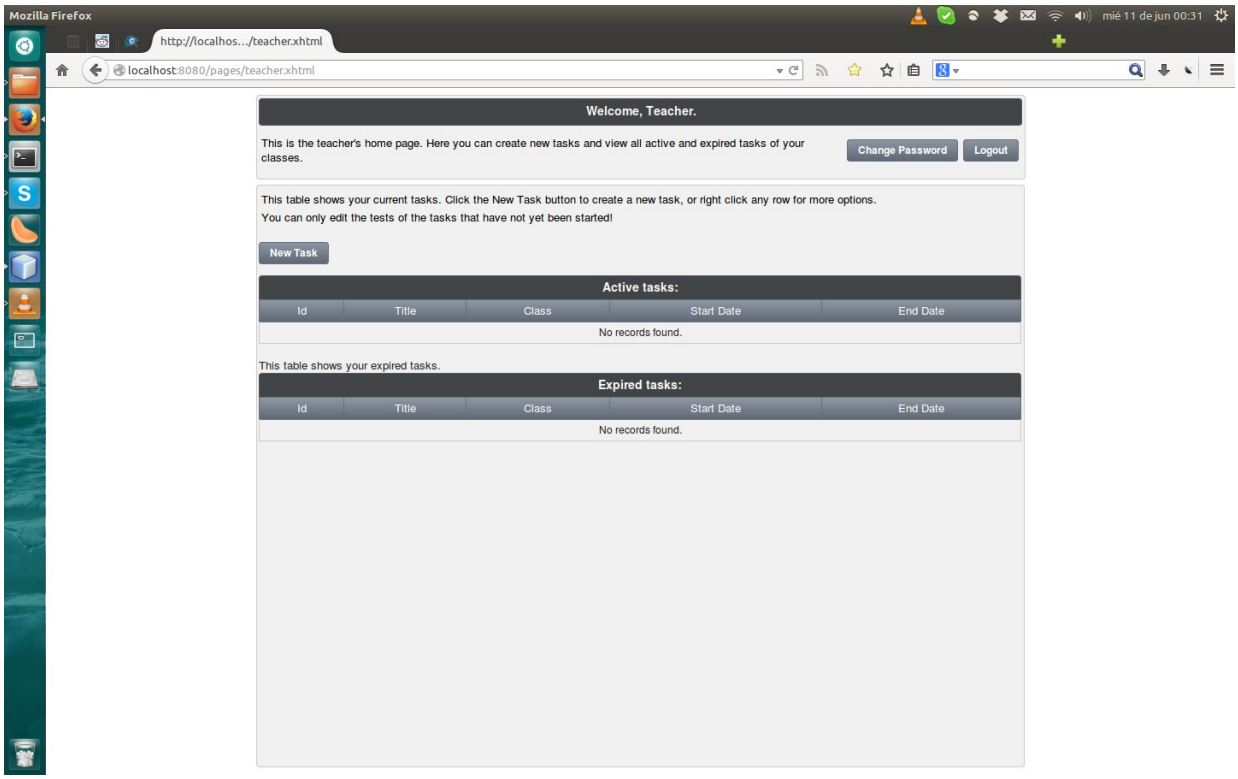

Fig 72: Vista principal del profesor vista desde Mozilla Firefox

5.1.2 Chrome

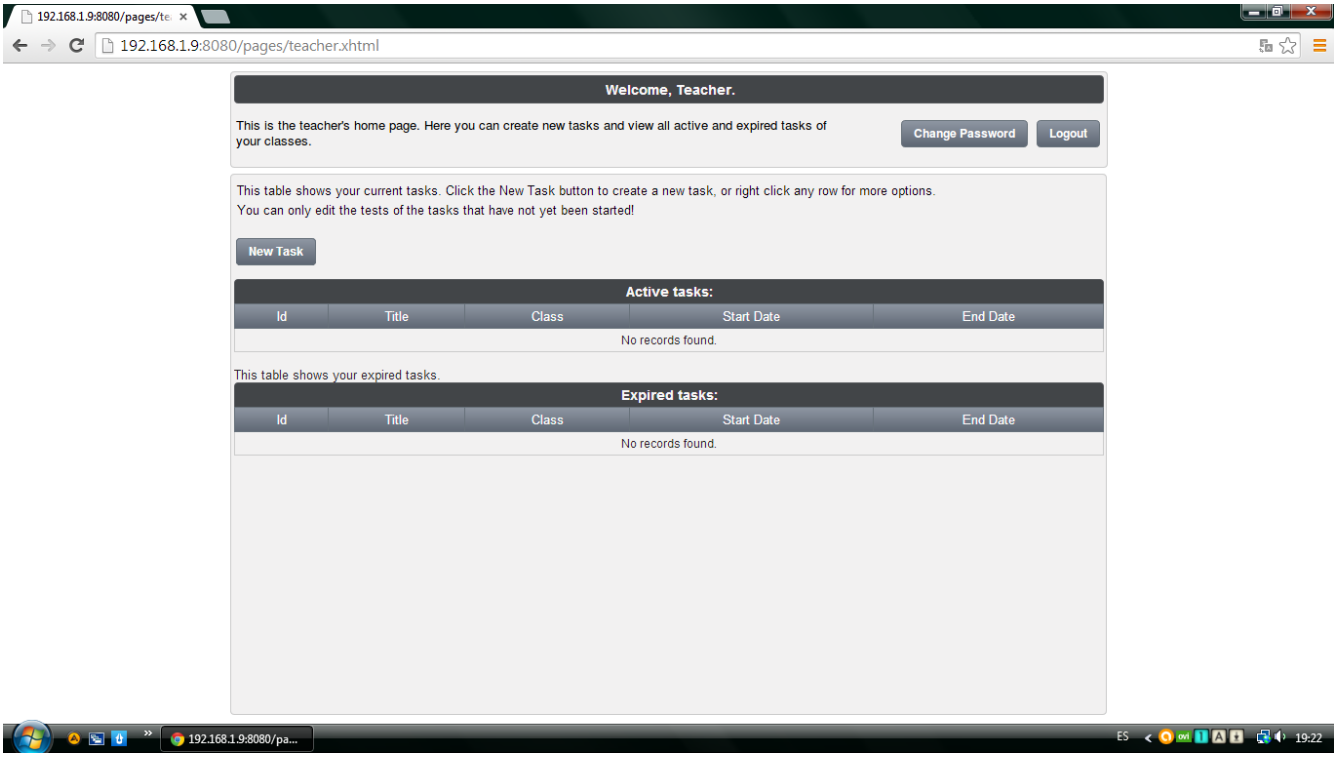

Fig 73: Vista principal del profesor vista desde Chrome

5.1.3 Internet Explorer 8

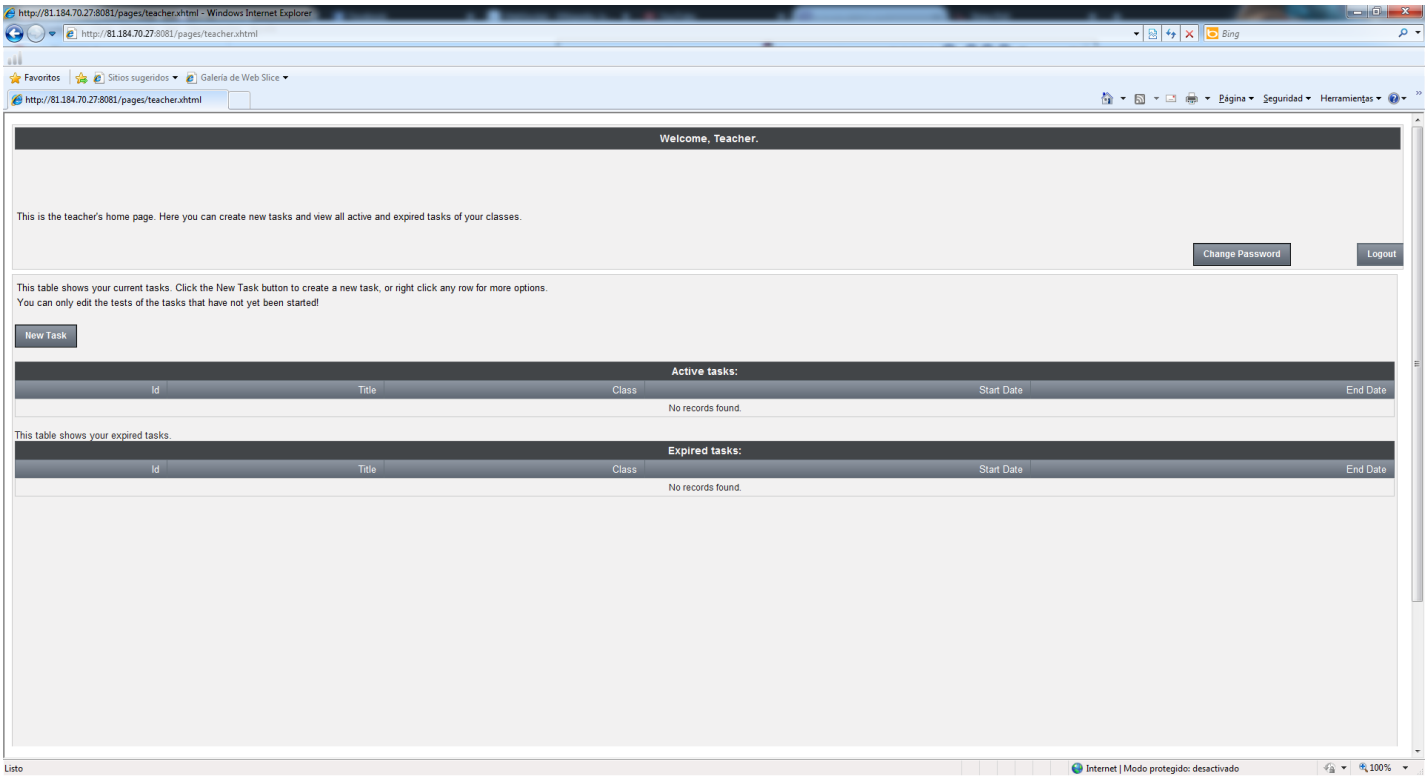

Fig 74: Vista principal del profesor vista desde Internet Explorer 8

5.2 Tests sobre la aplicación

En este apartado revisaremos todos los tests realizados para comprobar el funcionamiento correcto de la página web. Aquí detallaremos cada uno de los pasos y los resultados obtenidos.

Login

Si entramos unos datos incorrectos, la página se actualiza, sin mostrar ningún tipo de error, pero tampoco dejándote entrar, dando a entender que algo está mal, en este caso, la contraseña o el nombre de usuario.

Si introducimos unos datos correctos, como por ejemplo admin/1234 entraremos a la página correspondiente según el rol del usuario, en este caso, administrador.

5.2.1 Tests de administradores

Gestión de administradores

Seleccionamos la opción de crear nuevo administrador y, al introducir algún valor no válido, nos indica qué es lo que hay mal sin cerrar el diálogo de creación de administrador. Tras rellenar un nombre de usuario, nombre, apellidos y contraseña y damos a aceptar, se actualiza automáticamente la lista de administradores en la página principal.

Al hacer clic derecho sobre un administrador, nos aparecen distintas opciones adicionales: Ver, modificar y eliminar.

Al seleccionar “Ver administrador” se nos mostrará los datos entrados previamente (en ningún momento podremos ver la contraseña).

Modificar los datos de un administrador también funciona correctamente, aunque no se puede cambiar el nombre de usuario.

Al eliminar un administrador nos aparece un mensaje de confirmación, y si intentamos eliminar a nosotros mismos obtendremos el mensaje “Permission denied”.

Gestión de asignaturas

Seleccionamos la opción de crear nueva asignatura, y nos aparece un diálogo en el que únicamente podemos escoger el nombre de la asignatura. El nombre de la asignatura debe ser de 3-255 caracteres, en caso contrario nos aparecerá un mensaje similar al de la creación de administradores, y nos impedirá seguir adelante hasta que no sea un nombre correcto. Finalmente, al introducir un nombre correcto, la lista de asignaturas se actualizará con la nueva asignatura creada recientemente.

Al hacer clic derecho sobre una asignatura, nos aparecen distintas opciones adicionales: Ver, modificar participantes y eliminar.

Al hacer clic sobre “Ver asignatura” nos aparecerá un diálogo con un par de listados, uno para los profesores y otro para los alumnos. En estas listas, al hacer clic sobre un usuario o administrador no ocurrirá nada, simplemente es para ver quien forma parte de la asignatura.

En la opción para modificar participantes nos aparece la opción de cambiar el nombre de la asignatura, y una lista con todos los usuarios y profesores de la asignatura, pero en este caso si que podemos hacer clic derecho para ver al usuario en detalle, o para removerlo de esta asignatura.

En esta vista además, podemos añadir nuevos usuarios mediante los botones “Seleccionar existentes”, “Importar de fichero”, “Nuevo usuario”.

Al hacer clic en “Seleccionar existentes” nos aparece una lista con todos los alumnos y profesores, de modo que los que ya participaban en la clase están preseleccionados y podemos seleccionar o des-seleccionar más usuarios. Al acabar, si pulsamos “aceptar” volveremos a la vista anterior, y veremos que la lista con los participantes en la asignatura se ha actualizado.

Si seleccionamos “Importar de fichero” nos aparece un nuevo diálogo en el que se nos precisa de un fichero para importar usuarios. Al entrar “alumnes.csv” podemos ver que se ha subido correctamente.

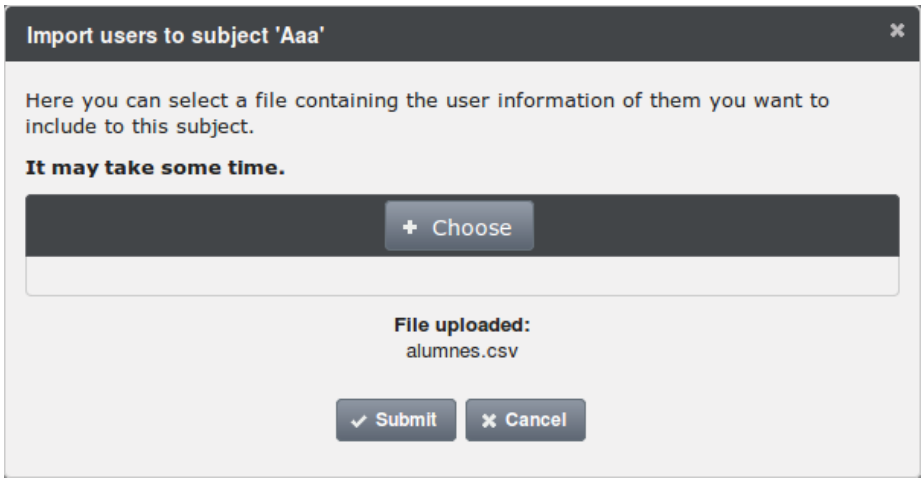

Fig 75: Diálogo para importar alumnos desde fichero

Al seleccionar “submit” se nos crearan aquellos usuarios que no existían en la base de datos ofreciéndoles la contraseña niub<NIUB>2014, y además se actualizarán a la lista de participantes de la asignatura.

Fig 76: Contraseña para los nuevos usuarios importados desde fichero

Finalmente, si seleccionamos la opción de “Nuevo usuario” podremos introducir los datos del nuevo usuario manualmente, parecido a como los hemos hecho con el administrador y al finalizar se añadirá a la lista también.

Al hacer clic derecho en la lista, y seleccionar “Remover de la asignatura” se eliminará de esta lista de participantes, y al hacer clic en “Update” los cambios se realizarán finalmente, y la asignatura tendrá los usuarios configurados. Si hacemos clic en “Cancel” únicamente se aplicarán los cambios del nombre de la asignatura.

Gestión de usuarios y profesores

Accediendo a la pestaña de profesores y usuarios, observaremos una lista extensa con todos los miembros de la base de datos, además, tenemos un filtro por si deseamos buscar a alguien en concreto.

Disponemos de dos botones “importar de fichero” y “nuevo usuario” que funcionan exactamente igual que al añadir usuarios para una asignatura en concreto, simplemente que ahora no se añadirán a ninguna asignatura, sino que se actualizará la lista de usuarios y se añadirán automáticamente a la base de datos.

Al hacer clic derecho sobre un usuario nos aparecerán las opciones de “ver usuario”, “editar usuario” y “eliminar”.

Si seleccionamos la opción para ver usuario, veremos su nombre de usuario, su nombre, apellidos y esas asignaturas en las que está apuntado. En esta vista no podemos realizar ningún cambio.

Si editamos un usuario podremos modificar todos sus campos excepto su nombre de usuario y su rol de alumno o profesor. Su contraseña, aunque la podremos modificar, no la podremos ver.

Finalmente, al eliminar un usuario, este desaparecerá de la base de datos. Antes de aceptar, nos aparece un mensaje avisándonos que el usuario ya no formará parte de ninguna asignatura y que todas sus tareas y entregas también se eliminarán.

Limpiar base de datos

Hacemos clic sobre el botón para limpiar la base de datos y nos aparece una confirmación y se nos actualiza la página donde vemos la lista de los administradores, y todas las demás vacías.

5.2.2 Tests de profesores

Para las pruebas de los profesores, crearemos distintos tipos de tareas para comprobar su funcionamiento.

Crear tarea en Java sin makefile con código de profesor

Damos al botón de crear tarea y se nos pide información básica sobre la tarea. Nombre, a que asignatura pertenece, fecha de inicio y fin y una descripción.

Creamos una tarea llamada TareaJ_CNM, con fecha de inicio de hoy, y fecha de finalización 30 de junio. Al hacer clic en siguiente vamos a otro apartado donde se nos pide el lenguaje, así que seleccionamos Java y vamos al siguiente paso.

Ahora aparece la configuración de compilación para nuestro código.

Dejamos desmarcada la opción de makefile, no añadimos ningún argumento de compilación, y no permitimos ningún fichero pdf, texto, vídeo o audio para el alumno.

En la siguiente ventana, se nos pregunta que deseamos hacer: introducir el código en la misma web, subir múltiples ficheros, o no subir nada.

Vamos a usar la primera opción. El fichero se llamará Test.java y el código introducido:

```
public class Test {
        public static void main(String[] args){
                System.out.println("Hello World!");
        }
}
```

Al escribir texto, se nos ha activado el botón “Check code” que inicialmente estaba desactivado. Al acabar de escribir ese código, lo pulsamos y nos aparecerá un aviso si se ha podido compilar correctamente o no.

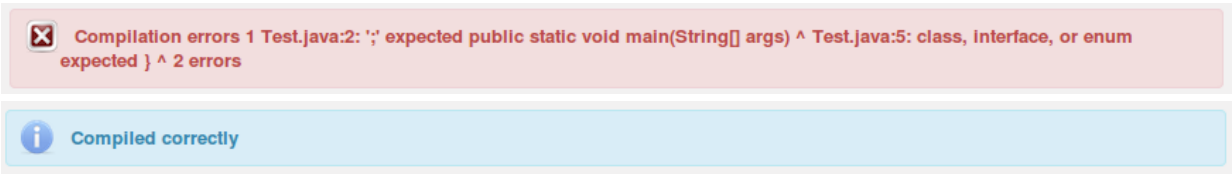

Fig 77: Mensajes sobre la compilación indicando si se ha encontrado un error

Una vez compilado correctamente, se activa el botón “Submit”.

Automáticamente nos aparece una página para programar los tests. Seleccionamos la opción de argumentos de ejecución, en este caso “0. Without Arguments” y hacemos clic en “Run Test”. Y automáticamente nos aparecerá el output de nuestro código tal y como vemos en la Fig. 78.

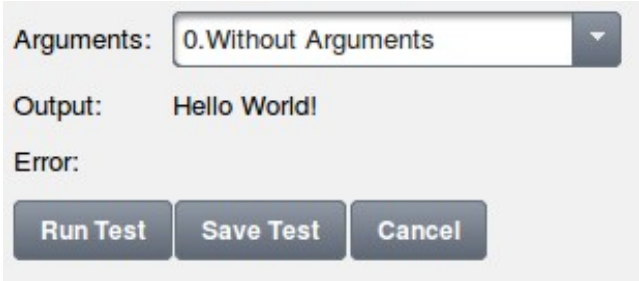

Fig 78: Vista del test ejecutado, listo para guardar

Al presionar "Save Test" el test quedará registrado en una lista que hay justo a continuación, así, podremos crear más tests si quisiéramos añadirlos. En nuestro caso, hemos acabado y presionamos "Go back" para volver a la página principal del profesor.

Crear tarea en C con makefile y sin código del profesor

Creamos la tarea siguiendo los pasos mencionados anteriormente. Ahora nuestra tarea se llamará TareaC_NCM, seleccionaremos C como lenguaje, y marcaremos la casilla makefile y la configuramos de la siguiente manera.

En el siguiente paso, seleccionaremos "Don't upload code", pero igualmente se nos pide un fichero makefile. Así que subimos uno con el siguiente contenido:

```
a:Test.c
	echo "No hago nada"
b:Test.c
	gcc -o Test Test.c
```

Y hacemos clic a "submit", que nos llevará otra vez a la página de tests.

Como antes, configuramos un test sin argumentos, pero ahora no nos mostrará ningún output, ya que no hemos subido ningún fichero. Simplemente guardamos un test sin argumentos a la lista, y volvemos a la página principal.

Crear tarea en Python sin makefile y sin código del profesor

Creamos otra nueva tarea, ahora llamada TareaP_NCNM, en lenguaje python. No podemos seleccionar makefile ni añadir argumentos de compilación, únicamente que tipo de ficheros sean aceptados por parte del alumno.

Seguimos al último paso donde deberemos subministrar el código, y seleccionamos la opción "Don't upload code" donde nos deja continuar a la fase de test subministrando un nombre de fichero, aquel que el usuario deberá entregar y el que se usará para ejecutar. Entramos Test.py y volvemos a introducir un test sin argumentos.

5.2.2 Tests de alumnos

La única función a realizar por los alumnos es la de subministrar sus códigos de las distintas tareas creadas por los profesores. Como previamente hemos creado 3 tareas de prueba, vamos a testearlas por parte del alumno.

Entregar tarea en Java configurada sin makefile y con código de profesor

En la página principal del alumno podemos ver todas esas tareas que se han creado en una asignatura donde estamos apuntados.

Si estas tareas se encuentran en la tabla de "tareas activas", podremos hacer clic derecho, y seleccionar "Upload Code" para entregar nuestro código.

Empezaremos por la tarea "TareaJ_CNM" creada anteriormente.

Al entrar nos notifica con un mensaje: *"A file called 'Test.java' will be compiled, be sure you're uploading a file with this name."* para que, al subir múltiples ficheros nos aseguramos que haya uno con ese nombre.

Subiremos un fichero Test.java con el mismo contenido que el código subministrado por el profesor:

```
public class Test {
	public static void main(String[] args){
		System.out.println("Hello World!");
	}
}
```

Y al hacer clic en "Run Test" nos aparece el siguiente registro dándonos a entender que nuestro código es correcto.

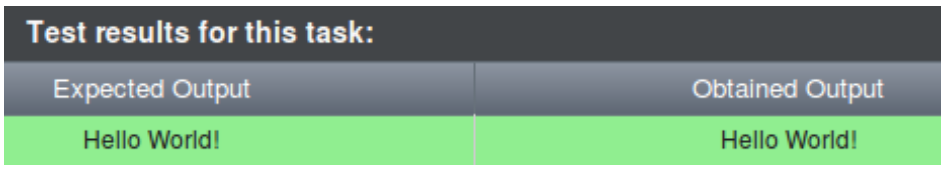

Fig 79: Test realizado con resultado satisfactorio

Si entramos un código incorrecto, el color de fondo se volverá rojo.

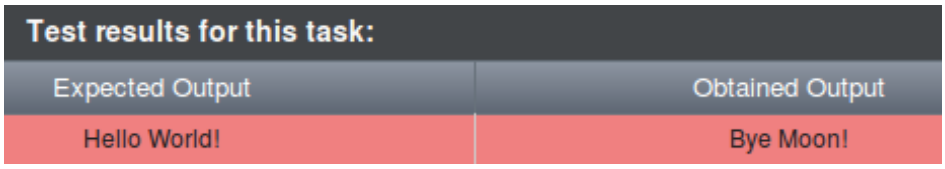

Fig 80: Test realizado con resultado erróneo

Una vez finalizada nuestra entrega, podemos hacer clic en "Save" para guardarla en la base de datos.

Entregar tarea en C configurada con makefile y sin código de profesor

Igual que antes, seleccionamos la tarea "TareaC_NCM" y en este caso nos advierte de que *"A 'makefile' will be used to compile the code. You don't have to upload this 'makefile' but you'll have to talk to your teacher to know the filenames you'll be using."* para que nos informemos por parte del profesor qué nombre ha de tener el fichero a compilar. En nuestro caso, el código que se compilaba era el contenido en Test.c, por lo que subiremos un fichero "Hello World" en C con ese nombre.

Al hacer clic en "Run Test" nos aparece el registro de la Fig. 81. En este caso no se nos muestra si es correcto o incorrecto, ya que el profesor no había subministrado ningún código, por lo que no se puede comparar con nada.

Test results for this task:		
Id	Arguments	Obtained Output
2		Hello World

Fig 81: Test realizado y guardado, sin conocer el valor correcto del código del profesor

Entregar tarea en Python configurada sin makefile y sin código de profesor

El funcionamiento exactamente igual al anterior, ahora subministraremos el código para "TareaP_NCNM" y, ahora no seleccionaremos "Upload files", sino que entraremos el código manualmente.

Si escribimos print "Hello World" veríamos una salida igual al de la Fig. 81.

Todas estas pruebas mostradas han sido únicamente usando un test, pero con múltiples tests el resultado también es satisfactorio

Test results for this task:			
Id	Arguments	Expected Output	Obtained Output
7		No arguments	No arguments
5	22 11 6	3 Arguments: '22' '11' '6' ((atoi sum 39))	3 Arguments: '22' '11' '6'
6	12 21 12 21	4 Arguments: '12' '21' '12' '21' ((atoi sum 66))	4 Arguments: '12' '21' '12' '21'
4	11	1 Argument: '11'	1 Argument: '11'

Fig 82: Código del alumno comparado con múltiples tests

6. Conclusiones

6.1 Discusión

Al inicio del proyecto se presentó como una aplicación que permitía corregir el código de los alumnos mediante una página web, sin embargo, está web tenía algunas deficiencias y limitaciones. El objetivo de este proyecto era analizar el funcionamiento y herramientas disponibles en la aplicación, ampliarlas y depurarlas para una mayor comodidad de uso y prestaciones, además de corregir errores que pudiera tener.

Los bloques principales de este proyecto han sido:

- Mejorar interfaz. Se intentó actualizar de la versión 3.0.1 de PrimeFaces a la versión 4.0. Sin embargo, está última daba problemas con Internet Explorer 8 (no se han testeado otras versiones), así que finalmente se actualizó a la versión 3.5.
- Dar más control al administrador para crear usuarios y añadirlos a asignaturas existentes.
- Modificar la base de datos. Antes el código de los programas se guardaban en la propia base de datos en un registro del tipo “TEXT”. La base de datos se ha modificado para aceptar soporte a múltiples ficheros, y sin contener este tipo de registros. Ahora los ficheros se leerán de la base de datos.
- Ofrecer nuevas herramientas para la creación de tareas: Múltiples ficheros, argumentos de compilación, compilación vía makefile, ficheros compartidos con alumnos y limitación de tipos de ficheros por parte de los alumnos.
- Descarga de las entregas de los alumnos en formato .zip por parte del profesor
- Ampliar la seguridad de aplicaciones Java limitando sus funciones.

La realización de este proyecto ha cumplido con éxito cada uno de los objetivos establecidos, y se ha comprobado la correcta funcionalidad de éstos mediante las pruebas realizadas.

Durante la elaboración del proyecto y el estudio del código inicial de los proyectos anteriores, se encontraron una serie de errores que comentaremos a continuación y que se han acabado solucionando.

El primer error que nos encontramos fue en la página del administrador admin.xhtml. Esta página estaba llena de referencias a id's no validas por PrimeFaces, lo cual provocaba que algunos contenidos de esta no se actualizaran cuando debían. Está página fue rediseñada completamente, ya que se cambió su estructura y, de esta manera, se solucionaron todos los problemas. La otra página que tenía problemas era la de adición de tests que, por un motivo en el código del servidor, no se añadían correctamente. Curiosamente todas las demás páginas estaban escritas perfectamente sin ningún tipo de error.

Otro problema era al momento de compilar. Aunque en la aplicación web había una opción para argumentos de compilación (-lm o -O3, por ejemplo), SecuritySCP no usaba ese contenido y esos argumentos no se realizaban. Se ha arreglado todo lo necesario para hacerlo funcional.

También habían problemas en los argumentos de ejecución, ya que si introducías los siguientes argumentos: `1 2 3`. SecuritySCP los ejecutaba como un único elemento "`1 2 3`". No había modo de enviar más de un argumento. Este aspecto también se ha solucionado y si definimos los argumentos: `1 "2 3"` se ejecutará con los dos argumentos: "`1`" y "`2 3`".

A grandes rasgos estos han sido los errores encontrados más importantes que hacían imposible la correcta ejecución de la aplicación.

6.2 Futuras mejoras

Aunque se han solucionado y mejorado muchos aspectos del SCP, todavía hay muchas mejoras que no se han podido efectuar. A continuación citaremos alguna de ellas.

- Ampliar la aplicación a nuevos lenguajes como Matlab
- Añadir restricciones de llamadas al ejecutar programas Python como ya se hace actualmente con C usando "nurse" o con Java y las "políticas".
- Comprobar el tiempo de ejecución de los programas, y si excede un límite, matarlos y marcar el test como inválido.
- Ampliar el tipo de salida como resultado de ejecución de un programa, pudiendo, por ejemplo, crear una fichero de imagen como resultado.
- Limpieza de las clases y métodos no utilizados en SCP4.0 proveniente de las antiguas versiones.
- Usar algún método de seguridad para transportar los datos en Internet de forma segura, tipo SSL, RSA, CBC, etc.
- Control de copias. El sistema debería detectar de forma automática si dos o más alumnos tienen practicas similares.
- Evitar poder añadir o modificar los tests de las tareas activas, ya que los alumnos que ya hayan subido su código no tendrán resultados para esos tests.
- Poder fijar un número de intentos que los alumnos tienen para subir un código. Actualmente es ilimitado.
- Soporte para carpetas en las entregas de los alumnos. Por ejemplo, poder subir un fichero zip que contenga el código y una carpeta con librerías.

7. Bibliografía

Documentación PrimeFaces y JSF

http://www.primefaces.org/showcase/

http://www.primefaces.org/themes

http://www.primefaces.org/primeui/demo.html

Documentación sobre políticas java

http://www.ibm.com/support/knowledgecenter/SSGMGV_3.2.0/com.ibm.cics.ts.doc/dfhtk/topics/dfhtk_reqs_jav2sec.html

https://publib.boulder.ibm.com/infocenter/iseries/v5r3/index.jsp?topic=%2Frzamy%2F50%2Fsec%2Fsecj2syn.html

Bussiness Objects & Data Access Objects

http://www.developer.am/documentation/jsf/?page=jsf-2-0-spring-hibernate-integration-example

http://en.wikipedia.org/wiki/Business_Objects

http://en.wikipedia.org/wiki/Data_access_object

Herramientas

Esquemas y diagramas (umlet): http://www.umlet.com/

Iconos free license: https://www.iconfinder.com

Adobe Photoshop CS5: http://www.adobe.com/products/photoshop.html

IDE Netbeans: https://netbeans.org/downloads/

Apache Tomcat: http://tomcat.apache.org/download-70.cgi

Ubuntu: http://www.ubuntu.com/download/desktop

MySQL: http://dev.mysql.com/

Otros

http://stackoverflow.com/

Anexo A: Manual instalación

La aplicación se ha de ejecutar en un sistema Linux. A continuación se detallará como ejecutar el servidor correctamente.

A.1 MySQL

Paso 1: Instalación

Si ya tenemos instalado MySQL en nuestro sistema, podemos saltarnos este paso. Para la instalación de la base de datos MySQL abrimos el terminal de Linux. y entramos el siguiente comando:

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

A continuación se nos pedirá la contraseña de nuestro usuario de Ubuntu, y una confirmación para empezar la descarga y la instalación.

Durante la instalación deberemos entrar una contraseña para el usuario root con más privilegios en la base de datos. La contraseña debe ser cualquiera, pero deberemos recordarla. Tras escribir la contraseña, pulsamos “enter” para continuar.

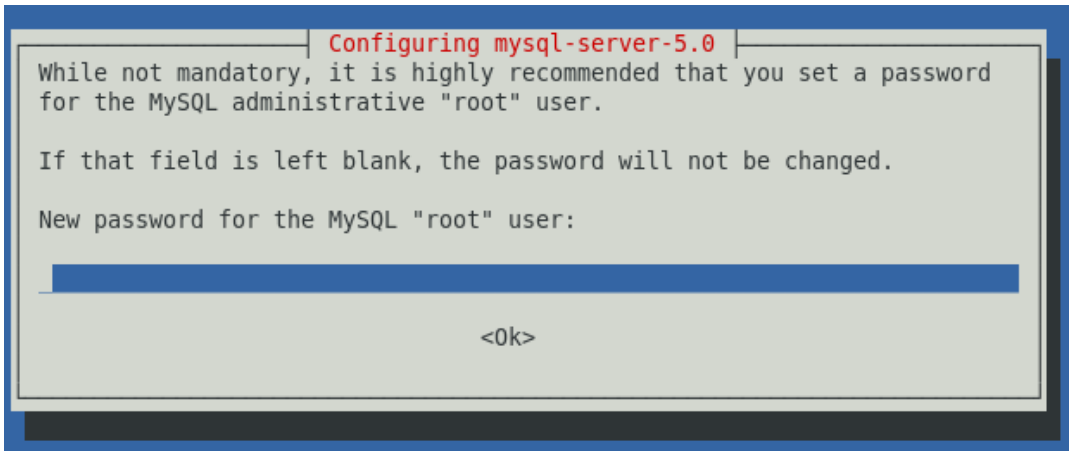

Fig 83: Ventana para asignar una contraseña al usuario root de MySQL

Paso 2: Accediendo a la Base de Datos MySQL como root

Al instalar la base de datos, podremos acceder a ella como usuario root (con todos los privilegios) con el siguiente comando:

```
mysql -u root -p
```

A continuación se nos pedirá la contraseña del usuario root definida en el paso anterior, y una vez dentro podremos ejecutar comandos SQL o funciones MySQL.

Paso 3: Preparando la base de datos para ser usado por SCP4.0

Nuestro cliente SCP4.0 se conectará a la base de datos MySQL, y debemos crear un usuario con los privilegios necesarios para modificar sus datos. Para ello, debemos situarnos en la carpeta donde se encuentra el fichero "create_db.sql", entrar en la base de datos como está explicado en el Paso 2 y entrar las siguientes lineas para crear el usuario.

```
create user 'user'@'localhost' identified by 'useruseruser';
grant all privileges on scp2.* to 'user'@'localhost';
```

Una vez creado el usuario, debemos crear la base de datos con la estructura de tablas correspondientes para guardar los datos. Para ello usaremos el fichero "create_db.sql" subministrado, y para cargar el fichero deberemos escribir dentro del MySQL:

```
source create_db.sql;
```

Importante: El fichero debe estar en ese mismo directorio donde hemos abierto MySQL siguiendo las instrucciones del Paso 2. Si no es el caso, podremos cargarlo igualmente introduciendo la ruta completa del fichero.

Paso 4: Salir de MySQL

Dentro de MySQL, basta con escribir **quit** o **exit**.

A.2 SCP4.0

Para iniciar nuestra aplicación web debemos abrir el terminal e dirigirnos a la carpeta "CLIENTE WEB", donde se encuentra el fichero "startup.sh". Y escribir el comando:

```
sudo ./startup.sh
```

Se nos pedirá la contraseña del usuario de Linux, y una vez ejecutado, podremos cerrar el terminal y ya tendremos la aplicación ejecutándose.

Para apagar SCP4.0 deberemos ejecutar "shutdown.sh" de la misma manera:

```
sudo ./shutdown.sh
```

A.3 SecuritySCP

Para iniciar SecuritySCP deberemos abrir el terminal, y ahora dirigirnos a la carpeta "SECURITYSCP", donde encontraremos otro fichero "startup.sh". De igual manera, deberemos ejecutar:

```
sudo ./startup.sh
```

Ahora NO debemos cerrar la ventana del terminal, si lo hacemos interrumpiremos la ejecución de SecuritySCP.

A.4 Navegador

Una vez SCP4.0 y SecuritySCP estén ejecutándose, podremos acceder a la aplicación usando cualquier navegador de internet, y dirigiéndonos a

http://localhost:8080/SCP4.0

Anexo B: Manuales de uso

B.1 Admininstrador

Al entrar en la aplicación como administrador. Tendremos 4 pestañas que deberemos seleccionar según los datos que queremos manejar:

- ADMINS: Podremos gestionar los demás administradores así como crear más.
- Subjects: Podremos gestionar asignaturas y añadir nuevos alumnos/profesores en ellas.
- Students and teachers: Podremos gestionar los usuarios existentes, modificando sus datos personales, eliminarlos o incluso crear más.
- DDBB: Aquí únicamente tendremos la opción de vaciar la base de datos dejando únicamente los administradores.

Gestionar administradores

- **Crear administrador**: Para ello debemos dirigirnos a la pestaña "ADMINS", seleccionar "New Admin". A continuación solo deberemos rellenar todos los campos necesarios.
- **Modificar datos de un administrador**: En el listado de todos los administradores en la pestaña "ADMINS" seleccionamos el que queremos modificar con el clic derecho y seleccionamos "Edit Admin". Al realizar los cambios deseados, aceptamos para guardarlos.
- **Eliminar administrador**: En el listado de todos los administradores en la pestaña "ADMINS" seleccionamos el que queremos eliminar con el clic derecho y seleccionamos "Delete".

Gestionar asignaturas

- **Crear asignatura**: Nos dirigimos a la pestaña "Subjects" y seleccionamos "New Class". A continuación sólo debemos introducir el nombre de la asignatura.
- **Modificar nombre de la asignatura**: En el listado de asignaturas en la pestaña "Subjects" seleccionamos el que queremos modificar con el clic derecho y seleccionamos "Edit Participants". En la parte superior de la ventana podremos modificar el nombre.
- **Añadir alumnos o profesores a una asignatura:** En el listado de asignaturas en la pestaña "Subjects" seleccionamos el que queremos añadir usuarios con el clic derecho y seleccionamos "Edit Participants". En la ventana aparecerán distintas opciones para añadir usuarios en la parte inferior izquierda. Podremos crear un nuevo usuario manualmente y añadirlo a la base de datos, seleccionar un usuario existente, o importar un fichero .csv con los usuarios de la asignatura.

- **Eliminar alumnos o profesores a una asignatura:** En el listado de asignaturas en la pestaña "Subjects" seleccionamos el que queremos eliminar usuarios con el clic derecho y seleccionamos "Edit Participants". En la ventana aparecerá un listado con todos los usuarios de la asignatura. Para expulsar uno de la asignatura, le hacemos clic derecho y seleccionamos "Remove User From Class".
- **Eliminar asignatura:** En el listado asignaturas en la pestaña "Subjects" seleccionamos el que queremos eliminar con el clic derecho y seleccionamos "Delete".

Gestionar alumnos y profesores

- **Crear nuevo usuario profesor o alumno**: Nos dirigimos a la pestaña "Students and Teachers" y seleccionamos "Add User". A continuación introducimos todos los datos solicitados para crear el usuario.
- **Importar usuarios de un fichero:** Nos dirigimos a la pestaña "Students and Teachers" y seleccionamos "Import from file". A continuación seleccionamos el fichero .csv donde encontraremos la información de los usuarios para crearlos. Los usuarios ya existentes en la base de datos no se crearan, para todos los demás sus contraseñas iniciales por defecto serán "niub<Nº NIUB>2014".
- **Modificar datos de un usuario**: En el listado de todos los usuarios en la pestaña "Students and Teachers" seleccionamos el que queremos modificar con el clic derecho y seleccionamos "Edit user". Al realizar los cambios deseados, aceptamos para guardarlos.
- **Eliminar usuario**: En el listado de usuarios en la pestaña "Students and Teachers" seleccionamos el que queremos eliminar con el clic derecho y seleccionamos "Delete".

B.2 Profesor

Al entrar como profesor veremos dos listas y un botón. El botón inicia el proceso de creación de una tarea, en la primera lista aparecerán aquellas tareas que aún no han expirado, y en la segunda lista las finalizadas.

Crear tarea

En la página principal del profesor, pulsar el botón "New Task". A continuación rellenamos los campos de la pestaña "General" sobre la configuración de la tarea (el campo de descripción no es necesario) y presionamos "Next".

Ahora debemos seleccionar el lenguaje del código. En esta caso C, Java o Python y pulsamos "Next".

Opciones de compilación: En este apartado configuramos CÓMO será compilado, es decir, con que argumentos. No confundir con los argumentos de ejecución una vez compilado. También tenemos la opción de usar un fichero makefile, el cual también tendremos que configurar sus argumentos e incluso indicar cual es el nombre del fichero ejecutable que creará.

Toda la configuración para la compilación no será visible si el lenguaje seleccionado fue Python.

Además, también podremos elegir qué tipo de ficheros podrán subir los alumnos.

Al finalizar la configuración presionamos "Next" para subir el código.

Subida de código: Tenemos distintos métodos para subir código que veremos a continuación:

- **Input Raw Code:** Nos permite introducir el código de un único fichero que usaremos para la tarea. También deberemos seleccionar el nombre del fichero. Esta opción no se encuentra disponible si hemos seleccionado usar makefile, ya que, en este caso, se deberían subir por lo menos dos ficheros: el código y el fichero makefile.
- **Upload Files**: Nos permite subir múltiples ficheros de cualquier tipo (excepto ejecutables) a nuestra tarea. En caso de haber seleccionado usar makefile, deberemos subirlo junto a todo el código.

  Finalmente, deberemos indicar qué fichero de todos aquellos subidos, será el que contiene la función main y será el ejecutable al compilarse.
- **Don't upload code:** Si queremos no subir ningún código para que los alumnos no tengan un valor de referencia esta es la opción. En caso de no querer subir un código, deberemos indicar, igualmente, un nombre de fichero, que será el que el alumno deberá entregar, y el que contendrá la función main del código.

  En caso de haber seleccionado usar makefile, deberemos subirlo en este apartado antes de continuar.

Crear tests

Tras crear una tarea podremos añadirle tests para comparar los resultados de los alumnos con los nuestros.

Para ello, tenemos que indicar que tipo de test crear: Sin argumentos o con argumentos. Estos argumentos se enviarán directamente al ejecutar el código una vez compilado.

Si seleccionamos ejecutar con argumentos deberemos indicar qué argumentos exactamente.

Si el profesor ha subido un código, deberá ejecutar el test contra su código. Por ello, hacemos clic en "Run tests" y veremos el resultado obtenido. Si es el resultado deseado, pulsamos el botón "Save Test" para añadirlo a la base de datos y visualizarlo en la lista de tests. Los alumnos podrán ver los resultados de nuestros tests tras entregar ellos su solución.

Si el profesor no ha subido un código, simplemente pulsaremos “Save test” tras seleccionar unos argumentos para guardarlos a la base de datos.

Editar tests

En la página principal del profesor, seleccionar una tarea de la lista “Active tasks” con el clic derecho, y elegimos “Edit Tests”.

Eliminar tests

En la página de los tests, seleccionar con el clic derecho el test que deseamos eliminar de la lista y elegimos “delete”.

Eliminar tarea

En la página principal del profesor, seleccionar con el clic derecho una tarea de cualquiera de las dos listas, y elegimos “Delete”.

Ver resultados de los alumnos y/o descargar su código

En la página principal del profesor, seleccionar con el clic derecho una tarea de cualquiera de las dos listas, y elegimos “View Report”.

B.3 Alumno

Al entrar como alumno veremos dos listas. La primera lista aparecerán aquellas tareas activas que debemos entregar código, y en la segunda lista las finalizadas.

Subir código

Como alumnos, esta es la única función que podemos realizar. Para ello, seleccionamos la tarea de la primera lista con el clic derecho y pulsamos “Upload Code”.

En la siguiente ventana veremos dos modos de poder subir el código:

- Input Raw Code: Introduciremos el código de un solo fichero por lo que, si nuestro código consta de más de uno, esta no es nuestra opción.
- Upload files: Aquí podremos seleccionar múltiples ficheros para nuestra entrega.

Es recomendable hablar con el profesor para descubrir cómo se compilará el código y cómo se han de llamar los ficheros para que funcione correctamente.

En cualquiera de los dos casos, al finalizar pulsamos “Run tests” para revisar el resultado obtenido, y si es el deseado, pulsaremos “Save” guardar nuestra entrega en la base de datos y volver a la página principal del alumno.