Proyecto de fin de carrera

Instituto de Ingeniería Eléctrica
Facultad de Ingeniería
Universidad de la República
Uruguay

WEBIDE

Aplicación web y objeto de aprendizaje para enseñanza de un lenguaje de programación.

Matías Tolosa
Marcos Nicolás García
Vittorio Dotti Ricagni

Tutor: Víctor González Barbone

Abril 2013

Agradecimientos

Agradecemos a todas las personas que colaboraron de alguna forma en la realización del Proyecto WebIDE, directa o indirectamente.

Muy especialmente al tutor del proyecto Ing. Víctor González Barbone, por su cercanía y su guía a lo largo de todo el año.

También especialmente a cada una de las familias de los integrantes del proyecto, ya que si no fuera por el apoyo brindado, hubiera sido difícil llevar a cabo el mismo.

En particular nos gustaría agradecer a Fernando y Anahí, Alfonso, Ana, Omar y Andrea. A Lorena por su colaboración. A Romina por sus opiniones, su colaboración y su apoyo, sin lugar a dudas, incondicional.

Contenido

Figuras

Tablas

I. Introducción

En el presente capítulo se verá cuál es la motivación para realizar este proyecto, los objetivos, las funcionalidades que se espera lograr y los criterios de éxito que permitirán evaluar, una vez finalizado el mismo, si se han cumplido o no con los objetivos propuestos.

I.1. Motivación

La enseñanza de un lenguaje de programación es especialmente apta para educación a distancia y/o asistida por computador. La posibilidad de disponer de una aplicación web para escribir código, compilar y ensayar pequeños programas permitirá a los estudiantes disponer de un ambiente de desarrollo elemental pero fácilmente accesible, sin requerir ninguna instalación en la máquina local. Esto es particularmente importante para una primera aproximación a la construcción de programas, o para dirigirse a un público objetivo no especializado, como los maestros del Plan Ceibal. Si se agrega la posibilidad de correr pruebas de unidad ("unit testing") en base a frameworks conocidos como PyUnit o DocTest puede lograrse una forma de realimentación al estudiante y de corrección automática. La integración de los recursos necesarios en un objeto de aprendizaje permite colocar estas facilidades en una plataforma de aprendizaje (e.g. Moodle), proponer tareas de programación, proveer realimentación al estudiante, calificar automáticamente en base a las pruebas unitarias (unit testing), y colectar las calificaciones obtenidas por el estudiante en su registro personal. Esta facilidad permite no sólo la educación a distancia, sino también la autoformación y una sensible ampliación de cupos en los cursos presenciales y semi-presenciales, sin requerir mayores recursos.

I.2. Objetivos

- Desarrollar una aplicación web para compilación, ejecución y ensayo (testing) de código fuente de un lenguaje de programación, para educación a distancia.

- Crear un objeto de aprendizaje en base a un protocolo estándar integrable a una plataforma de aprendizaje, que permita compilar, ejecutar y ensayar código fuente, obteniendo una calificación registrable en el historial del estudiante.

I.3. Funcionalidades esperadas

Se espera implementar las siguientes funcionalidades:

- Editor web para escribir código de programación.
- Soporte para lenguaje de programación Python.
- Compilación con informe de resultados.
- Ejecución con resultados en formato texto.

- Ejecución de pruebas unitarias e informe de resultados.
- Integración de los recursos necesarios en un objeto de aprendizaje, eventualmente comunicado con otros procesos locales o remotos para compilación, ejecución u otros, en forma transparente para el usuario.
- Intercambio de información con la plataforma Moodle para registro de calificaciones en la ficha del estudiante.

I.4. Criterios de éxito

- Lograr una aplicación web que permita compilar pequeños programas escritos en lenguaje Python y devuelva un informe de resultados.

- Crear un objeto de aprendizaje que permita la corrección y calificación automática de tareas de los estudiantes; el mismo será implementado en una plataforma de aprendizaje (por ejemplo Moodle).

II. Conceptos y tecnologías

"Todos los hombres tienen por naturaleza el deseo de saber", frase del Filosofo Aristóteles (Elcho Pan, 1988:45), en la cual enfatiza la curiosidad innata del hombre en la busca del conocimiento. En esta sección se describen algunos de los conceptos y herramientas en las que el proyecto Webide se basa para su desarrollo, y que el mundo actual utiliza activamente, estando muchas de ellas a la vanguardia. Como pensó Aristóteles, el hombre siempre está desarrollando nuevos objetivos, satisfaciendo su necesidad de conocimiento.

IDE (Integrated Development Environment – Entorno de Desarrollo Integrado): es un programa informático compuesto por un conjunto de herramientas de programación. Es un empaquetado, que consiste en un editor de código, un compilador, un depurador, y un constructor de interfaz gráfica (GIU). Los IDEs pueden ser aplicaciones por sí solas, o pueden ser parte de aplicaciones existentes.

Moodle: Aplicación web de tipo Ambiente Educativo Virtual, sistema de gestión de cursos, de distribución libre, que ayuda a los educadores a crear comunidades de aprendizaje en línea. Es un tipo de plataforma de aprendizaje o LMS (Learning Management System – Sistema de Gestión del Aprendizaje). Moodle fue creado por Martin Dougiamas, quien fue administrador de WebCT en la Universidad Tecnológica de Curtin. Basó su diseño en la idea que el conocimiento se construye en la mente del estudiante

Web2py: Es una plataforma libre de código abierto, para el ágil desarrollo de páginas web seguras sustentadas en base de datos. Está escrita en Python y es programable también en ese lenguaje. Esta plataforma es completa ya que posee todo lo necesario para construir páginas web completamente funcionales. Está basada en el Patrón Model View Controller (MVC), donde se separa la representación de Datos (Modelo), de la presentación de los datos (Vista), y el flujo de trabajo (Controlador).

Python: Es un lenguaje de programación interpretado (para ser ejecutado por un intérprete), multiparadigma. Soporta programación orientada a objetos, programación imperativa y en menor medida programación funcional.

Unittest/PyUnit: Es una familia de herramientas conocidas colectivamente como xUnix. Es un conjunto de frameworks basados en el software SUnit para Smalltalk.

Patrón MVC (Model View Controller – Modelo Vista Controlador): Es un patrón o modelo de abstracción de desarrollo de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de negocio en tres componentes distintos (Modelo, Vista y Controlador).

HTML (Hipertext Markup Language, Lenguaje de marcado de Hipertexto): Es un lenguaje de programación predominante en la elaboración de páginas web, utilizado para escribir, traducir la estructura, y la información en forma de texto, pudiendo complementar con imágenes. Se escribe en forma de etiquetas delimitadas por símbolos de mayor y menor, por ejemplo <head>.

JavaScript: Es un lenguaje de programación interpretado, dialecto del lenguaje ECMAScript. Se utiliza principalmente del lado del cliente, implementado como parte del navegador web, permitiendo mejoras en la interfaz de usuario y páginas web dinámicas.

Base de Datos: Es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso. Existen programas denominados sistemas gestores de base de datos que permiten almacenar y posteriormente acceder a los datos de forma rápida y estructurada. Ejemplos de gestores son MySQL, PostgreSQL y SQLite.

Objeto de Aprendizaje: Es cualquier recurso digital estructurado, que puede ser reutilizado, para lograr un objetivo de aprendizaje. Por ejemplo, un objeto de aprendizaje puede ser implementado en un paquete con estructura SCORM conteniendo recursos, como por ejemplo, imágenes, archivos PDF, enlaces a páginas web, y variadas funcionalidades, con el objetivo principal del aprendizaje.

SCORM (Sharable Content Object Reference Model – Modelo de Referencia de Objetos de Contenido Compartible): Es un conjunto de estándares y especificaciones que permiten crear un objeto de aprendizaje. SCORM hace posible la creación de un paquete para plataformas de aprendizaje (LMS) que permitan la importación de este contenido. Tiene la posibilidad de ser importado en variadas LMS (por ejemplo Moodle). Fue desarrollado por el Ministerio de Defensa de los Estados Unidos.

Reload Editor: Empaquetador de contenidos y editor de metadatos de código abierto, destinado a compartir material de enseñanza aprendizaje. Permite la creación de paquetes SCORM.

Common Cartridge: Es un conjunto de especificaciones para crear objetos de aprendizaje, desarrollado por IMS Global Consortium. Un paquete Common Cartridge tiene la posibilidad de ser implementado en distintas LMS, por ejemplo Moodle.

XAMPP: Es un servidor de plataforma, es de software libre y las principales características de este es que posee servidor Apache, maneja base de datos MySQL, y posee intérprete para PHP y Perl. El nombre proviene de las características de este software, como es “X” se refiere en el uso de cualquier sistema operativo, la “A” por su servidor apache, “M” ya que trabaja con base de datos MySQL, y las dos “PP” por los intérpretes de PHP y Perl. Está bajo la licencia de GNU y actúa como servidor libre. XAMPP viene para distintas versiones de Sistemas Operativos (SO), para GNU/Linux, Microsoft Windows, Solaris, Mac OS.

Mozilla Firefox: Es un navegador web libre, que se encuentra disponible para distintos Sistemas Operativos (SO), disponible en Microsoft Windows, GNU/Linux, Mac OS. Este Navegador es coordinado por la Fundación Mozilla y por la Corporación Mozilla, y es de código abierto. Firefox hoy en día está dentro de los tres navegadores más utilizados a nivel mundial.

Internet Explorer: Es un navegador web, desarrollado por Microsoft en 1995, anteriormente se conocía por Microsoft Internet Explorer. Es un navegador diseñado únicamente para el SO Microsoft Windows, para cualquiera de sus versiones. En los comienzos de año 2000 dominaba el mercado ampliamente, pero hoy en día no es así, teniendo una gran competencia como Google Chrome y Mozilla Firefox.

Google Chrome: Es un navegador web desarrollado por Google, disponible gratuitamente bajo condiciones de servicio específicas. Existen versiones para distintos sistemas operativos, para Microsoft Windows, GNU/Linux, y Mac OS. Al día de hoy se estima que es el navegador más utilizado a nivel mundial, pero dependiendo el origen de la medición puede aparecer en segundo lugar.

Lenguaje Unificado de Modelado (UML): Es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema

StarUML: Es una herramienta UML de código abierto, licenciado bajo una versión modificada de la GNU GPL. Permite la creación de diagrama de componentes, además de otros.

III. Estado del arte

El lenguaje de programación Python es cada vez más utilizado por los desarrolladores de software. Su sintaxis es limpia y está desarrollado para favorecer el código legible. Posee una amplia gama de posibilidades de utilización, ya que soporta programación orientada a objetos. Para programar en Python, es necesario un compilador que permita compilar y ejecutar código fuente. Es común tenerlo instalado en el propio computador, pero una facilidad para quienes comienzan con este lenguaje, y que brinda cierta independencia, es tener un compilador web, pudiendo realizar pruebas en línea, desde cualquier lugar. En Internet se pueden encontrar compiladores web para Python, existiendo varias opciones, entre ellas compiladores que están enfocados a otros lenguajes no solo a Python. Por ejemplo en la dirección http://codepad.org se encuentra un compilador Web, donde se pueden realizar pruebas y consultar código. Si bien soporta distintos tipos de lenguaje, como C, C++, PHP, Perl, etc., no tiene reconocimiento ni resaltado de sintaxis, y no informa la línea en la cual se está escribiendo el código.

Otro ejemplo de este tipo de compiladores web, es http://ideone.com/, que tiene también la opción de elegir variados lenguajes de programación. Tiene reconocimiento y resaltado de sintaxis, pero al realizar una nueva consulta, no mantiene el código antes ingresado.

En los casos antes mencionados, las interfaces en general y la información que se despliega para su utilización, se encuentran escritas en inglés.

Con respecto a métodos de corrección automática, si bien existen módulos con posibilidad de testear código en lenguaje Python, por ejemplo unittest, lo que permitiría comparar salidas dadas por un compilador con otras preestablecidas, no se han encontrado en la web aplicaciones que implementen una comparación de ese estilo, y que permitan una corrección automática de tareas realizadas por un grupo de estudiantes en el dictado de un curso, por ejemplo.

Existen extensiones para Moodle, que permiten ejercitar al estudiante, mediante casillas en blanco para rellenado. Al ingresar el estudiante a este cuestionario, se le presenta un conjunto de preguntas que debe completar, donde cada una puede tener varias respuestas correctas, por ejemplo segmentos de código de un lenguaje de programación. Luego se envía la consulta para ser corregida y calificarse. Un ejemplo es “Extensión de Moodle para facilitar la corrección automática de cuestionarios y su aplicación en el ámbito de bases de datos” (Abelló, Urpí, Rodríguez, Estévez, (n.d.)).

En Internet existe información de objetos de aprendizaje que permite compilar código de algún lenguaje de programación, pero tienen incorporado el compilador dentro del objeto. En la mayoría de estos es posible realizar consultas básicas.

III.1. Próposito

Por medio del primero de los objetivos del proyecto Webide, se pretende brindar la posibilidad de familiarizarse con un lenguaje de programación, y aportar facilidades que contribuyan con el aprendizaje del mismo. En base a lo analizado, se estima conveniente diseñar un compilador web que reúna algunas de las principales características de los compiladores nombrados anteriormente, e implemente mejoras, como mensajes adecuados, y se informe los resultados en idioma español.

Por otra parte, se busca crear una herramienta que colabore en el dictado de cursos de programación, implementando un objeto de aprendizaje con un mecanismo de corrección automática, que se comunique con el compilador Python localizado fuera del objeto. Esto, por un lado, beneficiaría a profesores, que muchas veces deben dedicar una considerable cantidad de tiempo en la corrección de las actividades de los alumnos, y por otro, lograría disminuir el tamaño de los paquetes, y los independizaría de versiones del compilador o del sistema operativo donde se utilicen. Se pretende entonces tener un sistema eficiente y además amigable hacia profesores y a estudiantes que tengan que transitar por los primeros pasos del conocimiento de un lenguaje de programación como Python.

IV. Herramientas

En este capítulo se hará una breve descripción de las herramientas más importantes que utilizaremos para poder implementar las aplicaciones propuestas, a saber: Moodle, SCORM, Common Cartridge y Web2py.

IV.1. Moodle

Moodle es un LMS (Learning Management System, Sistema de Gestión del Aprendizaje).

Es una aplicación web gratuita que los educadores pueden utilizar para crear sitios web dinámicos de aprendizaje en línea.

Para poder utilizarlo debe ser instalado en un servidor web (como por ejemplo XAMPP o Apache, ambos también gratuitos).

La plataforma es administrada por un usuario administrador que se define durante la instalación.

Pueden crearse múltiples cursos y distinguir roles de usuarios en los mismos (por ejemplo distinguir entre aquellos usuarios que son profesores y aquellos que son alumnos).

Soporta un rango de mecanismos de autenticación a través de módulos, cuyas principales características son:

- Método estándar de alta por correo electrónico: los estudiantes pueden crear sus propias cuentas de acceso.
- Método LDAP: las cuentas de acceso pueden verificarse en un servidor LDAP.
- IMAP, POP3, NNTP: las cuentas de acceso se verifican contra un servidor de correo o de noticias.
- Base de datos externa: Cualquier base de datos que contenga una tabla con al menos dos campos puede usarse como fuente externa de autenticación.

El profesor tiene un total control sobre las opciones de su curso y dispone de una serie de actividades que trae la plataforma por defecto: foros, cuestionarios, etc. También puede, como mecanismo de seguridad, añadir una clave de ingreso a los mismos.

Las calificaciones pueden registrarse en el historial del estudiante así como ser exportadas en diferentes formatos.

Es posible agregar recursos, como por ejemplo un paquete SCORM.

Por mayor información puede consultarse la página oficial www.moodle.org.

IV.2. SCORM

SCORM (Shareable Content Object Reference Model, Modelo de Referencia de Objetos de Contenido Compartible). Es un conjunto de estándares y especificaciones que permite crear objetos pedagógicos estructurados.

Con SCORM es posible crear contenidos que pueden importarse dentro de cualquier LMS, siempre y cuando este último soporte la norma SCORM (un ejemplo de tal caso sería Moodle).

Los componentes de la especificación son:

- Modelo de Agregación de Contenidos (Content Aggregation Model), que asegura métodos en materia de almacenamiento, de identificación, de condicionamiento de intercambios y de recuperación de contenidos.
- Entorno de Ejecución (Run-Time Environment), que describe las exigencias sobre el LMS que éste debe cumplir para que pueda gestionar el entorno de ejecución con el contenido SCORM.

Es necesario que el objeto y el LMS se comuniquen. Para ello existe una API escrita en JavaScript, que proporciona una manera estándar de comunicarse con el LMS, independientemente de la herramienta utilizada para desarrollar el contenido.

- Secuenciación y Navegación (Sequencing and Navigation), que permite una presentación dinámica del contenido. Describe cómo el sistema interpreta las reglas de secuenciación introducidas por un desarrollador de contenidos, así como los eventos de navegación lanzados por el alumno o por el sistema.

Los paquetes SCORM pueden incluir páginas web, programas JavaScript, presentaciones Flash, libros en diferentes formatos (como por ejemplo PDF) y cualquier otro elemento que funcione en un navegador web.

Existen editores gratuitos como el Reload Editor o el eXeLearning que permiten la creación de paquetes SCORM.

SCORM es el estándar de e-learning más utilizado a nivel mundial.

IV.3. Common Cartridge

Common Cartridge (CC) desarrollado por IMS Global, define un conjunto de especificaciones, para la distribución de contenido enriquecido, basado en web, ficheros multimedia, ficheros html, xml, etc, y especificaciones, que permite el intercambio de este tipo de paquetes y los LMS que soportan este tipo de especificaciones. Entre otros, se especifica Interoperabilidad de Test, cuestionario IMS o QTI, siendo los casos nombrados anteriormente, preguntas múltiple opción, verdadero y falso, etc.

El estándar define las especificaciones:

- IEEELOM, Dublin Core

- Paquete de contenidos IMS o CP v1.2
- Interoperabilidad Test y cuestionario IMS o QTI
- Servicios web de autorización v1.0

La especificación tiene como objetivo definir un estándar que permita el intercambio de estos paquetes, a cualquiera que esté en el área de la enseñanza.

En la siguiente figura se muestra un diagrama que explica el intercambio de un paquete CC.

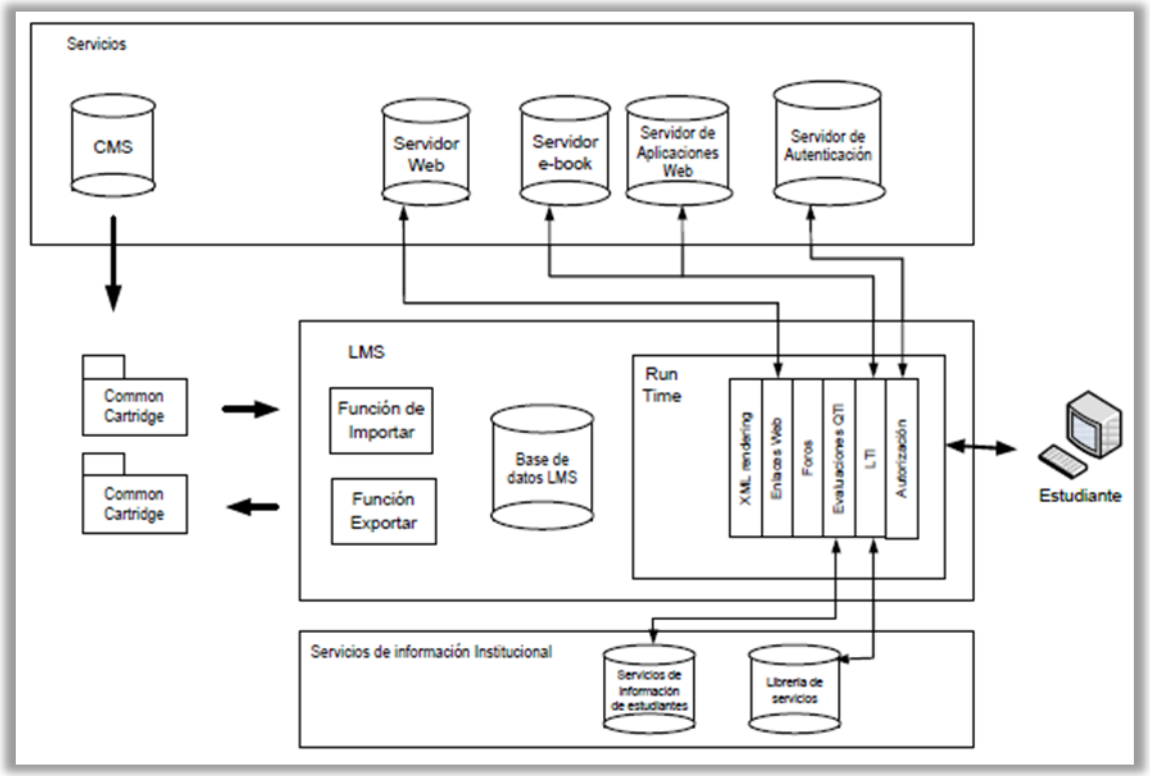

FIGURA IV-1 – Formato de intercambio de CC (San Cristóbal Ruiz, 2010).

Anteriormente fueron nombrados algunos de los contenidos que son soportados por Common Cartridge. A continuación se describen brevemente:

- Item-Folder: Una carpeta o folder representa una unidad de organización. Tiene un orden, y puede contener otras subcarpetas, almacenado elementos o ítems.
- Recurso. Contenido-Web: Los contenidos web son aquellos archivos que pueden ser entregados sobre la web.
- Recurso. Enlace-Web: Es un objeto de aplicación de aprendizaje. Es un enlace HTTP estándar.
- Recurso. Tema de discusión: Es un objeto de aplicación de aprendizaje, que permite crear un tema de discusión. Esto se realiza a través de las herramientas internas.
- Recuso. Evaluación: Es una instancia de evaluación QTI. Una evaluación puede tener distintos parámetros que la describen, como límites de tiempo, cantidad de intentos, cantidad de múltiple opción, etc.
- Recurso. Contenido asociado: Es un conjunto de archivos, utilizados por un objeto de aprendizaje.

- Referencia interna de Paquetes: Mediante una referencia interna permite que los archivos de un objeto de aprendizaje, puedan referenciar a otros archivos o ficheros dentro del paquete.
- Paquete de metadatos CC: Son metadatos del paquete, pudiendo contener descripción, accesibilidad, etc.
- Banco de preguntas: Puede incluirse (de forma opcional) un banco de preguntas dentro del paquete.

IV.4. Web2py

Web2py es un Framework Web escrito y programado en el lenguaje Python.

Un Framework es una estructura conceptual y tecnológica de soporte definido con base a la cual otro proyecto de software puede ser más fácilmente organizado y desarrollado. Un Framework Web está orientado más precisamente a facilitar el desarrollo de aplicaciones web, por ejemplo proporcionando bibliotecas para acceder a bases de datos, realizar gestión de sesiones, etc.

Web2py sigue el paradigma de programación MVC (Model View Controller, Modelo Vista Controlador), el cual se describirá en la siguiente subsección.

Posee importantes ventajas, entre las que se destacan:

- Está diseñado para tener seguridad.
- Incluye una capa de abstracción de bases de datos (DAL, Database Abstraction Layer) que escribe SQL dinámicamente sin que el desarrollador tenga que hacerlo. Soporta SQLite, MySQL y PostgreSQL, entre otros manejadores de bases de datos.
- Sus desarrolladores se comprometen a mantener la compatibilidad con versiones previas en futuras versiones.
- Ocupa poco espacio.

IV.4.1. Patrón de diseño Modelo Vista Controlador (MVC)

A continuación se describirá el paradigma MVC, que como ya se ha mencionado es el patrón de diseño seguido por Web2py así como por muchas otras aplicaciones web.

El Model View Controller (Modelo Vista Controlador, MVC) es un patrón de desarrollo de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica del programa en tres componentes distintos (modelo, vista y controlador).

Dicho patrón es altamente utilizado en aplicaciones web, donde la vista es la página HTML y el código que le provee dinamismo a la página (JavaScript, Python embebido en el HTML, etc), el modelo es el Sistema de Gestión de Base de Datos y el controlador es el que se encarga del flujo de trabajo de la aplicación.

Se ve entonces que una gran ventaja del MVC es el hecho de que permite una fácil y flexible estructuración del código.

Existen diferentes implementaciones del MVC, en general el flujo que sigue el control es el siguiente:

1) El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, presionando un botón).
2) El controlador recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada y gestiona el evento que llega.
3) El controlador accede al modelo, actualizándolo de forma correspondiente a la acción solicitada por el usuario.
4) El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario.
5) La interfaz de usuario espera nuevas instrucciones del usuario, retornando entonces al punto 1).

En el caso particular de Web2py el flujo de trabajo típico se describe en el diagrama de la figura siguiente:

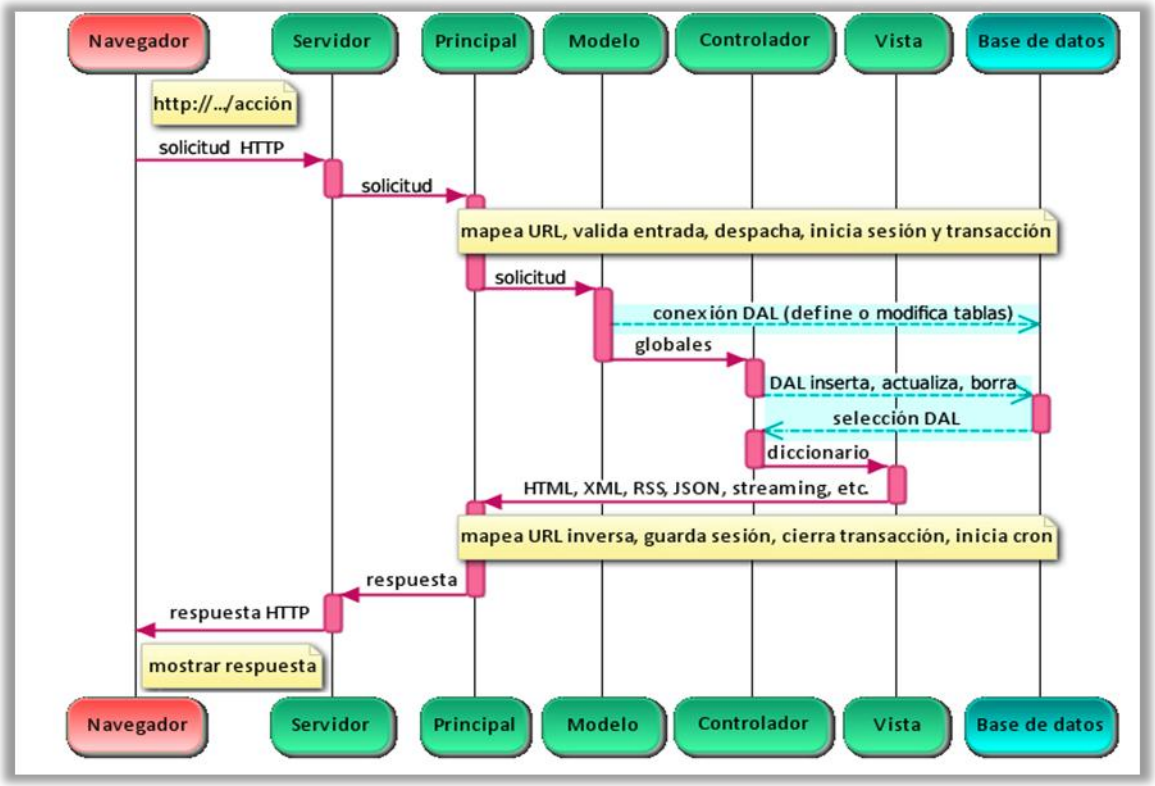

FIGURA IV-2 – Patrón MVC en Web2py (Di Pierro, 2007).

El servidor puede ser el incluido en Web2py o un servidor de terceros, como por ejemplo Apache.

"Principal" es la aplicación principal WSGI (Web Server Gateway Interface)[1]. Realiza todas las tareas comunes y envuelve las aplicaciones de usuario. Se encarga de los cookies, sesiones, enrutamiento de URL y enrutamiento externo, etc.

Los Modelos, Vistas y componentes del Controlador componen la aplicación de usuario.

Las líneas punteadas representan la comunicación con el motor(es) de base de datos. Las consultas de base de datos se pueden escribir en SQL crudo o utilizando la capa de abstracción de base de datos, de modo que el código de aplicación no dependa de un motor de base de datos particular.

El despachador "mapea" la dirección URL solicitada a una llamada de función en el controlador. La salida de la función puede ser una cadena o un diccionario de símbolos (una tabla hash). Los datos en el diccionario son presentados por una vista.

Todas las llamadas se encierran en una transacción y cualquier excepción no detectada hace que la transacción se cancele. Si la solicitud se hace correctamente, la transacción se confirma.

[1] Es un estándar Python para comunicaciones entre un servidor web y aplicaciones Python.

V. Compilador Web – Webide 1

V.1. Descripción de la solución

Para el diseño de un compilador Web, se parte de la función principal que es realizar una aplicación web, que permita compilar, ejecutar y ensayar código fuente de un lenguaje de programación, en este caso Python. Teniendo en cuenta las necesidades y objetivos, se estudió la opción de crear una aplicación web por medio del framework Web2py. Es necesario considerar en la implementación tres puntos fundamentales, los cuales serán la base para nuestro diseño:

- Una interfaz de usuario que permita pegar y/o escribir el código para capturar el texto ingresado por el usuario.
- Un método que permita el procesamiento del código y obtener la salida o errores.
- Una interfaz de usuario que permita desplegar la salida procesada por la aplicación, que le permita al usuario tener una realimentación.

Para el primer punto se pensó en crear una página web dinámica, que permita obtener el código introducido por el usuario. Esto se obtiene mediante la creación de una página web con código java script (soportado por web2py), permitiendo crear un área de texto editable, y además que reconozca sintaxis Python.

El tercer punto es similar a la solución descrita en el primer punto, pero con la diferencia que se divide la página en dos, dos áreas de texto no editables, donde en un área se desplegará la salida, y en la otra el código ingresado.

Para poder procesar el código, es necesario compilarlo mediante un compilador Python, para esto se debe correr un proceso en paralelo, ya que el framework Web2py es programado en Python. La solución para llevar a cabo esto, es hacer uso de un módulo de Python, el módulo “subprocess”, ya que este es el que mejor se adapta a nuestras necesidades utilizando la clase “Popen” de este subproceso. Este módulo, hace uso de la clase “Popen”, permitiendo procesar el texto obteniendo, obteniendo la salida o los errores en caso que se produzcan, para luego desplegarlo al usuario.

Web2py tiene la posibilidad de autenticar usuarios, pero Webide 1 está diseñada para ser implementada en un centro de estudio, para realizar pequeñas pruebas de código. Webide 1 es diseñada para que sea intuitivo y de rápido acceso, por lo que se decidió no autenticar ya que no se almacenará ningún tipo de datos en estas pruebas (podía ser modificado si es necesario).

Tratando de una aplicación diseñada para dar los primeros pasos de programación, se tomará en cuenta los problemas típicos, como bucles infinitos, en dónde se implementará un control, informando al usuario que se ha producido un error de este tipo.

Observando lo necesario para hacer uso de Webide 1, como lo es una conexión de banda ancha, además poseer un navegador web, se observó que es una buena práctica tener acceso

rápido a páginas de interés, como manuales Python, enlaces a sitio oficial de Python, etc. Para esto, se implementará una página de bienvenida, con enlaces de interés como los mencionados anteriormente.

V.1.1. Página de bienvenida

Webide 1 es una aplicación desarrollada con Web2py, accesible desde cualquier navegador (http://servidor:puerto/webide/default/index). Accediendo a dicha URL, se presenta la página de bienvenida.

En la parte superior se despliega el nombre de la Aplicación “Webide” y un mensaje definiendo la función de la página, “Programando en Python”.

En el centro de la página aparece un mensaje que describe la funcionalidad de la misma, y mediante un Botón “Comenzar!”, redirige a otra página dentro de la aplicación (http://servidor:puerto/webide/default/compilacion), que permite ingresar código Python.

A la derecha del mensaje de bienvenida presenta enlaces de interés, en primer lugar posee un enlace a un tutorial de Python por medio del enlace “tutorial Python”, en segundo lugar presenta un enlace al sitio oficial de Python en “python.org”. También existe un enlace a un libro de Python en español por medio del botón “Python para todos”. El cuarto enlace es “fing” que direcciona a la página de Facultad de Ingeniería de la UDELAR. El último enlace a “google”, teniendo acceso a un buscador online.

A la izquierda se presentan varios enlaces a diferentes páginas web propias de la aplicación, que tienen fines básicamente informativos. Además del botón “Inicio”, que redirecciona a la página de bienvenida, se encuentra “Filosofía Python”, que despliega información de los fundamentos populares de este lenguaje. En tercer lugar se tiene acceso a otra página que muestra prácticas recomendadas de programación, conocidas en el ambiente como los “mandamientos de Python”, por medio del enlace “Mandamientos”. Otro enlace lleva a la sección “Artículos”, donde pueden encontrarse artículos relacionados a Python que explican brevemente características y conceptos del propio lenguaje. Por último se tiene acceso a información propia de Webide, como versión, una pequeña descripción del desarrollo y los creadores de la aplicación por medio del botón “Acerca de Webide”.

También tiene opciones para compartir en redes sociales como Facebook y Twitter, mediante el botón “Share” que está situado a la derecha de la página de bienvenida, que además permite rápido acceso a casilla de mail, aplicaciones Google, etc.

Así se presenta la página de bienvenida:

FIGURA V-1 – Página de inicio de Webide1.

V.1.2. Página de compilación

Accediendo a la página de compilación, por medio del botón “Comenzar!” mencionado anteriormente, o simplemente insertando la dirección web siguiente (http://servidor:puerto/webide/default/compilacion), se tiene acceso a escribir o simplemente pegar código Python en el cuadro de texto, y poder compilar pequeños programas. Esta página presenta en la parte superior similares características a la página de bienvenida, pero a diferencia de la anterior, tenemos en el centro un área para ingresar el código (arriba de esta área editable aparece un mensaje de “Ingresa código Python por favor:”). Dispone de un botón “enviar consulta” que permite darle la orden de compilar el código, y un botón “Borrar” el cual permite borrar lo ingresado en el cuadro de texto. Luego de enviar la consulta del código se pueden producir dos sucesos:

- Si no se ingresa ningún carácter en el cuadro de texto y se envía la consulta, se despliega un mensaje en rojo, debajo del cuadro de texto, que informa lo siguiente: “Ingresa código Python en Webide”, no realizando ninguna acción de compilación.
- Si se ingresa código y se envía la consulta, se redirecciona a una página que despliega la salida (http://servidor:puerto/webide/default/salida), mostrando los mensajes correspondientes, según haya o no errores en el código ingresado.

Así se presenta la página de Compilación:

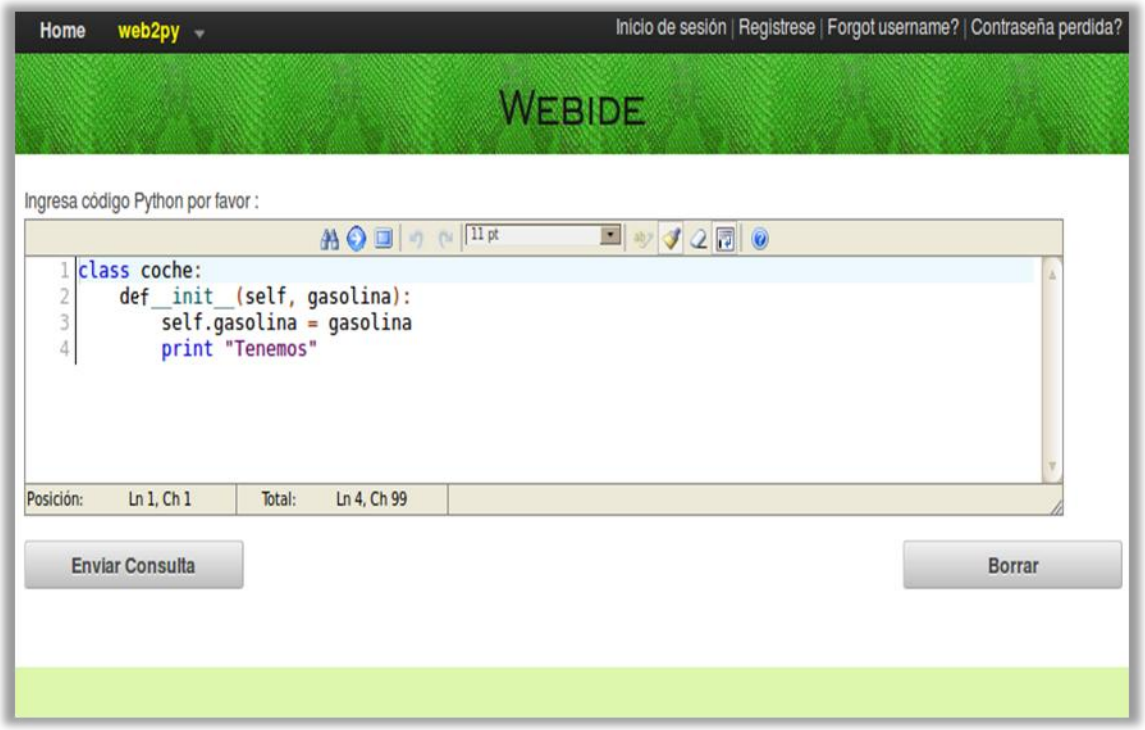

FIGURA V-2 – Página de compilación.

Así se presenta la página si se envía una consulta sin código:

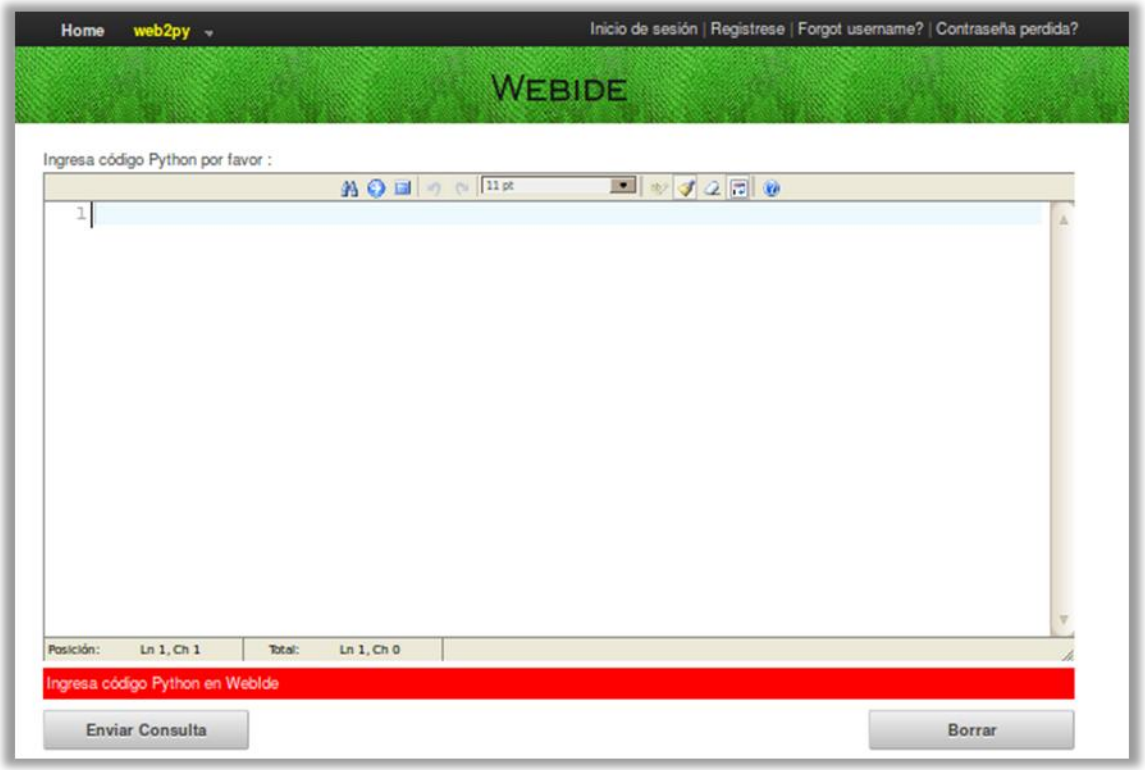

FIGURA V-3 – Página de compilación al enviar consulta vacía.

V.1.3. Página de Salida

Luego de ingresar el código en el cuadro de texto, y al realizar la consulta, se pueden producir dos sucesos:

- Si el código ingresado no posee errores de ningún tipo, la página de salida despliega la salida estándar dada por el compilador en un área de texto. Además se despliega a la derecha de la salida, el código ingresado por el usuario. También se muestra un mensaje temporal sobre el margen superior derecho imprimiendo “muy bien !” (letras en verde, durante 4 segundos), informando que la compilación del texto ingresado ha sido exitosa. Se proporciona un botón “Volver” de rápido acceso al cuadro de ingreso de texto, redireccionando nuevamente a la página de compilación para realizar otra consulta.
- Si el código ingresado posee errores de compilación, la página de salida despliega “Error”, imprimiendo la regla violada en el código. Además se despliega a la derecha el código ingresado por el usuario. También se muestra un mensaje temporal sobre el margen superior derecho imprimiendo “tuviste errores… Intenta de nuevo !” (letras en rojo, durante 4 segundos), informando que la compilación del texto ingresado ha tenido errores. Se proporciona nuevamente el botón “Volver” de acceso rápido al cuadro de ingreso de texto, redireccionando otra vez a la página de compilación para realizar una nueva consulta. Cuando se regresa a la página de compilación, por medio del botón volver, haya habido o no errores, el código ingresado es mostrado nuevamente. El mensaje de error indicará un error en el archivo.py “Traceback (most recent call last): File Archivo.PY,”, más el tipo de error que genera el código ingresado. El error que es desplegado por esta aplicación (Error en ARCHIVO.PY) fue modificado por seguridad, ya que se mostraría la ruta al lugar en el que se ejecutan y procesan los datos ingresados por el usuario.

Así se presenta la página de salida (mensaje de compilación sin errores):

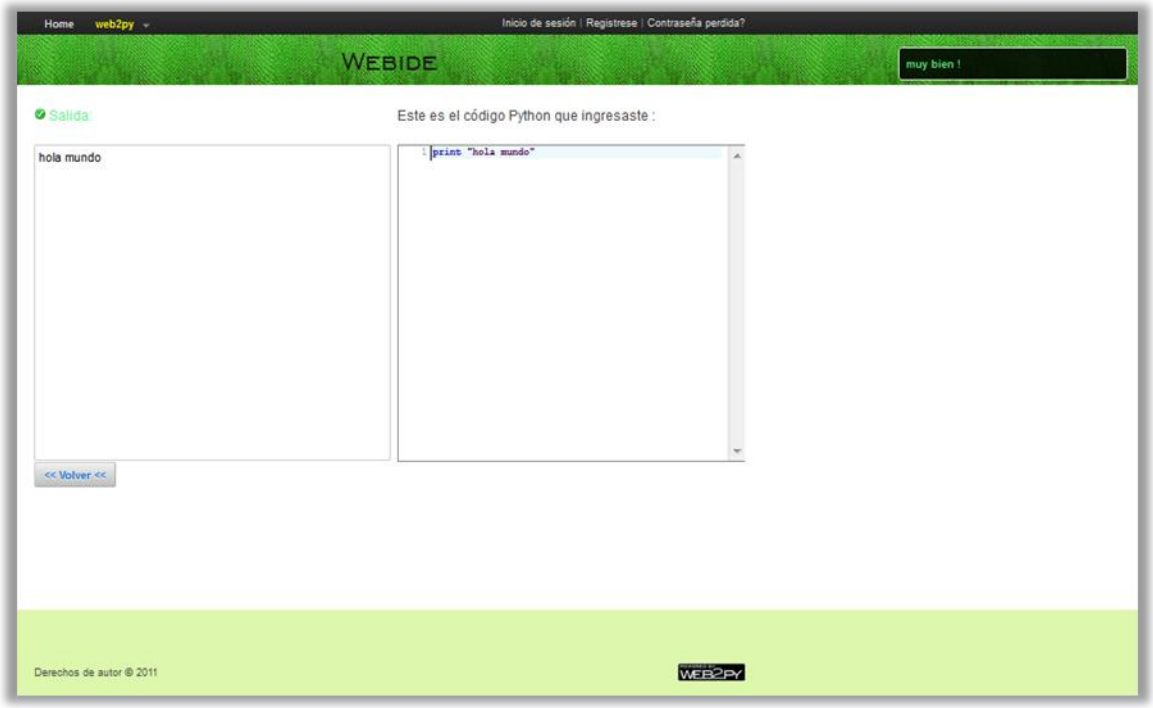

FIGURA V-4 – Página de salida sin errores.

Así se presenta la página de salida (mensaje de compilación con errores):

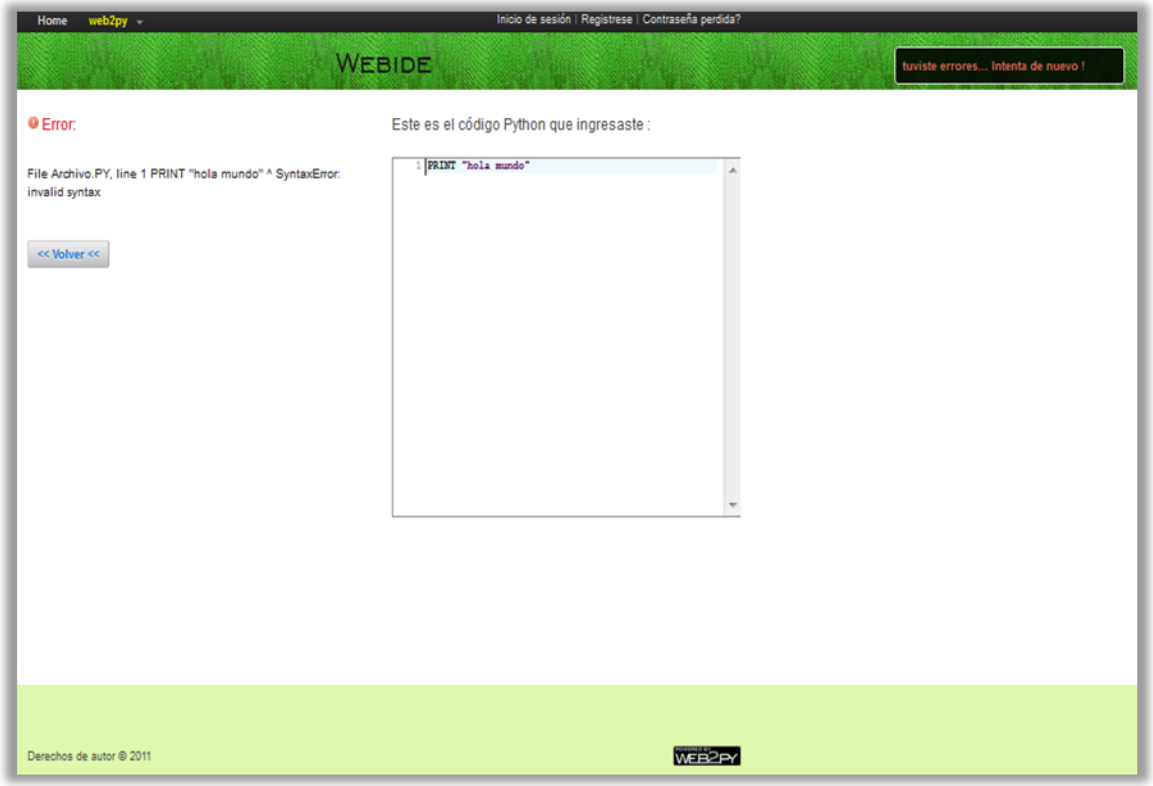

FIGURA V-5 – Página de salida con errores.

V.1.4. Página de bucles infinitos

Al producirse bucles infinitos en el código ingresado, se despliega la siguiente página, indicando posibles errores en el código:

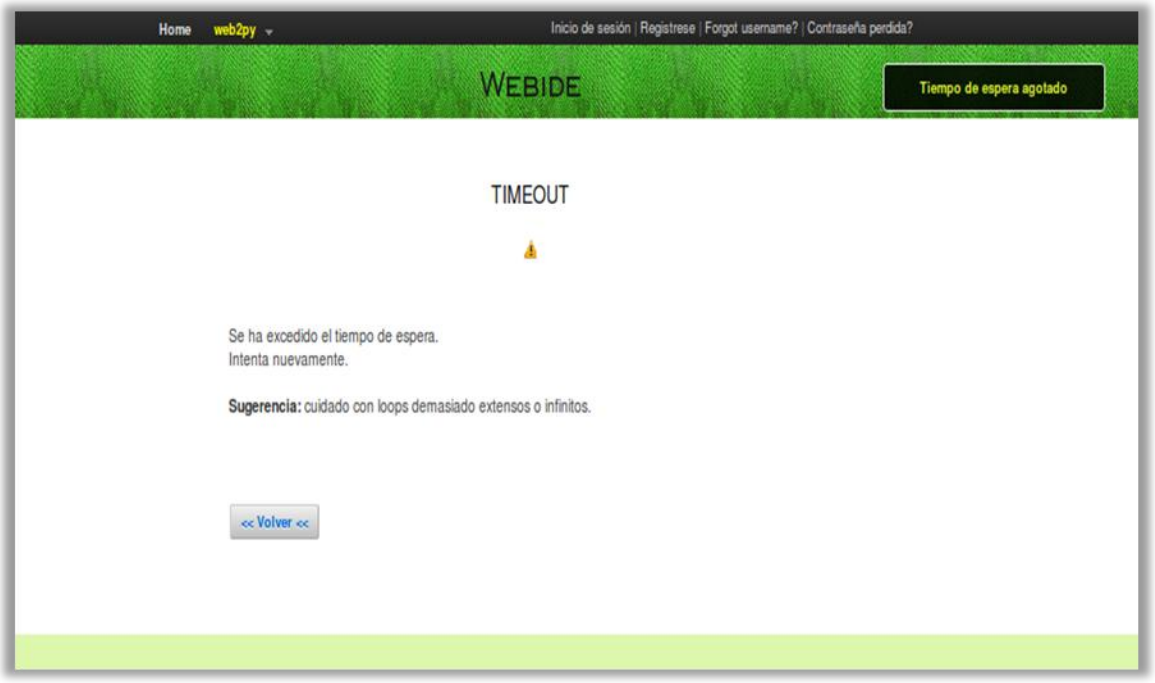

FIGURA V-6 – Error de "timeout".

V.1.5. Características funcionales

Ventajas

- La posibilidad de ejecutar lenguaje Python sin la necesidad de un compilador instalado en el PC del usuario.
- Bajos requerimientos para hacer uso de esta aplicación. Tener un navegador web (Mozilla Firefox, Google Chrome), y tener acceso a una conexión de banda ancha, ya que la aplicación se ejecuta en un servidor externo al usuario, mediante una aplicación web.
- Es una herramienta de manejo intuitivo y fácil. Cualquier persona que posea conocimientos mínimos de programación, será capaz de utilizar esta aplicación.
- Acceso rápido desde la página de inicio a sitios que pueden ser de ayuda para el usuario, como Python.org, tutorial de Python, libro de Python en español, buscador Google.
- Resaltado de sintaxis e informe de los números de cada línea, indicando además la posición del cursor.
- Aplicación de uso libre. Para su implementación se utilizó Python y Web2py (de uso libre), con el objetivo de la divulgación, el conocimiento y desarrollo del lenguaje Python.

- Fácil acceso; no es necesario tener un usuario y contraseña para hacer uso de esta aplicación.
- La aplicación soporta multi-acceso, teniendo la posibilidad de acceder y utilizar Webide 1 varios usuarios al mismo tiempo, debido a que tiene implementada esta facilidad.
- Webide 1 es de muy fácil instalación en cualquier PC o servidor que tenga un compilador Python y el servidor de Web2py para el servicio web, no siendo necesario realizar ninguna modificación en el código para que funcione.

Limitaciones

- Teniendo en cuenta que es una herramienta con un fin educativo, para la introducción a la programación y conocimiento del lenguaje Python, está diseñada para realizar pequeños programas de este lenguaje.
- Web2py tiene múltiples precauciones en cuanto a la seguridad y a la protección de los datos almacenados, pero esta aplicación, al no tener por el momento registro de usuarios (sí permisos de administrador), presenta bajo seguimiento del uso y accesos de los usuarios.
- No se ha podido probar el buen funcionamiento con una carga de usuarios al mismo tiempo.

V.2. Implementación

V.2.1. Diagrama de componentes

En la siguiente figura se observa el diagrama de componentes de la aplicación basada en el patrón MVC:

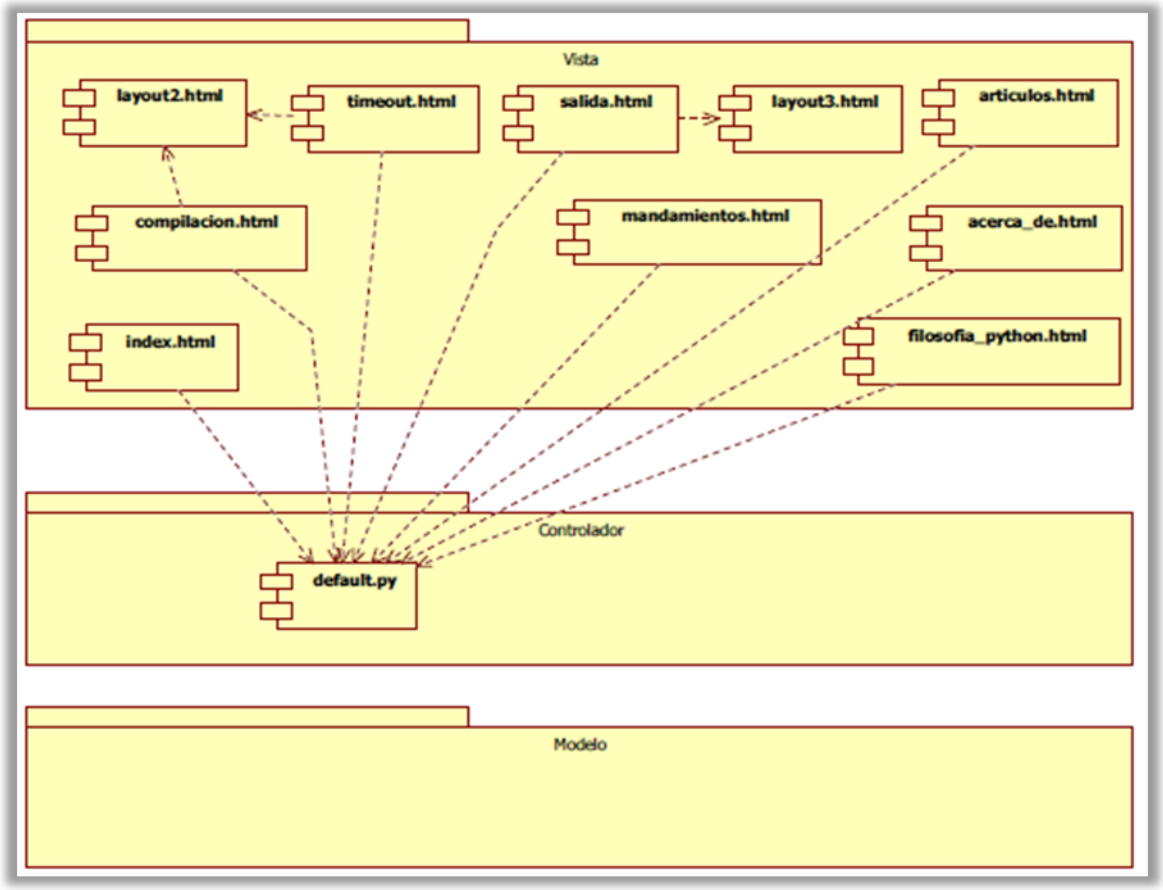

FIGURA V-7 – Diagrama de componentes.

V.2.2. Descripción del proceso básico

El proceso básico es el siguiente:

- Mostrar una página de bienvenida para el usuario.
- Mostrar la página en la cual el usuario va a ingresar el código.
- Capturar la entrada ingresada por el usuario (se supondrá es un texto plano).
- Almacenar dicha entrada en un archivo con extensión .py.
- Compilar y ejecutar dicho archivo.
- Capturar la salida del mencionado archivo y los errores en caso de que los haya en variables.
- Desplegar la salida o los errores en caso de que los haya en una página.

V.2.3. Descripción del código en Web2py

Web2py está basado en el uso del MVC (Model View Controller), donde en el Modelo se describen los datos, en el Controlador se define el flujo del programa y en la Vista se despliegan los resultados.

Para el diseño de la página no se trabajó con el modelo, ya que no fue necesario definir ningún tipo de datos en particular. No se desea almacenar el código que ingresa el usuario en una base de datos, sino que simplemente el código ingresado es compilado, ejecutado, y luego de desplegado es descartado.

Controlador

default.py: Es el único controlador, y es el que se encarga de las tareas del flujo de programa.

Vista

Existen cuatro vistas principales

index.html : esta vista es la que se encarga de la página de bienvenida al usuario.

compilacion.html: Esta vista es la encargada de ingresar el código en un área de texto, para su posterior procesamiento.

salida.html: Esta vista es la encargada de desplegar la salida, en los casos que posea errores o no en el código (No se encarga de bucles infinitos).

timeout.html: Esta vista es la encargada en desplegar si existen problemas de bucles infinitos o procesamiento de tiempo extenso (más de siete segundos).

Modelo

No se modifica el modelo, ya que no fue necesario realizar almacenamiento de datos, ni registro de usuarios.

Código del proceso básico

En la vista index.html se escribe código HTML para mostrar al usuario la página de bienvenida.

```
<h3> <div style="text-align:center;">

    {{=message}}<FONT style="font-family:tipo webide, engraversgothic
bt,vrinda;" SIZE=7>Webide</FONT>!

</div></h3>

</br>

<h7> <FONT COLOR="#305040"> Esta página te permitirá compilar y ejecutar
programas escritos en el lenguaje Python, desde cualquier lugar y sin
necesidad de tenerlo instalado en tu computadora. </font>

</h7>

</br></br>

<center>{{=A(T("Comenzar!"), _href=URL('compilacion'),
_class='button')}}</center>
```

En este código se destaca el mensaje de bienvenida y el botón “Comenzar !”, que redirige a la página de compilación. Encontrándose en la página de compilación (compilacion.html), el usuario tiene la posibilidad de ingresar su código. Esto se logró escribiendo el siguiente código en la vista compilacion.html:

```
<head>

<script    language="javascript" type="text/javascript"

src="{{=URL('static','edit_area/edit_area_full.js')}}">
```

```
</script>
<script language="javascript" type="text/javascript">
editAreaLoader.init({
      id : "editable",
      syntax: "python",
     start_highlight: true,
     font_size: 11,
     replace_tab_by_spaces: 4,
     show_line_colors: true,
     word_wrap: true,
     language: "es",
     allow_toggle: false });
</script>
</head>
```

En el código anterior se implementa un área de texto del tipo editArea. El usuario puede ingresar su código en el área de texto, esto se logra mediante _id= “editable”. Con este campo fijado de esta forma, es posible escribir o pegar texto. También se especifica el tipo de sintaxis, en este caso Python (syntax: "python").

En el siguiente segmento de código se observa la obtención del código ingresado. El texto ingresado por el usuario se almacenará en la variable de sesión llamada “session.texto”. El controlador (default.py) utiliza un formulario, con un validador IS_NOT_EMPTY para que en caso de realizar la consulta vacía, despliegue un mensaje de error informando de tal situación, dicho mensaje es “Ingresa código Python en WebIde”.

```
# Se crea formulario con sus validadores
form = FORM(TEXTAREA(_id="editable", _style="width:100%; height:405px",
_wrap="soft",     _name='texto_a_ingresar', value=session.texto,
requires=IS_NOT_EMPTY("Ingresa código Python en WebIde")),
       INPUT(_type='submit', _style="width:20%; position:relative;
top:10px"),
       INPUT(_type='button', _style="width:20%; position:relative;
top:10px; left:60%", _value="Borrar", _onclick="history.go(0)"))
```

Se tienen dos botones, uno que es para borrar el texto en el editor, y otro para enviar la consulta. Se especifica el tipo, se indica que el botón del tipo “submit”, siendo para realizar la consulta, además permite enviar el formulario una vez que éste ha sido aceptado y completado. El botón borrar del tipo “button”, permite borrar lo escrito en el área de ingreso de texto.

Con el texto ingresado por el usuario se procede a almacenar su contenido en un archivo con extensión .py. Para lograrlo se tiene el siguiente fragmento de código en el controlador:

```
# Aceptado el formulario se crea archivo python (python.py) y sus permisos

ran = random.randint(100,999)

arch_Pyth = "archivo"+str(ran)+".py"

a2 = open (arch_Pyth ,"w")

a2.write('#-*- coding: utf-8 -*-\n'+form.vars.texto_a_ingresar)

a2 = open (arch_Pyth,"r")

# Se almacena el texto ingresado para mostrarlo luego en la página de
resultados

session.texto = a2.read()

session.texto = session.texto.replace('# -*- coding: utf-8 -*-\n','',1)

a2.close()
```

En este segmento de código, permite copiar el contenido de la variable de la entrada, que está almacenado en la variable de formulario "_name=texto_a_ingresar", en un archivo con extensión .py. Dicho archivo se guarda en el servidor, en este caso con un nombre aleatorio (arch_Pyth = "archivo"+str(ran)+".py"). Esto se realiza para evitar problemas de concurrencia, previendo que múltiples usuarios podrían acceder al servicio al mismo instante. Hacer esto, reduce la probabilidad de tener inconvenientes. También en este segmento se almacena el código ingresado por el usuario en la variable a2 que será codificado en formato utf-8.

En el controlador se implementó un subproceso, que es la parte fundamental del código, lo que permite ejecutar, y compilar el código ingresado por el usuario, teniendo en cuenta que se ingresa el código mediante una página web. Se implementa el módulo "subprocess", y de éste la clase "subprocess.Popen" que permite almacenar en distintas variables la salida que se desplegará al ejecutar el código del usuario. Esencialmente lo que hace es correr un proceso en paralelo, en el cual almacena la entrada estándar o error estándar del código ingresado en tuberías. Al generar este subproceso, almacena en una variable la salida, si no se produce errores de ningún tipo, y en cambio, si por algún motivo, existe un error en el código o se viola alguna regla de Python, "subprocess.Popen" almacena en una variable el error estándar. El proceso se define de la siguiente manera en el controlador:

```
proc = subprocess.Popen([arch_Pyth], stdout=PIPE, stderr=PIPE, shell=True)
```

El módulo de este subproceso permite generar nuevos procesos, conectarse a la entrada, salida, error estándar, y obtener sus códigos de retorno. La creación y gestión de procesos subyacentes es manejado por la clase "Popen". Además ésta clase permite detener el proceso si así se desea.

En la variable "stdout" se almacena la salida estándar, y en "stderr", almacena el error estándar. Si el código no presenta errores "stderr" es vacío. En cambio si existen problemas en el código, por ejemplo, indentación, compilación, la clase "subprocess.Popen" almacena los

errores en “stderr”. El parámetro “shell=True” en Windows debe ser especificado cuando el Shell tiene el comando ingresado integrado. En el caso de Unix “shell=true” significa utilizar el Shell por defecto, en este caso bin/sh.

Un detalle a tener en cuenta es que para llamar al subproceso de la clase “Popen”, se debe discriminar entre los diferentes sistemas operativos, ya que estos funcionan diferentes a la hora de ejecutar un archivo de extensión .py. En el código del controlador siguiente, se puede apreciar la discriminación realizada según los sistemas operativos, dónde en la variable “so” se almacena el tipo se sistema operativo (Windows u otros):

```
# Se llama a subprocess dependiendo del Sistema Operativo

so = os.name

if so == 'nt':

   proc = subprocess.Popen([arch_Pyth], stdout=PIPE, stderr=PIPE,
shell=True)

else:

   proc = subprocess.Popen('python '+ arch_Pyth, stdout=PIPE, stderr=PIPE,
shell=True)
```

En el controlador (default.py) también se toma en cuenta que el usuario puede ingresar un código que posea un bucle infinito, lo que es común en los primeros pasos de aprendizaje de un lenguaje de programación. Con el siguiente código se realiza un contador de tiempo en el cual, si el proceso de ejecución “proc” persiste por más de siete segundos, detiene el proceso invocando la función “matar()”, y luego se redirige a una página (página de bucle infinito) en la cual despliega un mensaje de problemas de bucle infinito:

```
# Timer

TIEMPO_MAX=7.0

t=threading.Timer(TIEMPO_MAX,matar)

t.start()
```

Al terminar este contador (luego de 7 segundos), llama a la función “matar()”, definiéndose de la siguiente manera:

```
def matar():

    proc.stdout.close()

    proc.stderr.close()

    proc.kill()

    return
```

Para detener el proceso, previo se debe cerrar los archivos principales del subproceso que esta ejecutándose, los archivos son “stdout” y “stderr”. Luego con un bucle While, se consulta si el proceso está corriendo, si tarda siete segundos o mas se corta la ejecución. Seguidamente se consulta si está cerrado “stdout”, si se debe a que llamó a la función “matar()”, se redirecciona

a una página informando sobre el error de “timeout” (para ello se creó una función “timeout()” y su vista correspondiente), si no, se guardan “stdout” y “stderr” en las variables “pstdout” y “pstderr” para su posterior procesamiento. Para consultar si el proceso “proc” ha finalizado, se realiza la consulta con “proc.poll()”, si esta variable es distinta de “None”, significa que finalizó el proceso.

```
# Se pregunta por el estado del proceso

esta_vivo=True

while esta_vivo:

   if proc.poll() != None:

       esta_vivo=False

       t.cancel()
```

Paso siguiente se procede con la captura de la salida y los errores del código, en caso de que presente problemas en el código ingresado, discriminando como se mencionó anteriormente, el código no haya producido un bucle infinito. Esto se logra mediante el siguiente código, en el controlador:

```
# Si stdout está cerrada en este punto es porque se llamó a la función
matar()

if proc.stdout.closed:

    pstderr="error de timeout"

    pstdout=""

    session.flash = {'adv' : 'Tiempo de espera agotado'}

    # Se elimina el archivo creado

    os.remove(arch_Pyth)

    redirect(URL(request.application, 'default', 'timeout'))

else:

    pstdout, pstderr = proc.communicate() # communicate() devuelve una
tupla con la salida estándar y errores

# Salida para mostrar en las vistas

session.salida = pstdout
```

Se hace uso de “proc.communicate()”, lo que permite capturar la salida o los errores del archivo, en las variables “stdout” y “stderr” respectivamente. Estas variables contienen la salida que se desplegará en la página “salida.html”.

Cuando hay errores, se despliega una ruta artificial al archivo y el informe de error. Esto se almacena en la variable “session.error”. Por temas de seguridad, no se muestra la ruta real al usuario, se modificó “session.error” y se despliega la salida de error conteniendo únicamente el tipo de falla ocurrido.

En la vista salida.html se tiene el siguiente segmento de código:

```
{{#If para distinguir la salida y desplegarla según haya o no errores}}

{{if (session.error != ""):}}

<h5> <img src="{{=URL('static','images/error.png')}}"/> <FONT
COLOR="#DD0C0C">Error: </FONT> </h5></br>

<h6> {{=session.error}}</h6> </br>

{{else:}}

<h5> <img src="{{=URL('static','images/ok.png')}}"/> <FONT COLOR="#5BF18B">

Salida: </FONT></h5></br>

<textarea style="border-color:#; resize:none; font-size:12pt; color:#000;

  width:100%; height:389px"; onfocus="this.blur()"; readonly="true";
wrap="soft">{{=session.salida}}

</textarea>

{{pass}}
```

Lo que se encuentra en corchetes dobles es código Python embebido en código HTML.

Sólo se despliega un mensaje de error en caso de que exista y de lo contrario se muestra la salida. Esto se logra a través de la sentencia if-else, que permite discriminar si se ha producido error, para luego desplegar la salida correspondiente.

Se incluye el código siguiente en la vista salida.html, una funcionalidad que le permite al usuario ver en la página, el código que se ha ingresado, además de la salida (o el error). Esto es necesario para detectar fácilmente los errores, ya que se despliega el error del código y el código ingresado.

```
<h5> <FONT COLOR="#424242"> Este es el código Python que ingresaste:
</FONT> </h5></br>

<Textarea id="no_editable", style="width:100%;
height:390px">{{=session.texto}}

</Textarea>

{{end}}
```

En el CD se puede ver el código completo del controlador y las vistas antes mencionadas. Aquí se ven los segmentos de código que describen el funcionamiento principal de la aplicación.

V.2.4. Resolución de algunos problemas

- Reconocimiento de sintaxis Phyton

El reconocimiento y resaltado de sintaxis se logró mediante el editor EditArea. La característica principal que posee este editor, es que puede implementar reconocimiento de sintaxis de diferentes lenguajes, entre ellos Phyton. Además tiene varias facilidades, por ejemplo informar número de línea, buscar palabras, etc.

Para tener accesible este editor, fue necesario almacenar archivos que permiten la implementación del mismo. El directorio en el que fueron almacenados es "/static/edit_area". La manera de implementar este editor en Webide 1 fue comentado anteriormente, colocando en las vistas un área de texto del tipo editable (o no editable, ej página de salida), como es en la página de compilacion.html. Para el caso de la página de salida, se implementó un área de texto no editable, ya que se desea desplegar código ingresado. Para esto se debió modificar el código en el archivo edit_area_full.js del directorio antes mencionado, la línea "is_editable=false".

- Testeo simultaneo de código

Teniendo en cuenta que esta aplicación será utilizada por más de un usuario, se presentó el problema de que dos o más usuarios podrían enviar una consulta simultáneamente, pudiendo producirse un conflicto entre los archivos. Para esto se implementó un nombre aleatorio para los archivos del usuario, mediante una función, que calcula un entero "ran", que varía entre 100 y 999. De esta manera se disminuye considerablemente que dos usuarios hagan una consulta al mismo tiempo, y que se produzca un conflicto entre archivos. En el siguiente código se implementa:

```
ran = random.randint(100,999)

arch_Pyth = "archivo"+str(ran)+".py"
```

- Ejecución con el interprete Python del servidor

Para ejecutar el archivo anterior, el cual contiene el código ingresado por el usuario, se debió tener en cuenta que esta aplicación puede correr en un servidor con sistema operativo Windows o Unix. Para tener el caso cubierto y poder correr un subproceso con el archivo de extensión .py, se implementó la clase "subprocess.Popen" del módulo "subprocess". Mediante ésta y discriminando con una sentencia if-else, permite solucionar estos inconvenientes y almacenar las salidas, ya sea errores o salida estándar en variables ("stdout" o "stderr"). Además permite finalizar el proceso de ejecución, por ejemplo si se produce un bucle infinito, esto se realiza a través de la función matar(). La variable "so" almacena el tipo de sistema operativo (Windows o Unix):

```
so = os.name

if so == 'nt':

   proc = subprocess.Popen([arch_Pyth], stdout=PIPE, stderr=PIPE,
shell=True)

else:

   proc = subprocess.Popen('python '+ arch_Pyth, stdout=PIPE, stderr=PIPE,

shell=True)

# Función para terminar el proceso

def matar():

   proc.stdout.close()
```

```
   proc.stderr.close()
   proc.kill()
   return
```

- Bucles infinitos

En los primeros pasos de un lenguaje de programación es común cometer errores de bucles infinitos. Para este problema se implementó en esta aplicación una función la cual permite detectar este problema. Para ello se tiene un contador, durante siete segundos (podría modificarse), que comienza a contar luego de ejecutado el subproceso. Si el proceso dura siete segundos o más, el temporizador invoca a la función "matar()", la cual termina el proceso "proc". La manera en que se consulta si el proceso "proc" está finalizado se realiza mediante un bucle while, que consulta si el proceso finalizó, mediante "proc.poll()". Si el proceso terminó antes que el contador llegue a los siete segundos, se cancela el temporizador. En caso contrario, se invoca a la función "matar()" y se finaliza el proceso, evitando el bucle infinito. En el siguiente código se tiene esta implementación:

```
# Función para terminar el proceso
def matar():
   proc.stdout.close()
   proc.stderr.close()
   proc.kill()
   return
# Timer
TIEMPO_MAX=7.0
t=threading.Timer(TIEMPO_MAX,matar)
t.start()
# Se pregunta por el estado del proceso
esta_vivo=True
while esta_vivo:
  if proc.poll() != None:
     esta_vivo=False
     t.cancel()
```

En la función "matar()" se cierran las salidas estándar y errores del proceso "proc" antes de detener el subproceso, y luego se finaliza el proceso mediante "proc.kill()". Luego para discriminar la salida, se realiza una consulta a la salida "stdout". Si ésta fue cerrada por la función "matar()", entonces se redirige a una página informando el problema detectado. En el caso que la "stdout" no fue cerrada abruptamente se continúa con el flujo de la aplicación, como se puede ver en el código siguiente:

```
# Si stdout está cerrada en este punto es porque se llamó a la función
matar()

if proc.stdout.closed:

   pstderr="error de timeout"

   pstdout=""

   session.flash = {'adv' : 'Tiempo de espera agotado'}

   # Se elimina el archivo creado

   os.remove(arch_Pyth)

   redirect(URL('webide', 'default', 'timeout'))

else:

   pstdout, pstderr = proc.communicate() # communicate() devuelve una tupla
con la salida estándar y errores
```

VI. Aplicación de corrección automática

VI.1. Implementación mediante el protocolo SCORM

La idea es crear un objeto de aprendizaje para luego empaquetarlo según las reglas del protocolo SCORM e integrarlo a la plataforma Moodle.

Como requisito interesa que el compilador no forme parte del objeto, ya que no sería ni práctico ni razonable que para cada tarea diferente que pueda realizarse el tener que incluir junto con ella el compilador (eventualmente puede haber diversas tareas y cada una de ellas corresponderá a un objeto diferente).

Sí se debe tener instalado el compilador en alguna parte, de otra forma sería imposible el poder compilar y ejecutar el código fuente. Se supondrá que el mismo se encuentra en un servidor.

La idea de resolución es la siguiente:

1) El alumno ingresa a la plataforma de aprendizaje.
2) El alumno abre el objeto de aprendizaje, ingresa a la tarea e intenta resolverla.
3) El objeto accede a un sitio local o remoto al cual envía el código del estudiante y el de testeo y recibe los resultados.
4) El objeto le reporta a la plataforma de aprendizaje la nota obtenida en la tarea.

Para lograr cumplir con el ítem 3) se debe poder lograr una comunicación entre el objeto y el servidor donde se encuentre el compilador, ya que no se incluirá este último dentro del objeto.

Por tanto se tienen diversos problemas de comunicación a resolver, uno entre el Moodle y el objeto, otro entre las diversas páginas que puedan requerirse dentro del objeto y otro entre el objeto y el servidor donde se encuentra el compilador. Esto se muestra en la figura siguiente.

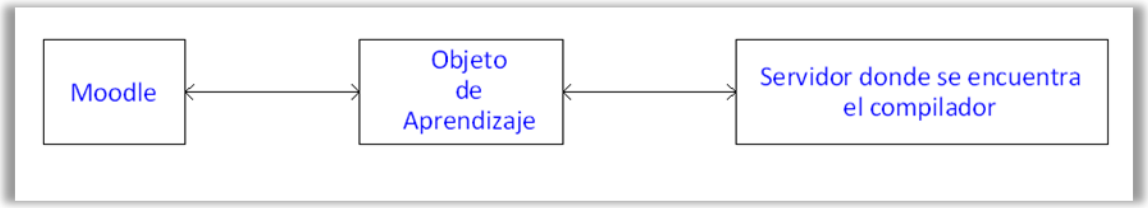

FIGURA VI-1 – Esquema de comunicación, objeto de aprendizaje.

La comunicación entre el Moodle y el objeto se logra a través de una interfaz definida por el protocolo SCORM. Ésta consta de algunas funciones más bien básicas, se destacan:

- Initialize(): Inicializa una sesión.
- Terminate(): Finaliza una sesión.
- GetValue(): Obtiene un valor.
- SetValue(): Establece un valor.

- GetLastError(): Código de error del error actual.
- GetStringError(): Devuelve una descripción en formato texto del error actual.
- GetDiagnostic(): Le permite al LMS definir diagnósticos adicionales a los que provee la API.

De esta forma se puede saber qué alumno es el que inició una sesión, mandarle datos al objeto, asignar una nota, etc.

La comunicación entre las diversas páginas del objeto se realiza a través del lenguaje de programación JavaScript.

Es la comunicación entre el objeto y el servidor donde se encuentra el compilador lo que trae las mayores dificultades. Al momento de escribir este texto no hay ninguna función que permita una comunicación directa entre el objeto y una página exterior al mismo en el sentido de enviar, solicitar y recibir información sin perder los datos de quién lo está solicitando, es decir, se puede redireccionar al usuario (el alumno) a una página exterior y pedirle que suba un archivo, corregir el mismo y obtener una nota, pero luego de esto no hay forma de saber en la plataforma de aprendizaje a qué estudiante le pertenece la misma. Esto se debe a que la inserción/actualización de notas a través del protocolo SCORM se hace reconociendo una sesión, y cuando se envía la solicitud a otra página se pierden los datos necesarios para saber quién había iniciado la misma.

La API que proporciona el estándar SCORM versión 2004 para comunicarse con el exterior está totalmente orientada a hacerlo con el LMS (en nuestro caso el Moodle), dejando la comunicación entre la aplicación y lo que "está fuera de la misma" totalmente a cargo del LMS.

Como alternativa para solucionar este inconveniente podría incluirse el compilador dentro del objeto; de esta forma no haría falta el último tipo de comunicación y podría implementarse una solución. Un problema de la misma es como ya se ha mencionado, que no es práctico tener el compilador en cada objeto que se suba, pero peor aún, de hacerlo, ¿qué compilador se debería cargar? ¿El de Windows? ¿El de Linux? (se recuerda son diferentes). Una de las metas es crear una aplicación que sea independiente del sistema operativo que se utilice, por lo tanto se deberían cargar ambos compiladores.

También se podría tratar de modificar el protocolo pero esto excede el alcance del proyecto.

Una alternativa viable, al menos en un principio, es estudiar otro protocolo de objetos de aprendizaje, el Common Cartridge, para saber si el mismo provee alguna función que SCORM no que permita lograr el objetivo planteado.

VI.2. Implementación mediante el protocolo Common Cartridge

Con el estudio de este protocolo se pretende lograr crear un objeto de aprendizaje, que pueda cubrir las necesidades mencionadas anteriormente, es decir, poder conectar un objeto de Aprendizaje con el exterior, para realizar la consulta al compilador por fuera. Common Cartridge (CC) es un conjunto de especificaciones que permite la creación de un Objeto de

Aprendizaje. Para nuestro objetivo, Common Cartridge implementa una facilidad que es Basic Learning Tool Interoperability (Basic LTI).

Basic LTI es una especificación que permite la integración de servicios dentro de un LMS, pudiendo comunicarse la plataforma de aprendizaje (TC, Tool Consumer, Herramienta Consumidor), con un recurso (TP, Tool Provider, Herramienta Proveedor).

En la etapa de diseño, se realizó un esquema de lo necesario para cumplir con los requisitos, y lograr un sistema de corrección automática, asignando una calificación, de tareas de lenguaje Python, y que realizara la compilación fuera del objeto. Para ello se tomó en cuenta la nueva especificación Basic LTI. Siguiendo el esquema:

- Primero obtener un LMS, en este caso Moodle, versión estable, que soportara la nueva funcionalidad Basic LTI.
- Tener en cuenta las limitaciones y opciones, que presenta esta nueva especificación para el envío de código Python (en este caso) en formato texto plano, para su posterior procesamiento.
- Elegir un recurso donde se implementará el compilador, no descartando la posibilidad de un paquete Common Cartridge para la consulta.
- Desarrollar un método de calificación luego de obtener la salida del código compilado.
- Almacenar la calificación en el LMS.

Mensajes soportados por la especificación:

Son parámetros de intercambio mediante mensajes POST, en un lanzamiento Basic LTI definidos por esta especificación, entre la TC y la TP. Se debe destacar que no todos los parámetros son obligatorios, según lo requiera la TP.

lti_message_type=basic-lti-launch-request: Este dato es requerido, e indica que es un lanzamiento Basic LTI.

lti_version=LTI-1p0: Este parámetro indica la versión del lanzamiento Basic LTI. Este parámetro es requerido.

resource_link_id=88391-e1919-bb3456: Identificador único en el enlace. Si la herramienta se encuentra varias veces en el contexto, tendrán estos números distintos. Este parámetro es necesario.

resource_link_title=My Weekly Wiki: Este es el título que aparece en el recurso, para activar el enlace. Este parámetro es recomendado.

user_id=0ae836b9-7fc9-4060-006f-27b2066ac545: Este identifica al usuario. Este parámetro es recomendado.

resource_link_description=…: Mensaje enviado con descripción del destino del enlace.

user_image=http://....: Atributo que determina la URI (Uniform Resource Identifier, identificador uniforme de recursos) de la foto del usuario que realiza el enlace. Pueden ser JPG, GIF, PNG. Parámetro opcional.

roles=Instructor: Lista de roles separada por comas, debe poseer un rol por lo menos del sistema LIS. Parámetro recomendado.

lis_person_name_given=Jane

lis_person_name_family=Public

lis_person_name_full=Jane Q. Public

lis_person_contact_email_primary=user@school.edu: Estos campos contienen información de la cuenta del usuario. Parámetros recomendados.

context_id=8213060-006f-27b2066ac545: Identificador del enlace. Parámetro recomendado.

context_type=CourseSection: Cadena separada por comas. Tiene que tener al menos un URN. De manera de identificar el contexto. Parámetro opcional.

context_title=Design of Personal Environments: Título en texto sin formato. Recomendado.

context_label=SI182: Destinada para la etiqueta del contexto. Parámetro recomendado.

launch_presentation_locale=en-US: Etiqueta que identifica el idioma y país.

launch_presentation_document_target=iframe: Identificador de ventana donde presenta el lanzamiento la TC. Parámetro recomendado.

launch_presentation_css_url=: Permite adaptar la apariencia de los LMS. También se utilizará para mejoras futuras.

launch_presentation_width=320: Ancho de la ventana o marco que se presenta la herramienta. Recomendado.

launch_presentation_height=240: Altura de presentación de la ventana o marco, que se presenta la herramienta. Recomendado.

launch_presentation_return_url=http://lmsng.school.edu/portal/123/page/988/: Redirige a página si se produce algún inconveniente con la TP. Mostrando un mensaje apropiado.

tool_consumer_info_product_family_code=desire2learn: Orientado a mejorar la herramienta con extensiones que permita una mejor interfaz con el usuario. Algunos parámetros posibles: learn, desire2learn, sakai, eracer, olat, webct. Recomendado.

tool_consumer_info_version=9.2.4: Numero de versión.

tool_consumer_instance_guid=lmsng.school.edu: Identificador único para la TC.

tool_consumer_instance_name=SchoolU: Texto desplegado al usuario normal. Recomendado.

tool_consumer_instance_description=University of School (LMSng): Texto desplegado al usuario normal. Parámetro opcional.

tool_consumer_instance_url=http://lmsng.school.edu: URL instanciado por el TC. Opcional.

tool_consumer_instance_contact_email=System.Admin@school.edu: Un dirección de mail para contacto de TC. Recomendado.

custom_keyname=value: Parámetro de seguridad para el enlace LTI.

Aquí se han listado los mensajes que se realizan en el intercambio en un lanzamiento Basic LTI, unos obligatorios, otros opcionales, y algunos recomendados. No se ha encontrado un parámetro con el cual se pueda enviar una cadena de caracteres en texto plano para uso general, o que posea una opción de crear un parámetro para el envío texto plano en el lanzamiento, sin estar definido.

¿Por qué no se implementó con CC y Basic LTI?

Si bien existe una especificación, a la hora de implementar y buscar información se encontraron algunos inconvenientes.

- CC no es el estándar más utilizado en crear objetos de aprendizaje, donde el acceso a información de este, se centra en la página oficial. Existen editores libres como lo es el editor eXe-learning que permite crear objetos, pero no estando actualizados a la hora de implementar nuevas funcionalidades, como por ejemplo Basic LTI.
- Considerando los parámetros listados de intercambio mediante el método post en un lanzamiento Basic LTI, no se ha encontrado uno que permita enviar texto plano, o definir algún parámetro que presente la posibilidad de enviar datos para nuestro interés.
- Se pretendió establecer contacto vía mail con IMS Global, obteniendo un mail de recibo automático, sin tener una respuesta posterior.
- Se verificó la existencia de un mecanismo de soporte para implementar paquetes Common Cartridge, siendo este la alianza CC/LTI. Ser miembro de la alianza (CC/LTI Alliance), implica cuotas monetarias, las cuales dependen de los fines de las empresas, es decir, si son con fines de lucro, sin fines de lucro, instituciones educativas. Estas cuotas anuales varían entre los U$S 250 (dólares americanos) hasta U$S 3000 (dólares americanos).
- El hecho de querer asegurar la no posibilidad de implementar un objeto de Aprendizaje mediante el protocolo SCORM para los requerimientos del proyecto, nos llevó más tiempo de lo esperado, acotando el tiempo disponible para el estudio de Common Cartridge y posible implementación mediante la especificación Basic LTI.
- El estudio en paralelo de una solución alternativa. Teniendo en cuenta la posibilidad de que no existiera una implementación mediante CC y Basic LTI al divisar inconvenientes como los nombrados anteriormente, llevó a decidir continuar con el estudio de CC y Basic LTI pero al mismo momento diseñar y desarrollar una solución alternativa: una aplicación web mediante el framework Web2py que ofrecía una línea más clara y permitía visualizar una solución que cumplía con los requisitos globales del proyecto (la corrección y asignación automática de calificación de tareas escritas en Python).

Se debe mencionar que en enero del 2013, se consultó la página web de IMS Global detectando una nueva versión de Basic LTI (versión 2.0), con fecha de publicación del documento 1 Noviembre del 2012, lo que hace notar nuevas actualizaciones en Basic LTI.

Por los motivos antes mencionados, además del avance en el proyecto, en cuanto a tiempo disponibles y teniendo una solución alternativa, con la cual podía ser implementada sin mayores inconvenientes, y cumpliendo con los objetivos, se decidió continuar por el camino alternativo.

VI.3. Conclusiones respecto a SCORM y Common Cartridge

El primer protocolo estudiado posee limitaciones con respecto al tipo de comunicación que se necesita establecer. Cuenta con una API básica para comunicarse con una plataforma de aprendizaje, pero no considera la comunicación fuera del objeto o de la plataforma que lo contiene. Este punto podría solucionarse modificando el protocolo, de manera de agregar la funcionalidad buscada, pero esta posibilidad escapa a los límites del proyecto.

Considerando el tiempo dedicado al estudio de SCORM, y tomando como referencia la estimación original de tiempos al inicio del proyecto, se puede afirmar que no se asignó en primera instancia un período tan prolongado como el que demandó, hecho que repercutió en el estudio del segundo protocolo.

En cuanto a Common Cartridge, se reconoce que se ha transitado por un camino que no era el esperado, sin lograr a obtener la profundidad de conocimientos deseada. Las razones principales atribuibles al hecho, son la relativa a tiempos ya mencionada, y las dificultades para acceder a información aplicable a nuestros objetivos. Respecto a esto último, no fue posible recopilar la información necesaria que permitiera utilizar este protocolo de la forma deseada, ya que la misma está limitada a personas y empresas pertenecientes a determinada organización, cuya membresía es paga. Se consideró, por lo tanto, que no estaba dentro de las posibilidades de un proyecto basado en herramientas libres y gratuitas, la utilización de este protocolo.

VII. Soluciones alternativas al objeto de aprendizaje

Habiendo descartado la posibilidad de construir un objeto de aprendizaje basado en un protocolo estándar, que implementara la corrección automática de tareas escritas en Python, se debió pensar en una solución alternativa.

Posteriormente a una etapa de análisis, se decidió que el camino a seguir sería la utilización del marco de trabajo Web2py, dados los buenos resultados obtenidos con el mismo hasta el momento (compilador web), para la creación de una aplicación web que cumpliera con los objetivos planteados inicialmente. Esto es, además de la corrección automática del código Python, la asignación de una calificación al alumno y el almacenamiento de la misma en una base de datos, la posibilidad de intercambiar información de registros con una plataforma de aprendizaje (por ejemplo Moodle), pero ser, al mismo tiempo, independiente de ella.

Se consideraron dos enfoques diferentes de solución. El primero, basado en el compilador web realizado para la primera parte de este proyecto, se orienta a que el alumno intente resolver en el aula una tarea de programación en el momento que el docente se la presenta. El alumno ingresa código Python a través de un área de texto con reconocimiento de sintaxis apropiado, obteniendo, al compilarlo, los resultados correspondientes y una calificación acorde. Tiene además la posibilidad de corregir su código las veces que considere necesario en un determinado tiempo de habilitación de la tarea.

El segundo enfoque, basado en sistemas de corrección existentes en cursos de la Facultad de Ingeniería, como Desarrollo de Software, está orientado a que el estudiante resuelva una tarea dentro de un período de tiempo más amplio, trabajando en su hogar y subiendo, desde allí u otro sitio con acceso a internet, un archivo, conteniendo código Python, al servidor donde se compilará. Se le informa entonces la calificación obtenida, pudiendo volver a subir la solución las veces que necesite durante el tiempo que permanezca habilitada la tarea.

Se pretendió probar el desarrollo de los dos enfoques de forma independiente entre sí, para lograr un par de prototipos con menos requerimientos cada uno y para poder evaluar su desempeño separadamente.

Se implementaron, por lo tanto, dos aplicaciones web distintas, que bien pueden ser complementarias o totalmente independientes.

VII.1. Webide 2

VII.1.1. Descripción

Es una aplicación web, basada en el marco de trabajo (o framework) Web2py, que permite la realización, corrección y calificación en línea, de tareas de programación escritas en el lenguaje Python.

Está pensada para utilizarse en el aula, aunque nada impide, a priori, que se utilice desde el hogar del estudiante, sin la presencia de un docente. En cualquiera de los casos, el alumno podrá ingresar a un sitio web donde verá la letra de la tarea a realizar, y contará con un área de texto para ingresar el código Python, que resuelva el problema que se le plantea. Es posible ingresar código todas las veces que se quiera o necesite, mientras esté habilitada la tarea. La habilitación o no de las tareas la establece el docente, mediante una interfaz destinada a ello. Cada vez que se envía el código a corregir, se le corren tests, cargados en la aplicación previamente por el profesor, y se le asigna una calificación al alumno, que se guarda en la tabla correspondiente en la base de datos.

La aplicación es independiente de cualquier plataforma de aprendizaje (como Moodle), pero puede intercambiar con ella información contenida en bases de datos, por ejemplo calificaciones, a través de archivos de tipo CSV.

Utiliza el sistema de autenticación que proporciona la herramienta Web2py, que incluye algunos controles de registro, como largo de contraseña mínimo o evitar registros con el mismo nombre de usuario, por ejemplo.

Webide2 basa su dinámica en la existencia de grupos de alumnos y de profesores. Se denominará de aquí en adelante como Profesor a alguien que pertenece al grupo de profesores y Alumno a quien pertenece al de alumnos. Dependiendo de qué tipo de usuario esté trabajando, se podrán ver o no determinadas páginas y utilizar o no ciertas funcionalidades. Cuando un usuario se registra, no pertenece a ningún grupo hasta que un Profesor lo incluye en alguno de los dos. El primer integrante del grupo de profesores viene por defecto con la aplicación.

VII.1.2. Funcionamiento

Experiencia de Usuario

Usuarios externos

Un usuario que no pertenece a ningún grupo, incluso si no está registrado, puede tener acceso a enlaces internos donde se presentan algunos artículos que describen características y conceptos relacionados a Python, y buenas prácticas de programación, particularmente en este lenguaje. Además se puede acceder a alguna información básica de la aplicación a través del enlace “Acerca de Webide2”.

En todas las páginas pertenecientes a la aplicación, existe una barra en la parte superior, donde se encuentra un botón “Home”, sobre la izquierda, que lleva a la página de inicio. Sobre la derecha hay dos enlaces para iniciar sesión o registrarse, pero una vez iniciada la sesión, en su lugar se ve un mensaje de bienvenida y enlaces para finalizar sesión, actualizar datos del perfil de usuario y modificar la contraseña. Esto es válido también para usuarios pertenecientes a alguno de los grupos.

A continuación se muestra la página de inicio. A la izquierda se observan los enlaces internos mencionados antes. A la derecha se muestran enlaces a páginas externas, disponibles para cualquier usuario, y que pueden ser de utilidad para quien esté realizando el curso al que está destinado esta aplicación, o cualquier entusiasta de la materia. En el centro se despliega una pequeña descripción de los pasos que debería seguir un estudiante para poder llevar a cabo, posteriormente, las tareas del curso. A ellas se accede, si se pertenece a alguno de los grupos, a través del botón “Comenzar!”.

FIGURA VII-1 – Página de inicio de Webide2.

Alumnos

Una vez registrado, el usuario será agregado, por parte de un Profesor y si corresponde, al grupo de alumnos. Ahora entonces pasa a ser un Alumno. Para registrarse deberá completar un pequeño formulario con algunos datos básicos. Como nombre de usuario se recomienda utilizar el número de la cédula de identidad sin el guión ni el dígito verificador, es decir, los siete primeros dígitos.

FIGURA VII-2 – Página de registro.

Habiendo iniciado sesión, el Alumno puede tener acceso a todos los enlaces comentados anteriormente, a los que podían acceder usuarios que no pertenecían a ningún grupo, y además tendrá privilegios para acceder a las páginas donde se presentan las tareas para resolver, sus calificaciones y posibles soluciones a los problemas. El inicio de sesión se realiza con el Nombre de usuario y contraseña. La siguiente imagen muestra la página de inicio cuando un Alumno ha iniciado sesión (se agregan enlaces en la parte inferior derecha de la pantalla).

FIGURA VII-3 – Página de inicio para un Alumno.

- Mediante el botón “Comenzar!” (o el enlace “Ir a las tareas”) se accede a una interfaz donde se puede elegir la tarea a realizar. Luego de seleccionarla y presionar el botón “Ir a esta tarea”, se redirige al usuario a una nueva página. Si la tarea no está habilitada, la redirección se hace hacia una página que informa al usuario que no está autorizado a ingresar a donde solicitó. Si la tarea elegida está habilitada, se redirige a tareaX.html, donde se presenta la letra del ejercicio a resolver y un área de texto con reconocimiento de sintaxis Python, para ingresar el código que procure dar solución al problema planteado.

FIGURA VII-4 – Elección de tarea a realizar.

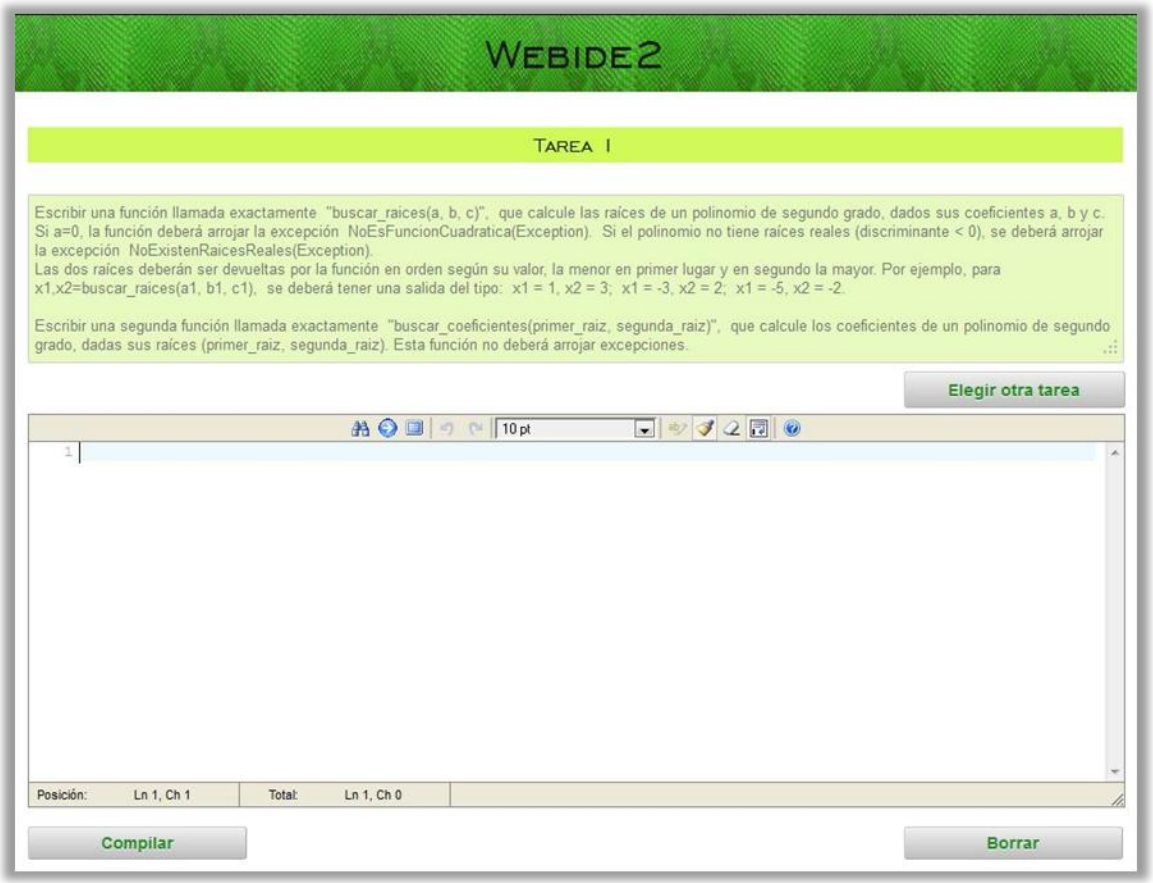

FIGURA VII-5 – Página donde se presenta una tarea.

- El botón “Elegir otra tarea” permite volver a la página anterior donde se selecciona la tarea a realizar. Con “Borrar” se elimina todo lo escrito en el área de texto. Mediante el botón “Compilar” se envía el código ingresado a corregir

y se avanza a la página de resultados. En ésta se informa al usuario cuántos tests le dieron correctos y, en base a ello, cuál fue su calificación en términos de porcentaje. Si este puntaje corresponde al máximo (100%), se desplegará un mensaje flash en la parte superior derecha de la pantalla que dirá "Perfecto!". Si el código tuvo errores de compilación, se informará cuál fue y se calificará la tarea con cero puntos. Se desplegará también un mensaje flash informando, esta vez, que hubieron errores de compilación. En ambos casos se muestra, además, el código ingresado, sobre la derecha de la pantalla. Si el código presenta algún bucle infinito o demasiado extenso, se cortará la ejecución del mismo, se verá un nuevo mensaje flash, indicando que se ha agotado el tiempo de espera, y se redirigirá a una página que informa el error de tiempo de espera agotado (o "timeout"). Se asignará nuevamente una calificación igual a cero. A continuación se presentan imágenes de las tres posibilidades.

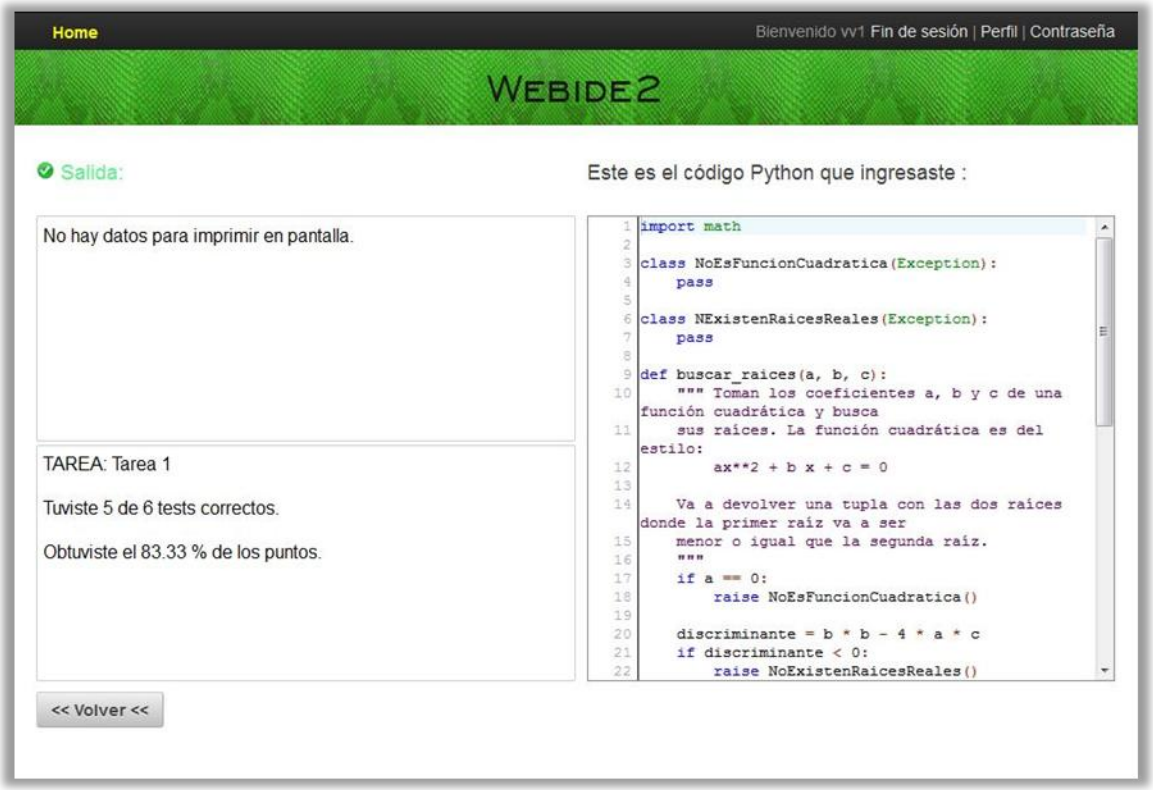

FIGURA VII-6 – Página de resultados al correrse los tests correctamente.

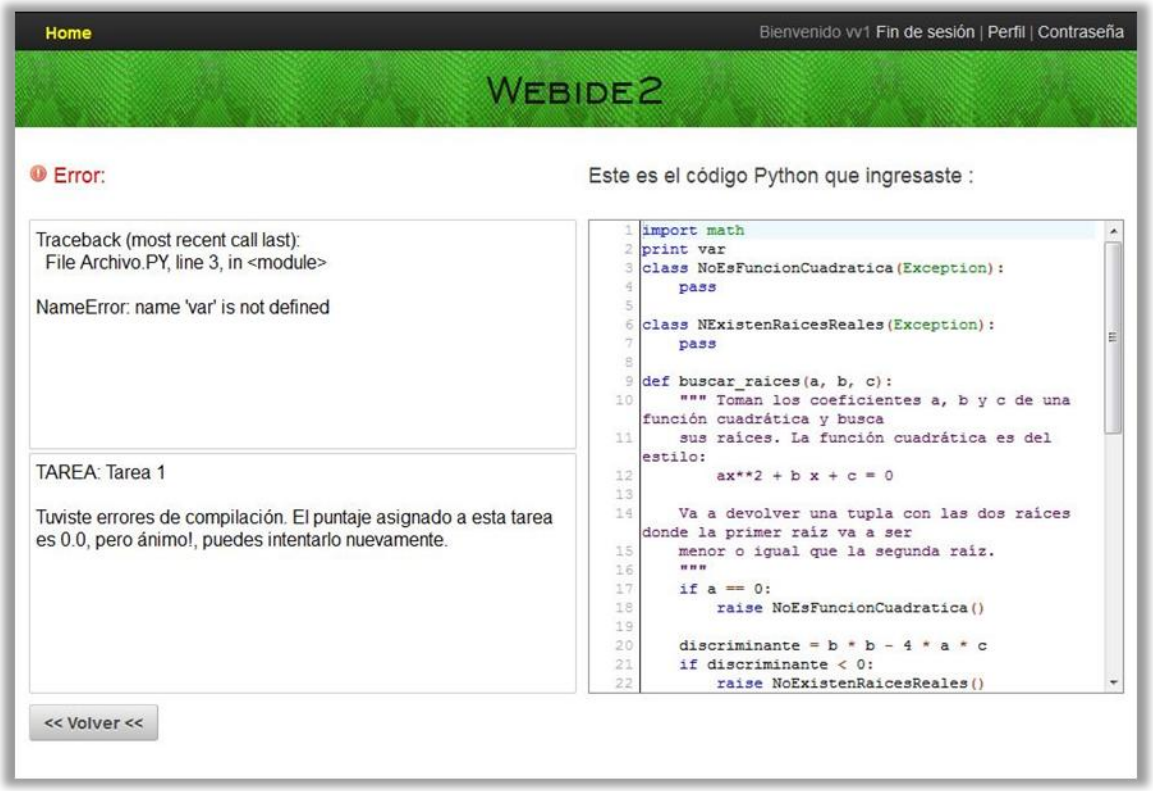

FIGURA VII-7 – Página de resultados cuando hay errores de compilación en el código ingresado.

FIGURA VII-8 – Error de timeout.

- El enlace en el Inicio, “Ver soluciones”, permite ingresar a una página donde se puede seleccionar una tarea para ver una solución a la misma, si fue cargada por un docente previamente y si está habilitada para Alumnos. En caso de no existir o no estar habilitada, se informará al usuario.

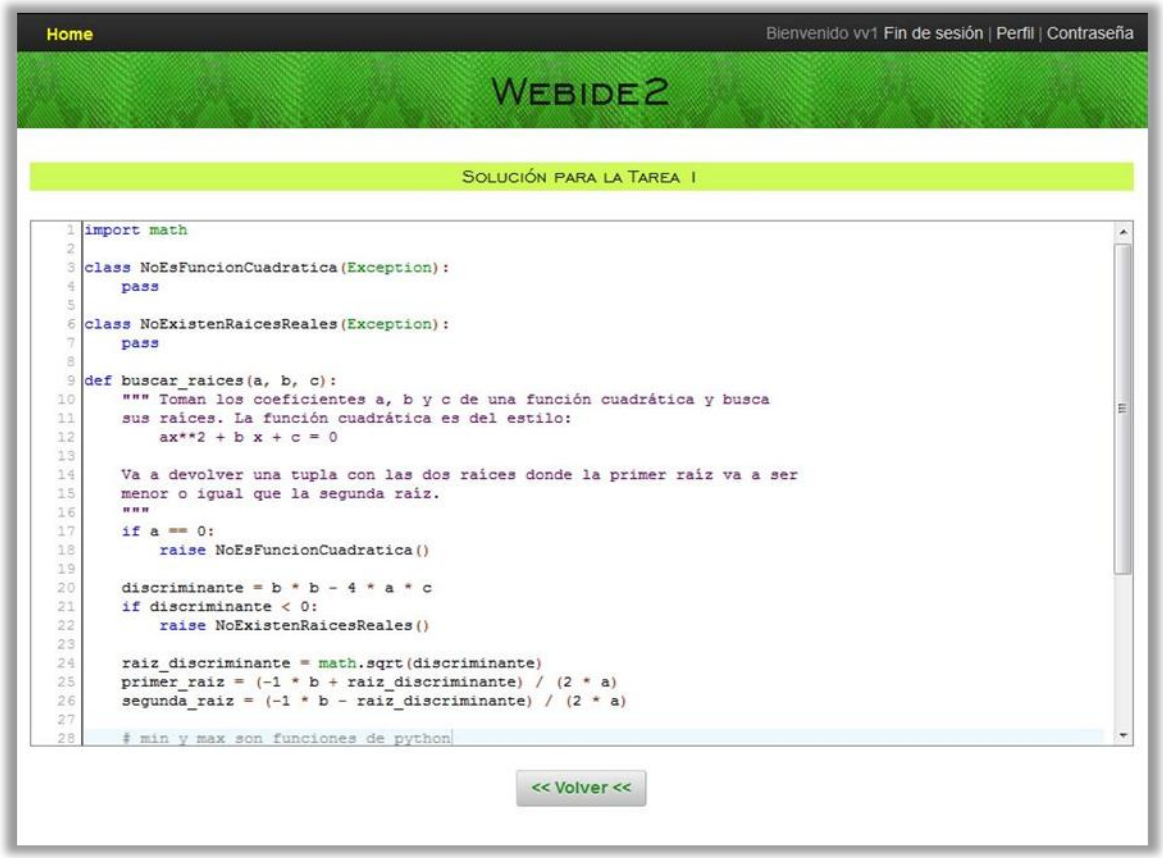

FIGURA VII-9 – Vista de la solución a una tarea.

- Por medio del enlace “Ver mis calificaciones”, también en el Inicio, el estudiante puede consultar las notas de las tareas realizadas. Las tareas que aún no ha intentado realizar, tendrán la palabra “None” en lugar de la nota.

FIGURA VII-10 – Vista del Alumno vv3 dd3 de sus calificaciones.

- Accediendo por “Ayuda”, se encontrará un manual básico de funcionamiento para Alumnos.

Profesores

Si un usuario registrado es agregado al grupo de profesores por parte de un Profesor, también será un Profesor.

FIGURA VII-11 – Página de inicio para un Profesor.

El Profesor tiene acceso a todos los enlaces mencionados antes, pudiendo ejecutar código que será corregido igual que el de un Alumno, a modo de prueba y sin que quede registrada en ningún lugar la calificación obtenida. La diferencia en la vista de la tarea para un Profesor es que ahora aparece la información de fechas y horas a las que se habilitará y deshabilitará la tarea, y a la que se publicará la solución para la misma, más un enlace que lleva a la interfaz donde se modifican dichos horarios. Además tiene acceso al resto de los enlaces que se ven en la FIGURA VII-11, que no aparecían para los demás usuarios, y que se describen a continuación.

- Ingresando a “Tareas”, el Profesor accederá a una página que le permite elegir una tarea para consultar, modificar o subir por primera vez. Luego de seleccionarla y presionar el botón “Consultar/Modificar esta tarea”, verá un formulario con un par de áreas de texto para ingresar la letra del ejercicio y los tests, que corregirán los posibles códigos que intentarán dar solución al problema. Es posible ingresar solo uno de los campos a la vez, dejando el otro vacío, de manera de poder subir letra y tests en instancias diferentes. Una vez que la página de la tarea esté habilitada para los Alumnos, deberá existir tanto letra como tests. Es posible eliminar una tarea previamente cargada desde esta página. Para ello se debe borrar el texto que aparezca

por defecto en ambas áreas de texto, dejándolas en blanco, y presionar el botón "Enviar datos". Las dos páginas mencionadas se muestran a continuación.

FIGURA VII-12 – Elección de tarea a subir o consultar.

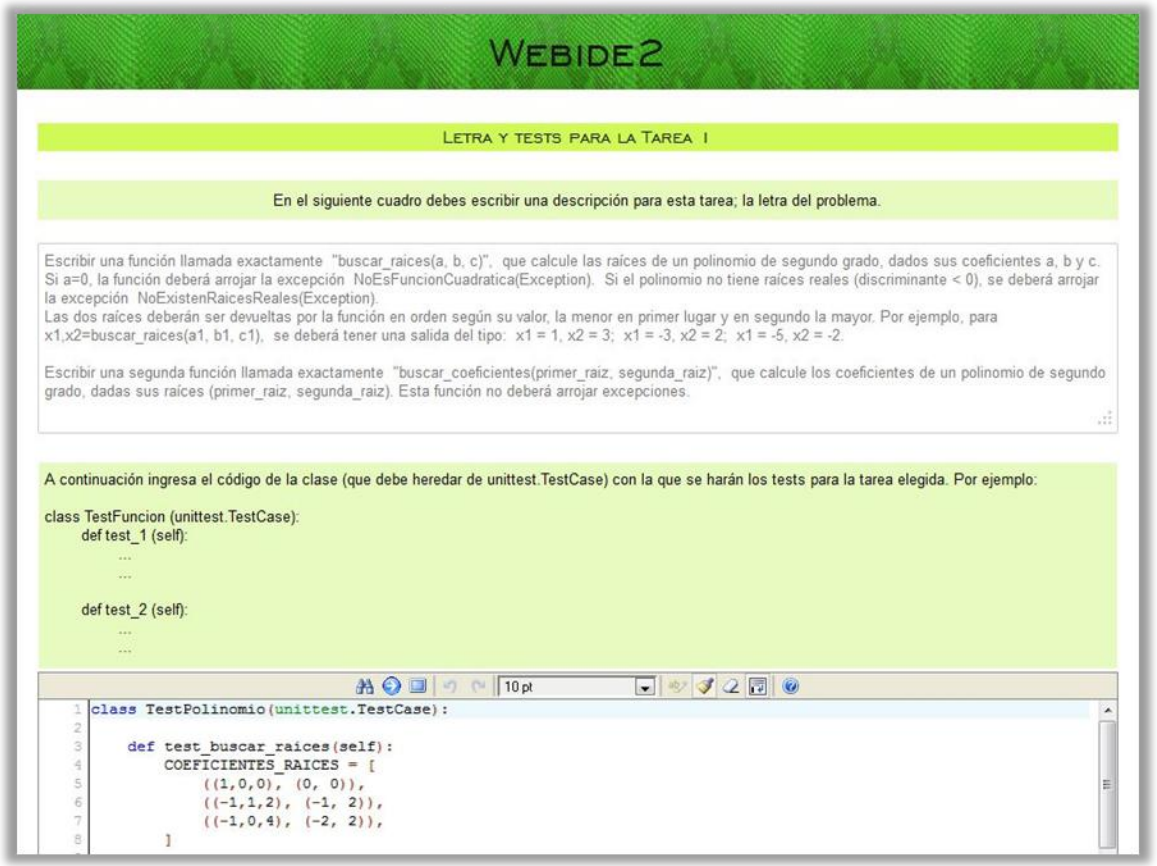

FIGURA VII-13 – Página para subir, consultar o eliminar tareas.

- De la misma forma se mostrará, ingresando en “Soluciones”, una página para seleccionar una tarea y posteriormente un formulario para cargar una posible solución a la misma en la aplicación, es decir, un ejemplo de lo que podría escribir un Alumno para obtener el máximo puntaje en esa tarea. Es posible eliminar la solución consultada, borrando el texto que aparezca cargado en el formulario, dejándolo en blanco, y presionando el botón “Enviar datos”. Se muestra a continuación la página donde se sube, consulta o elimina una solución.

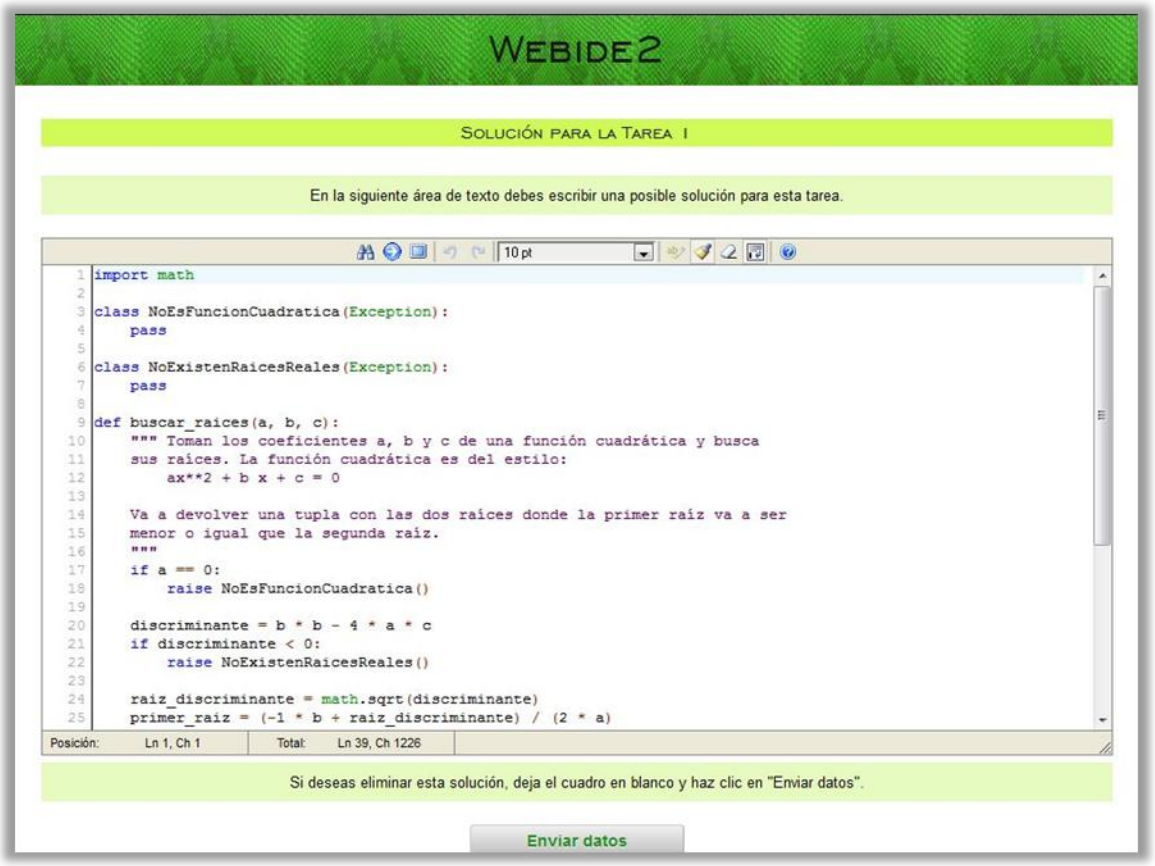

FIGURA VII-14 – Página para subir, consultar o eliminar soluciones.

FIGURA VII-15 – Administración de registros.

- El enlace “Administración de registros” lleva a una página (FIGURA VII-15) donde se encuentran varios enlaces más que permiten la consulta y modificación de las tablas en la base de datos y de archivos guardados en el servidor. En este último caso, solo para ver qué archivos existen y poder eliminarlos si se desea. Los enlaces y sus páginas destino se describen a continuación. Todas ellas cuentan con un botón para volver a “Administración de registros”.
    - “Administrar membresías”. Aquí se muestran todos los usuarios que pertenecen a uno de los dos grupos definidos. Un Profesor puede agregar (botón “Agregar”) o eliminar (botón “Eliminar” junto a cada registro) usuarios de los grupos de alumnos y profesores. Existe la posibilidad de exportar la tabla presentada, en varios formatos, en particular CSV, mediante los botones dispuestos bajo la tabla junto a “Exportar”. El botón “Borrar tabla” elimina todos los usuarios del grupo de alumnos (no modifica el de profesores), pasando antes por una página de advertencia que da la posibilidad de confirmar la acción o de volver sin hacer cambios. “Borrar tabla”, está pensado para vaciar la tabla una vez que finaliza el curso. De similar manera, con el botón “Agregar todos”, se agregan todos los usuarios registrados que no pertenecen a ningún grupo, al grupo de alumnos. Está pensado para cuando se inicia un nuevo curso y ya se han registrado todos los estudiantes.

FIGURA VII-16 – Administración de membresías.

FIGURA VII-17 – Página de confirmación de la opción "Borrar tabla".

- "Administrar horarios". En esta página se agregan, editan y eliminan las fechas y horarios de inicio, cierre y publicación de la solución, de cada tarea. También se pueden eliminar todos los horarios a la vez con el botón "Borrar tabla", con previa advertencia. Los horarios asignados son desplegados en una tabla como se muestra en la figura siguiente, y se tiene, también aquí, la posibilidad de

exportarla a archivos de distintos formatos. Las tareas estarán habilitadas para los Alumnos, si la fecha y hora actual se encuentra entre la de inicio y final; y las soluciones lo estarán si la fecha y hora actual es posterior a la de publicación.

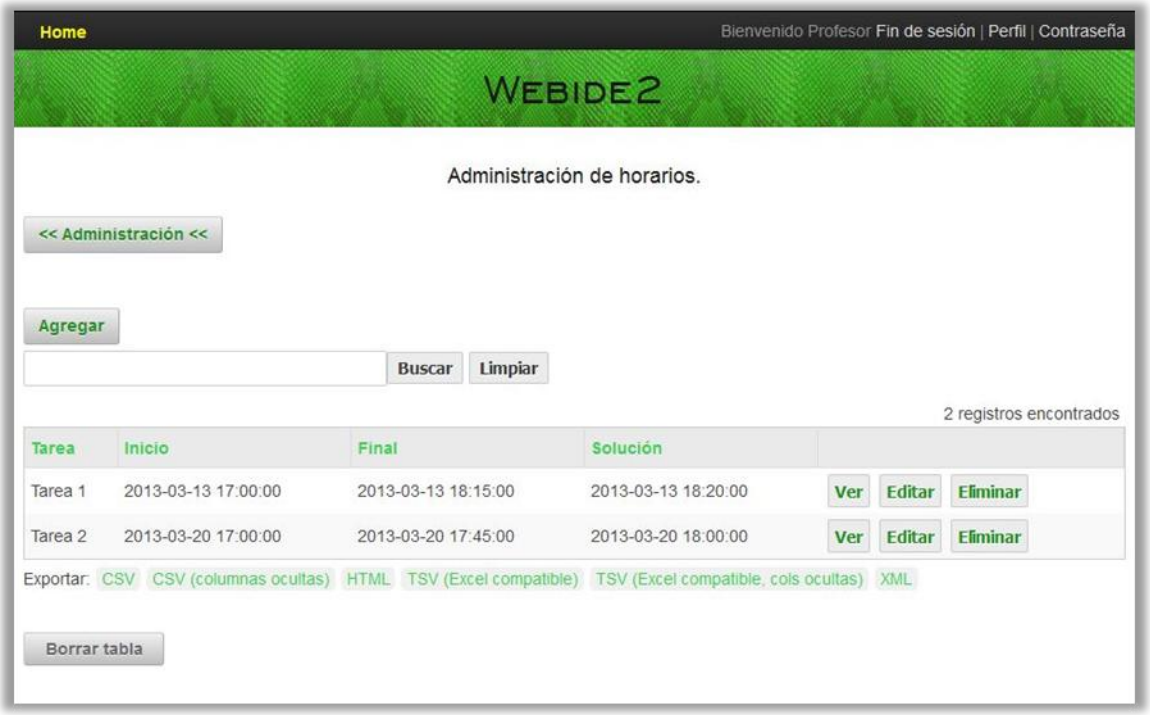

FIGURA VII-18 – Administración de horarios.

- "Administrar calificaciones". Las calificaciones obtenidas por los Alumnos son mostradas en una tabla similar a las anteriores, pudiéndose exportar también a un archivo de formato CSV, entre otros. De la misma manera que los puntos descritos antes, existen botones para agregar, editar y eliminar registros, incluso eliminar todos a la vez. Se muestran además los promedios de las calificaciones de los Alumnos, pero no son calculados de forma automática, sino que se da la posibilidad al Profesor de elegir entre qué tareas quiere estimar el promedio. Para ello deberá presionar el botón "Calcular promedios", que lo redirige a una página donde debe ingresar las tareas que serán el principio y el final de las consideradas para el cálculo, que se hará al dar clic en "Calcular".

FIGURA VII-19 – Administración de calificaciones.

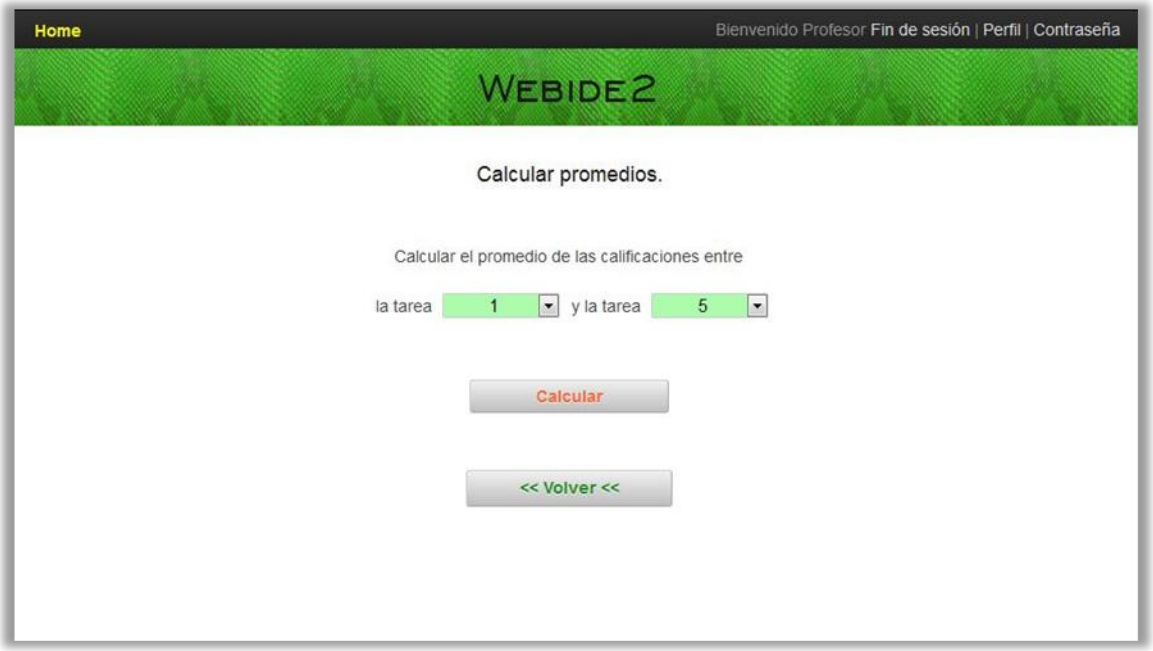

FIGURA VII-20 – Página para el cálculo de promedios.

- "Administrar usuarios". Así como en las páginas de administración anteriores, aquí se pueden consultar, agregar, editar o eliminar registros. En este caso los mismos corresponden a los datos personales de todos los usuarios registrados. En particular un Profesor puede acceder a modificar las contraseñas de los

demás usuarios, en caso de que sea necesario por pérdida de la misma, por ejemplo. En las dos figuras siguientes se muestra la vista de la página "Administración de usuarios" y la de la edición de un registro.

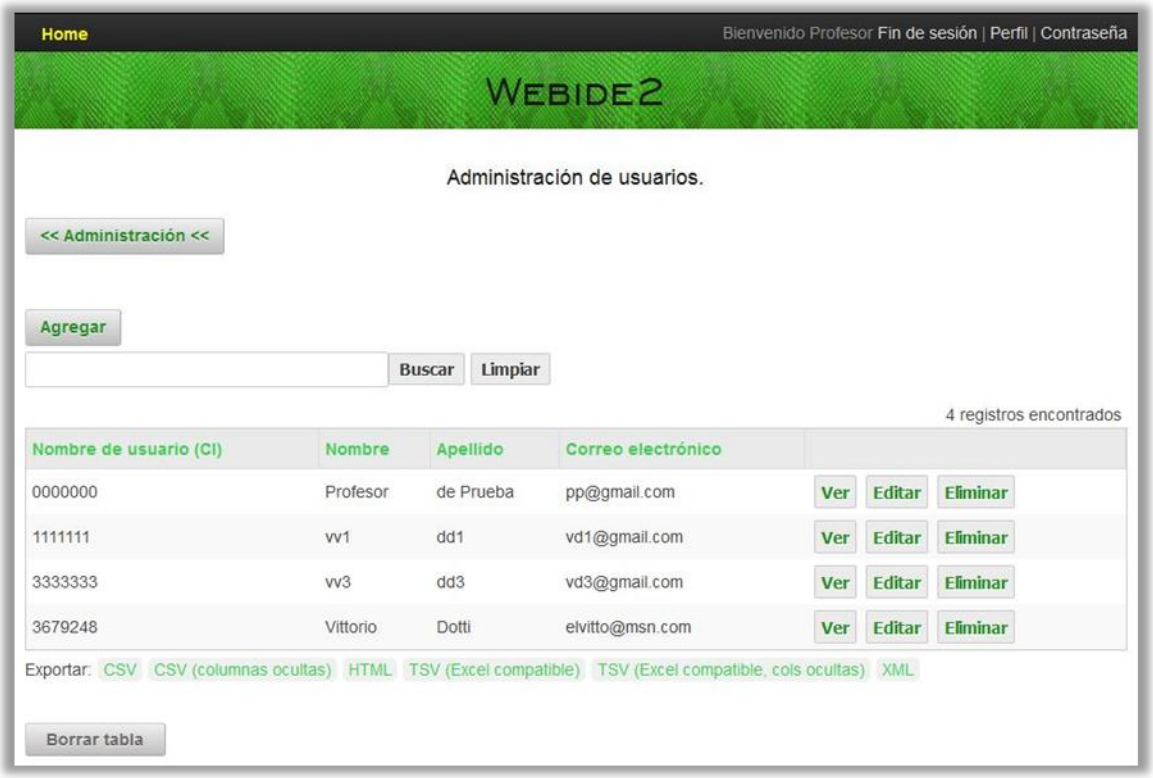

FIGURA VII-21 – Administración de usuarios.

FIGURA VII-22 – Edición de datos de un usuario.

- "Administrar respaldos". Cuando un Alumno ingresa y ejecuta cierto código como solución a una tarea, se guarda una copia en el servidor (que se sobrescribe para el mismo nombre de usuario y número de tarea). Esta página permite ver qué archivos hay en el servidor y borrarlos si se desea (todos al mismo tiempo). Esto está pensado para cuando finaliza el curso y se quiere "limpiar" el servidor.
- "Administrar tareas". Tiene la misma función que la página anterior, pero en este caso los archivos guardados que pueden eliminarse contienen las letras, tests y posibles soluciones de las tareas, subidos por un Profesor en algún momento.

FIGURA VII-23 – Administrar respaldos.

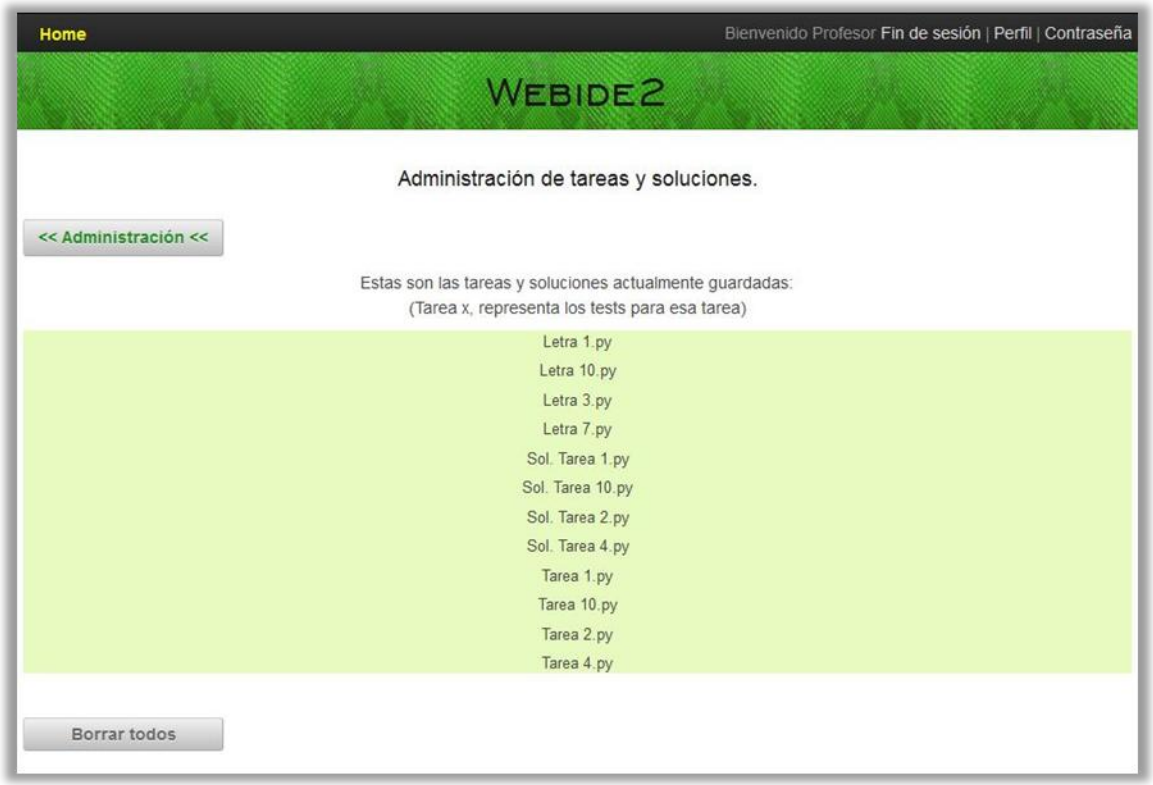

FIGURA VII-24 – Administrar tareas.

- Por medio del enlace “Consultar respaldos” es posible ver y probar, como si se estuviera realizando una tarea, el último código ingresado por un Alumno para una determinada tarea, el cual se almacena con fecha y hora de guardado. En primera instancia se redirige a una página donde se deberá ingresar el nombre de usuario (cédula de identidad) del Alumno y seleccionar la tarea que se desee consultar. Mediante el botón “Consultar” se accede a una nueva página, que es básicamente la que se observa cuando se está realizando la tarea, pudiéndose ejecutar el código de la

misma manera, y obteniendo el informe de resultados correspondiente, pero sin que se modifique ninguna tabla en el proceso, en particular la calificación del Alumno.

FIGURA VII-25 – Consulta de respaldos.

FIGURA VII-26 – Visualización del código consultado, con fecha y hora de almacenado.

- Accediendo por “Ayuda”, se encontrará un manual básico de funcionamiento para Profesores.

Flujo de trabajo

Tomando como punto de partida que los usuarios ya están registrados y agregados a los grupos correspondientes (Alumno o Profesor), el flujo básico de trabajo es el siguiente (no se toman en cuenta las funciones de tipo administración de registros, consulta de respaldos, etc.):

- Profesor:
  - Carga en la aplicación la letra de la tarea y los tests que la corregirán.
  - Carga (opcionalmente) una posible solución para la misma.
  - Establece las fechas y horarios entre los que la tarea estará habilitada para los Alumnos, y la fecha y hora de publicación de la solución (si existe).
- Alumno:
  - Ingresa a través del botón “Comenzar!” a la página de elección de tareas y elige la que le corresponda realizar.
  - En la página de la tarea seleccionada, lee la letra del problema y escribe código Python, definiendo e implementando las funciones que se le indiquen en la letra.
  - Hace clic en el botón “Compilar” y observa los resultados obtenidos.
  - Con el botón “Volver” regresa a la página de la tarea, donde modifica su código y lo compila nuevamente, o regresa a la página de elección de tareas mediante el botón “Elegir otra tarea”.

VII.1.3. Características generales y funcionales

Ventajas generales

- Es una aplicación libre y de código abierto.
- Está autocontenida y es independiente de otros recursos.
- De ser necesario, posee la facilidad de intercambiar información con plataformas de aprendizaje (por ej. Moodle) a través de archivos de formato CSV, TSV, XML y HTML.
- Tiene pocos requisitos para su instalación y funcionamiento. Básicamente se necesita un intérprete Python y Web2py instalados en el servidor donde vaya a funcionar.
- No depende del sistema operativo de dicho servidor.
- Es reusable. Solo se deben vaciar los registros al iniciar un nuevo curso (existen interfaces apropiadas para eso).
- Es una herramienta durable. No se visualizan motivos para su obsolescencia a corto y mediano plazo.
- Establece una idea primaria para la realización de herramientas similares, que podrían ser utilizadas en cursos de otros lenguajes de programación.

Ventajas funcionales

- Accesos rápidos a páginas internas y externas que pueden ser de utilidad a quien utilice la aplicación.

- Reconocimiento de sintaxis Python al tiempo que el usuario va escribiendo código en este lenguaje, en las áreas de texto destinadas a ello.
- Corrección automática de las tareas sin que el docente intervenga más que para cargar los tests que evalúan el código a corregir.
- No obstante, el profesor puede ver el código del estudiante, si lo desea, ya que se guarda una copia del último código escrito por el alumno.
- La corrección se lleva a cabo en tiempo real. El estudiante sabe inmediatamente de compilar su código, la calificación obtenida (pudiendo intentar mejorarla las veces que sea necesario mientras la tarea esté habilitada).
- Habilitación y cierre automático de las tareas, así como publicación automática de soluciones a las mismas, una vez establecidos los horarios por parte del docente.
- Interfaces amigables e intuitivas para los usuarios.

Limitaciones generales

- Se trata de una primera versión de la aplicación, por lo que no están optimizados el diseño ni el código con los que se ha programado.

Limitaciones funcionales

- La cantidad de tareas es fija. El docente no puede crear tareas (tampoco el administrador), solamente agregar letra y tests a las existentes. La cantidad mencionada se establece en un número que podría ser apropiado (diez) para el desarrollo de un curso, pero si se necesitara, no sería posible utilizar más tareas, sin modificar el código de programación; cuando se utiliza una cantidad menor al total establecido, existen tareas de más, subutilizándose, por lo tanto, recursos de la aplicación.
- Si se deseara, no existiría la posibilidad de interacción o comunicación entre personas involucradas. Por ejemplo, no hay un espacio para consultas al docente o foros de discusión.
- No se implementa ningún sistema antiplagio automático. En caso de querer comparar el código de dos alumnos, el docente deberá hacerlo él mismo, consultando los códigos almacenados por la aplicación.

VII.1.4. Implementación

Diagrama de componentes

Ya se ha descrito el paradigma de programación Modelo-Vista-Controlador (MVC), por lo que se hacen referencias a él o sus componentes naturalmente.

En la siguiente figura se presenta un diagrama de módulos de la aplicación. Como se mencionó en Limitaciones generales, el diseño no está optimizado, y puede apreciarse notando que existe solamente un módulo en el Controlador (default.py).

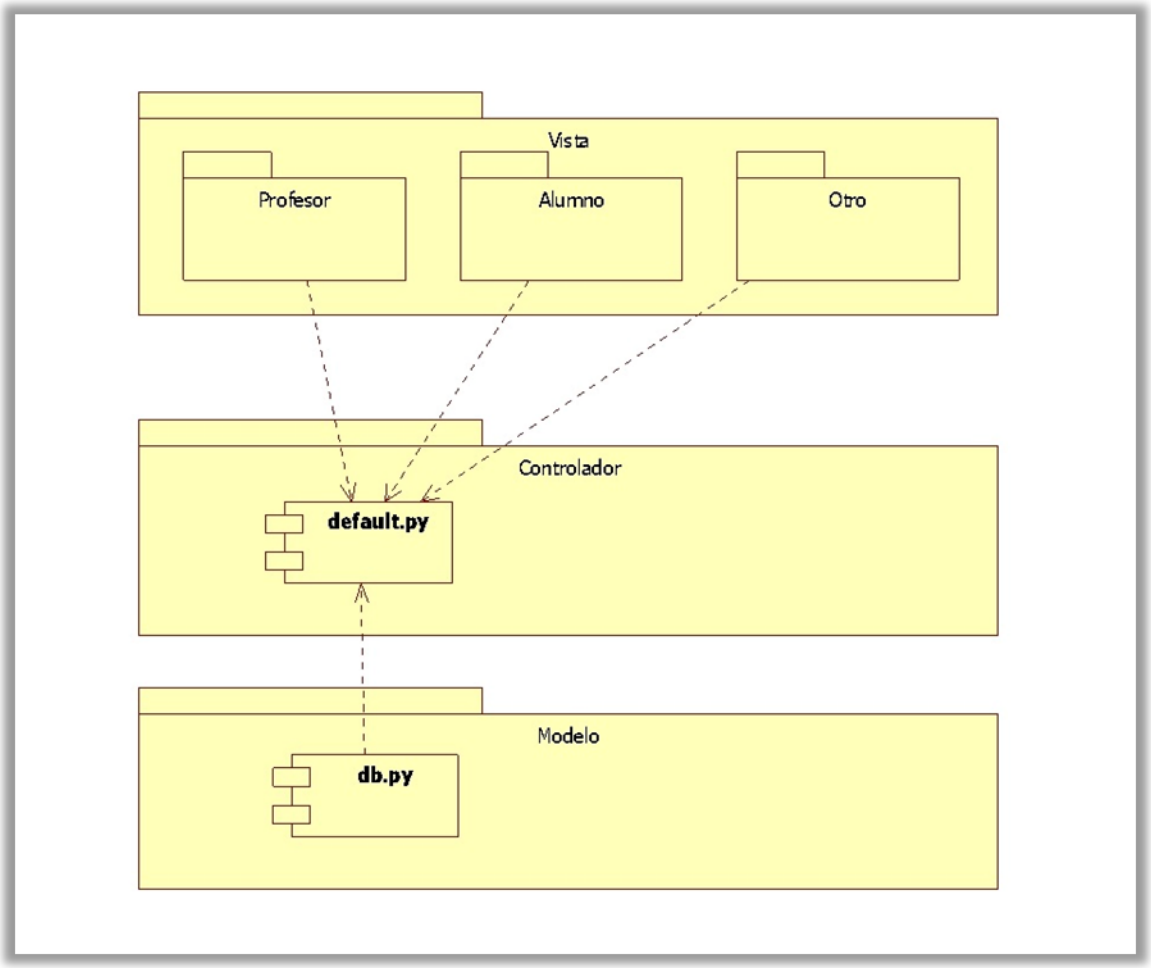

FIGURA VII-27 – Diagrama de componentes Webide2 – 1.

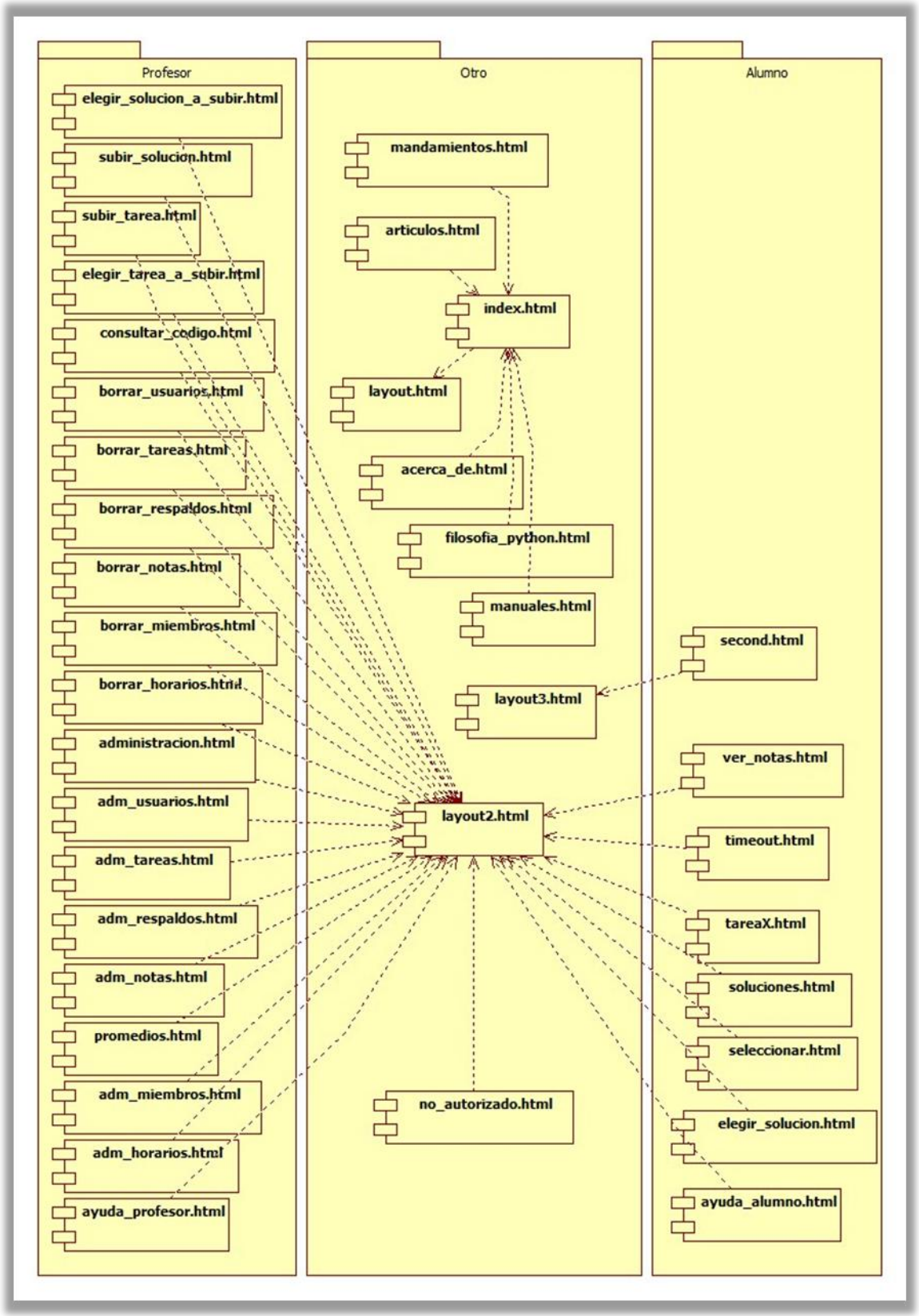

FIGURA VII-28 – Diagrama de componentes Webide2 – 2.

Se muestran a continuación las tablas en la base de datos del Modelo, los archivos pertenecientes a la Vista, y todas las funciones utilizadas en el Controlador, organizadas dependiendo a la capa de MVC que corresponda.

Modelo	Vista	Controlador
En /models/db.py:	*En /views/default:*	*En /controllers/default.py:*
db.auth_user	index.html	index()
db.auth_group	mandamientos.html	mandamientos()
db.auth_membership	filosofia_python.html	filosofia_python()
db.Nota	articulos.html	artículos()
db.Hora	acerca_de.html	acerca_de()
	manuales.html	manuales()
	ayuda_profesor.html	ayuda_profesor()
	ayuda_alumno.html	ayuda_alumno()
	second.html	second()
	timeout.html	timeout()
	elegir_tarea_a_subir.html	elegir_tarea_a_subir()
	subir_tarea.html	subir_tarea()
	elegir_solucion_a_subir.html	elegir_solucion_a_subir()
	subir_solucion.html	subir_solucion()
	elegir_solucion.html	elegir_solucion()
	soluciones.html	soluciones ()
	seleccionar.html	seleccionar()
	ver_notas.html	ver_notas()
	tareaX.html	tareaX()
	administracion. .html	administracion()
	adm_usuarios.html	adm_usuarios()
	adm_miembros.html	adm_miembros()
	adm_notas.html	adm_notas()
	adm_horarios.html	adm_horarios()
	adm_respaldos.html	adm_respaldos()
	adm_tareas.html	adm_tareas ()
	promedios.html	promedios()
	borrar_notas.html	borrar_notas()
	borrar_horarios.html	borrar_horarios()
	borrar_usuarios.html	borrar_usuarios()
	borrar_miembros.html	borrar_miembros()
	borrar_respaldos.html	borrar_respaldos()
	borrar_tareas.html	borrar_tareas()
	consultar_codigo.html	consultar_codigo()
	layout.html, layout2.html, layout3.html	corregir(resultado)
	user.html *(Por defecto en Web2py)*	calificar(tarea, nota)
	En /views:	ejecutar(texto, tarea)
	init.py, web2py_ajax.html, user.html, appadmin.html *(Por defecto en Web2py)*	matar()
		guardar_copia(cod, tarea)
		tarea(nro_tarea)
		verificar(ruta_letra)

		fecha_actual_mayor_a(fecha)
		fecha_actual_menor_a(fecha)
		fechas(nro)
		separar(fecha)
		user(), download(), call(), data() *(Por defecto en Web2py)*
		En /controllers/appadmin.py:
		get_databases(request), eval_in_global_env(text), get_database(request), get_table(request), get_query(request), query_by_table_type(tablename, db, request=request), index(), insert(), download(), csv(), import_csv(table, file), select(), update(), state(), ccache() *(Por defecto en Web2py)*

Tabla VII-1 – Tablas, archivos y funciones de Webide2.

Las distintas capas interactúan entre sí de manera de procesar información, consultar o actualizar la base de datos y mostrar los resultados al usuario a través de su navegador web.

Funciones y tablas principales

Se describen en forma breve las tablas y funciones principales de la aplicación.

En el Modelo, las tablas en la base de datos (nombrada como “db”) que trae por defecto Web2py y serán utilizadas para el funcionamiento de la aplicación son:

- db.auth_user. Contiene los datos (nombre, cédula de identidad, contraseña de acceso, e-mail) de todos los usuarios registrados.
- db.auth_group. Almacena los grupos Alumno y Profesor. Esta tabla no se modificará.
- db.auth_membership. Contiene la información de a qué grupo pertenece cada usuario (si pertenece a alguno).

Además se agregaron:

- db.Nota. Guarda el registro de calificaciones de los Alumnos.
- db.Hora. Almacena la información de fechas y horarios de inicio, final y publicación de solución de las tareas.

En el Controlador se definen varias funciones que permiten, por ejemplo, cargar y consultar las letras de las tareas, tests y soluciones, consultar y modificar bases de datos, etc. (más información disponible en documentos anexos donde se encuentra el código fuente); pero se destacan a continuación las que dan la lógica de funcionamiento principal a la aplicación, es decir, las que permiten ingresar código para resolver una tarea, que el mismo sea corregido por los tests previamente cargados, y asignar, en base a lo anterior, una calificación al Alumno.

- TareaX(). Esta función muestra (FIGURA VII-5), junto a su Vista asociada, la tarea X (X representa un número entre el 1 y el máximo de tareas, que por defecto será 10) al Alumno (o Profesor, si lo desea). Previo a esto, se verifica que la hora actual se encuentre entre la de inicio y de final de la tarea, o dicho de otra manera, que la tarea esté habilitada. Si se cumple esta condición, entonces se busca la letra del problema en el archivo donde se almacena, para desplegarla al usuario, y posteriormente se llama a la función Tarea(nro_tarea), que es la que crea el formulario para el ingreso de código y se encarga de invocar al resto de la funciones necesarias para el procesamiento de los datos. La función TareaX() distingue si el usuario que intenta acceder a la Vista es un Alumno o Profesor. En este último caso no importa que la tarea esté habilitada o no, un Profesor siempre puede verla.
- Tarea(nro_tarea). Recibe como entrada el número de tarea (en realidad una cadena de caracteres de la forma "Tarea X") a la que se quiere acceder. Crea un formulario en el que posteriormente se ingresará el código Python. Luego de ingresado el código y de validado el formulario, guarda una copia en el servidor del código y llama a la función Ejecutar(texto, tarea), que lo procesará.
- Ejecutar(texto, tarea). Recibe como entradas el código ingresado y una cadena de caracteres de la forma "Tarea X" que indica el número de tarea que se está realizando. Crea un archivo temporal en el servidor, con extensión .py, que se borrará al finalizar la ejecución de la función. En él se escribe el código ingresado por el usuario, más los tests que fueron previamente cargados por un Profesor, más un texto fijo que sirve para correr los tests e identificar luego los resultados en la salida estándar. De no encontrarse los tests, no se podrá corregir la tarea, por lo que se mostrará un mensaje apropiado en lugar de los resultados. A continuación se ejecuta el archivo .py creado. Si el código presenta algún bucle infinito o la ejecución se hace demasiado extensa (se controla con un temporizador implementado con threading.Timer()) de manera que pueda hacer que la aplicación deje de responder, se corta la ejecución mediante la función Matar(), que se encuentra dentro mismo de la función que se está describiendo, y se redirige a una página informando el error de tiempo de espera agotado ("timeout"). Si esto no ocurre y se lleva a cabo una ejecución "normal", se analizan las salidas estándar y de errores. Si esta última presenta mensajes de errores de compilación, se llama a la función Calificar(tarea, nota), con una calificación igual a cero; de lo contrario se llama a la función Corregir(resultado) para que analice la salida estándar, donde se han puesto palabras clave (mediante la porción de texto fijo mencionada antes) que permiten identificar la cantidad de tests corridos, aprobados y fallados. Con estos resultados se llama a la función Calificar(tarea, nota), para asignar la calificación correspondiente al Alumno. Se borran de la salida estándar y de errores lo que pueda estar agregado por la aplicación (debido al funcionamiento descrito, pues en la salida estándar se agrega lo ya mencionado, y en la salida de errores se escribe el resultado de los tests que da Python cuando se utiliza su módulo unittest) y se redirige a la página de resultados, donde se muestra al usuario su código, la salida correspondiente a la ejecución del mismo y la cantidad de tests aprobados junto a la calificación correspondiente.
- Corregir(resultado). Toma como entrada la salida estándar (o la "standard error" si la versión del intérprete Python del servidor es menor a 2.7, pues en esas versiones

inferiores no es posible escribir en la salida estándar en la misma ejecución del archivo, luego de correr los tests) donde se encuentran los resultados de los tests. Encuentra cuántos tests fueron corridos, aprobados y fallados, y efectúa el cálculo de la calificación en términos de porcentaje. Devuelve este valor, junto a la cantidad de tests fallados y corridos.

- Calificar(tarea, nota). Recibe como entrada una cadena de caracteres del tipo "Tarea X" y la calificación a asignar. Si el usuario que está autenticado es un Alumno, entonces inserta o actualiza el registro correspondiente en la tabla db.Nota, con el valor que se indique en "nota". Si el usuario es un Profesor, no se modifica ningún valor en ninguna tabla. De esta forma, un Profesor puede probar código sin que se afecte ningún registro.

Descripción del proceso básico

Considerando el flujo de trabajo básico para el Profesor y el Alumno, se describe el proceso con el que se lleva a cabo (no se toman en cuenta las funciones de tipo administración de registros, consulta de respaldos, etc.):

- Para el Profesor:
  - Mostrar página de bienvenida con enlaces a páginas para subir tareas, soluciones y horarios de tareas.
  - Si se ingresó a subir tareas o soluciones:
    - Mostrar áreas de texto para el ingreso de letra o código correspondiente.
    - Tomar la información ingresada, crear archivos para cada una de las tres opciones (letra, tests, solución) y almacenarlos en el servidor, en un directorio adecuado dentro del de la aplicación.
  - Si se ingresó a subir horarios:
    - Mostrar tabla con horarios ya asignados, dando la posibilidad de modificarlos y agregar nuevos.
    - Almacenar los cambios efectuados o los horarios nuevos agregados.
- Para el Alumno:
  - Mostrar página de bienvenida con enlace a página de selección de tareas.
  - Desplegar página de elección de tareas y redirigir a la página de la tarea seleccionada.
  - Verificar si la tarea elegida está habilitada.
  - Si la tarea no está habilitada: No permitir el acceso a la misma, informar al Alumno y dar la posibilidad de volver a la página de bienvenida.
  - Mostrar la letra de la tarea y un área de texto para ingresar el código solución.
  - Capturar la entrada dada por el usuario.
  - Guardar una copia del código ingresado en un archivo en el servidor, en un directorio adecuado dentro del de la aplicación.
  - Crear un archivo con extensión .py, formado por el código ingresado por el Alumno, los tests cargados previamente por el Profesor y por una porción de código fijo que permita correr los tests y obtener en la salida (al ejecutar el archivo) la cantidad de tests corridos y sus resultados.

- Ejecutar dicho archivo con el intérprete Python del servidor donde se encuentra la aplicación.
- Si la ejecución lleva demasiado tiempo, de manera que la aplicación deja de responder (por ejemplo si se da un bucle infinito en el código ingresado):
    - Cortar la ejecución y redirigir a una página donde se informe sobre el error de tiempo de espera agotado (o "timeout") y que se asignará una calificación igual a cero.
    - Calificar al Alumno (guardar registro en la tabla destinada a ello) con cero puntos en la tarea correspondiente.
- Capturar la salida y los errores provenientes de la ejecución en variables apropiadas.
- Si no hay errores:
    - Obtener de la salida la cantidad de tests corridos y fallados.
    - Calcular un puntaje (en porcentaje) en base a los resultados del punto anterior.
    - Calificar al Alumno (guardar registro en la tabla destinada a ello) con el puntaje obtenido.
    - Mostrar la salida al Alumno (eliminando de la misma lo que se haya podido agregar debido a la ejecución de los tests) junto a su código ingresado, e informar cuál fue la calificación que se le asignó.
- Si hay errores:
    - Calificar al Alumno (guardar registro en la tabla destinada a ello) con cero puntos en la tarea correspondiente.
    - Mostrar el error al Alumno (eliminando lo que se haya podido agregar debido a la ejecución de los tests) junto a su código ingresado, e informar que se ha asignado una calificación igual a cero.
- Eliminar el archivo que fue creado y ejecutado.

Descripción del código

A continuación se describen algunos segmentos de código de importancia para el funcionamiento básico de la aplicación.

Para almacenar la información de usuarios, los grupos Alumno y Profesor, y la pertenencia de los usuarios a esos grupos, se utilizaron las tablas definidas por defecto en Web2py ya mencionadas anteriormente. Para la creación de los grupos, se utilizó la interfaz administrativa de la herramienta Web2py (por más información ver el "Manual del Administrador" adjunto).

Para guardar la información de las calificaciones de los Alumnos y los horarios de las tareas, en el archivo db.py perteneciente al Modelo, se definieron las tablas db.Nota y db.Hora, respectivamente.

- db.Nota:

```
db.define_table('Nota',
    Field('username',unique=True),
    Field('tarea1','double'),
```

```
    Field('tarea2','double'),
    Field('tarea3','double'),
    Field('tarea4','double'),
    Field('tarea5','double'),
    Field('tarea6','double'),
    Field('tarea7','double'),
    Field('tarea8','double'),
    Field('tarea9','double'),
    Field('tarea10','double'),
    Field('promedio','double'))
```

Con el comando "db.define_table" se define una nueva tabla en la base de datos "db". El primer argumento es el nombre de la tabla, en este caso "Nota". Posteriormente se agregan todos los campos que se deseen. El primero agregado es "username" (nombre de usuario) y se establece que sea un valor único mediante "unique=True". Luego se agregan los campos "tareaX", que contendrán las calificaciones del Alumno para cada una de las tareas, y el campo "promedio", donde se guardará el promedio de dichas calificaciones, una vez que se calcule. En estos casos se establece que el tipo de dato almacenado será un número fraccionario ("double").

- db.Hora:

```
db.define_table('Hora',
    Field('Tarea',unique=True,default='Tarea x'),
    Field('inicio','datetime'),
    Field('final','datetime'),
    Field('solucion','datetime'))
```

En esta nueva tabla, "Hora", se asigna un valor por defecto para el campo "Tarea", igual a "Tarea x", de manera que el Profesor tenga una referencia del formato con el que debe ingresar el nombre de la tarea, por ejemplo "Tarea 3". En el resto de los campos se establece que el tipo de dato será una fecha ("datetime").

Se consideran a continuación algunas funciones que permiten el flujo de trabajo básico para un Profesor.

- Formulario para subir letras de tareas y sus tests (FIGURA VII-13).

```
@auth.requires_membership('profesor')
def subir_tarea():
    form = FORM((TEXTAREA(_style="width:99%; height:150px; ", _wrap="soft",
_name='letra', value=letra_leida)),
```

```
           DIV(BR()),

           TEXTAREA(_id="editable", _style="width:100%; height:375px",
_wrap="soft", _name='tests', value=test_leido),

           P('Si deseas eliminar esta tarea, deja ambos cuadros en blanco y
haz clic en "Enviar datos".', _style="background-color:#E7FBC0; color:#000;
text-align:center; font-size:13px"),

           INPUT(_type='submit', _style="width:20%; position:relative;
top:0px; color:ForestGreen;", _value="Enviar datos"))
```

Es un formulario html, pero está implementado en el Controlador, en el módulo default.py, mediante los ayudantes ("helpers") de Web2py, como TEXTAREA, DIV, P, etc., debido a que la información ingresada en él se procesará en la misma función una vez presionado el botón "Enviar datos". La sentencia inicial impide el acceso a la página a cualquier usuario que no pertenezca al grupo Profesor. Luego se define la función con "def subir():". En la variable "form" se almacena el formulario, que consta básicamente de dos áreas de texto, una para la letra y otra para los tests, y un botón de tipo "submit" para dar inicio al procesamiento de los datos. Como los tests deben ser escritos en lenguaje Python, la segunda área de texto cuenta con un "id" igual a "editable", lo que significa que es un área con reconocimiento de sintaxis Python en la que se puede escribir.

Para desplegar el formulario en el navegador web del usuario, en el archivo html de igual nombre a la función, subir_tarea.html, se escribe lo siguiente:

```
<p style="background-color:#E7FBC0; color:#000; font-size:13px; text-
align:center;"> En el siguiente cuadro debes escribir una descripción para
esta tarea; la letra del problema.</p>

{{=form}}
```

El traspaso de la variable "form" del Controlador a la Vista se hace a través de los símbolos especiales "{{" y "}}", que es la manera que tiene la herramienta Web2py para embeber código Python en archivos de tipo html. Para que esto funcione, la función "subir_tarea()" en el Controlador, devuelve en su última línea la variable "form" con la sentencia:

```
return dict(form=form)
```

- Creación de archivos conteniendo letra y tests de una tarea.

Una vez que el Profesor escribió en el formulario anterior la letra y los tests para una tarea, se crean un par de archivos conteniendo esa información y se guardan en el servidor, en el directorio /tareas dentro del directorio de la aplicación.

```
if form.accepts(request.vars, session):

    ruta = 'applications/'+request.application+'/tareas/'

    if form.vars.letra != '':

        l = open(ruta+ruta_letra_tas+'.py','w')

        l.write(form.vars.letra)
```

```
        l.close()

    elif os.path.exists(ruta+ruta_letra_tas+'.py'):
os.remove(ruta+ruta_letra_tas+'.py')

    if form.vars.tests != '':

        t = open(ruta+ruta_test_tas+'.py','w')

        t.write(form.vars.tests)

        t.close()

    elif os.path.exists(ruta+ruta_test_tas+'.py'):
os.remove(ruta+ruta_test_tas+'.py')

    if form.vars.letra == '' and form.vars.tests == '':

        session.flash = {'warn' : 'Tarea eliminada'}

    else:

        session.flash = {'warn' : 'Tarea subida correctamente'}

    redirect(URL('elegir_tarea_a_subir'))
```

Si el formulario se acepta, se crea una variable conteniendo la ruta en la que se quieren guardar los archivos (la variable “request.application” contiene el nombre con el que se cargó la aplicación en el servidor). Si el área para el ingreso de la letra no se dejó en blanco, se crea un archivo dentro de /tareas llamado “Letra x.py”, donde x es el número de tarea. En él se escribe y almacena la letra ingresada. Si se dejó en blanco y previamente existía un archivo de nombre “Letra x.py”, el mismo se elimina. De igual forma, si el área de ingreso de tests no se dejó en blanco, se crea un archivo dentro del mismo directorio, llamado “Tarea x.py”, donde se escriben y almacenan los tests para la tarea indicada, y se elimina si se dejó el área en blanco. Si las dos áreas se dejan en blanco, no se crea ningún archivo, se eliminan los archivos que correspondan, y se muestra un mensaje flash indicando que se ha eliminado la tarea. Si fue posible cargar la letra o los tests, según corresponda, se muestra un mensaje flash indicando el éxito. En cualquiera de los casos, finalmente se redirige a la página de elección de tarea a subir.

- Agregar o modificar horarios (FIGURA VII-18).

Web2py implementa la función “SQLFORM.grid”, que genera, dada una tabla en una base de datos, una interfaz de administración de los registros de dicha tabla, permitiendo consultarlos, modificarlos y agregar nuevos. En el Controlador, se utilizó esta función para el manejo de horarios de la siguiente manera (dentro de la función “adm_horarios()”):

```
grid =
SQLFORM.grid(db.Hora,exportclasses={},fields=[db.Hora.Tarea,db.Hora.inicio,
db.Hora.final,db.Hora.solucion],headers={'Hora.solucion':'Solución'},orderb
y=db.Hora.Tarea)
```

Como primer argumento se pasa la tabla a administrar, db.Hora en este caso. Con “exportclasses”, se indica a qué tipo de archivos puede exportarse la información. Al pasarle a la función un diccionario vacío ({}) en este argumento, será posible exportar a todos los

formatos que por defecto da la posibilidad Web2py (CSV, TSV, XML, HTML). En “fields” se pasan todos los campos de la tabla que se quieren mostrar, teniendo la posibilidad de ocultar algunos que se generan por defecto, como el campo “id”, el cual no se desea mostrar al usuario. Mediante el argumento “headers” se puede establecer, para cada campo, el encabezado que se desee que aparezca en la visualización de la tabla. En este caso solo se hace para “Hora.solucion”. Con “orderby=db.Hora.Tarea” se ordena la visualización según el número de tarea.

Mediante

```
form = FORM(INPUT(_type='submit', _style="width:120px; position:relative;
top:30px; color:dimgray;", _value="Borrar tabla", _class="button"))
```

se agrega el botón “Borrar tabla”, que redirige a la página de confirmación antes de borrar todos los horarios establecidos (que se lleva a cabo con “db.Hora.drop()”).

Las variables “grid” y “form” se pasan a la vista (adm_horarios.html) a través de la sentencia

```
return dict(grid=grid,form=form)
```

En la vista se escribe entonces:

```
<p>{{=grid}}

{{=form}}</p>
```

Las restantes páginas que presentan una cuadrícula para la administración de registros se implementaron de manera similar.

Se consideran ahora algunas funciones y segmentos de código que permiten el flujo de trabajo básico para un Alumno.

- Presentación de una tarea (FIGURA VII-5).

Ya se han descrito las funciones que presentan una tarea, con su letra y un área de texto para ingresar código Python, en la sección Funciones y tablas principales. Se ha visto también cómo se presenta un formulario en esta misma sección. A continuación se describe el procedimiento para visualizar la letra del problema, en la página de la tarea correspondiente, previamente cargada por un Profesor.

La función en el Controlador que se invoca al ingresar a la página que contiene una tarea (tareaX.html) es “tareaX()”. Antes de acceder a la tarea se verifica que el usuario sea un Alumno y que la fecha actual se encuentre entre la de inicio y final de la tarea, o que el usuario que intenta acceder sea un Profesor. Esto se logra escribiendo en el Controlador, en la línea anterior a la que define la función, el siguiente segmento de código:

```
@auth.requires((auth.has_membership('alumno') and
(fecha_actual_mayor_a(session.inicio) and
fecha_actual_menor_a(session.final))) or auth.has_membership('profesor'))
```

Las funciones “fecha_actual_mayor_a(fecha)” y “fecha_actual_menor_a(fecha)”, verifican, como sus nombres lo indican, que la fecha actual sea mayor (o menor) que la fecha que se les

pasa como argumento. Los valores pasados, en este caso “session.inicio” y “session.final”, se obtienen de la tabla db.Hora previamente, en la función “seleccionar()” que se invoca al ingresar a la página de elección de tarea.

Luego se define la función y se establece el valor de la variable “ruta_letra”, con la que se invoca a la función “verificar(ruta_letra)”.

```
def tareaX():
    ruta_letra = 'Letra '+session.nro_tarea
    letra = verificar(ruta_letra)
    letra = letra + '\n\n\n# Estás usando Python versión ' + verPyth
    tarea_nro = 'Tarea '+session.nro_tarea
    form = tarea(tarea_nro)
```

La variable session.nro_tarea se establece en la función “seleccionar()” al elegir la tarea en la página correspondiente. “verificar(ruta_letra)” verifica que la letra de la tarea exista, y en caso afirmativo la lee y la devuelve en una variable. Si la letra no ha sido cargada o el archivo donde debería guardarse está vacío, la variable devuelta contiene un mensaje apropiado.

```
def verificar(ruta_letra):
    ruta = 'applications/'+request.application+'/tareas/'
    ruta_letra = ruta+ruta_letra+'.py'
    if os.path.exists(ruta_letra):
        l = open(ruta_letra,'r')
        letra = l.read()
        l.close()
        if letra == '':
            letra = 'Aún no existe una descripción para esta tarea.'
    else:
        letra = 'Aún no existe una descripción para esta tarea.'
    return(letra)
```

Se agrega al finalizar el texto de la letra, un mensaje que muestra la versión de Python con la que se está trabajando (contenida en la variable verPyth), ya que pueden existir algunas diferencias a la hora de escribir código dependiendo de este hecho. Por ejemplo, en versiones superiores a 3, la función “print” debe escribirse necesariamente con paréntesis, es decir “print(‘Hola’)”.

En la variable “form” se almacena el área de texto que se desplegará para ingresar código Python, y se obtiene de la ejecución de la función “tarea(nro_tarea)”. Las variables “letra” y “form” se pasan luego a la Vista para presentarlas al usuario. En tarea1.html se escribe:

```
{{=FORM(TEXTAREA(_style="width:99%; height:150px; background-
color:#E7FBC0", _wrap="soft", _name='letra', _readonly='true',
value=letra))}}

{{=form}}
```

La letra se presenta en un área de texto de solo lectura (o no modificable).

- Testeo del código ingresado.

Una vez que se escribe código Python en el área de texto correspondiente en la página de una tarea y se da clic en el botón “Compilar”, se testea dicho código con los tests previamente cargados por el Profesor. Para ello se crea un archivo temporal que contiene el texto ingresado por el usuario, más los tests que se van a buscar al archivo en el que están almacenados, más un trozo de código fijo que permite la ejecución de los tests y la obtención de los resultados.

En primer lugar se captura el texto ingresado por el usuario en el formulario y se lo asigna a una variable “texto” (se hace en la función “tarea(nro_tarea)”):

```
FORM(TEXTAREA(_id="editable", _style="width:100%; height:405px",
_wrap="soft", _name='texto_a_ingresar', value=session.texto,
requires=IS_NOT_EMPTY("Ingresa código Python en WebIde2"))

if form.accepts(request.vars, session):

    texto = '# -*- coding: utf-8 -*-\n'+str(form.vars.texto_a_ingresar)
```

Como “texto” compone el principio del archivo, se le agrega antes el segmento de código que establece la codificación con la que se trabajará (UTF-8).

Luego, y dentro de la función “ejecutar(texto,tarea)”, se leen del archivo correspondiente los tests para la tarea actual.

```
t = 'applications/'+request.application+'/tareas/'+tarea+'.py'

if os.path.exists(t):

    a1 = open(t, "r")

    testeos = a1.read()

    testeos = 'import unittest\n'+testeos

    a1.close()

    haytest = True

else:

    testeos = 'import unittest\n'

    haytest = False
```

La variable “tarea” es una cadena de caracteres de la forma “Tarea x”, de manera que la variable “t” contiene la ruta al archivo Tarea X.py dentro del directorio /tareas. El texto leído se almacena en la variable “testeos”, agregándole al inicio la sentencia “import unittest\n”, de forma que el Profesor no necesita escribirla cada vez que vaya a cargar un test. Si no se

encuentra el archivo, se guarda solo esa sentencia en la variable “testeos” y se establece el valor de la variable “haytest” en Falso, para poder luego dar un mensaje de error apropiado.

Por último se almacena en la variable “final” el segmento de código fijo mencionado anteriormente. Esta variable depende de la versión detectada de Python del servidor, pero en rasgos generales es una cadena de caracteres que contiene el siguiente texto:

```
final = """if __name__ == '__main__':
    u = unittest.main(exit=False)
    totalmal=[]
    errores=[]
    fallas=[]
    if u.result.errors:
        for indice in range(len(u.result.errors)):
            e = str(u.result.errors[indice][0])
            es = e.split()
            if es[0] not in errores:
                errores.append(es[0])
                totalmal.append(es[0])
    if u.result.failures:
        urf = u.result.failures     # lista de 2-tuplas(algo con el nombre,
mensaje)
        for indice in range(len(urf)):
            f = str(urf[indice][0]) # string de "algo con el nombre"
(nombre en 1er lugar)
            fs = f.split()
            if fs[0] not in fallas:
                fallas.append(fs[0])
                if fs[0] not in errores: totalmal.append(fs[0])
    print
('#!#corridos#=#'+str(u.result.testsRun)+'#=#','#!#totalmal#=#'+str(len(tot
almal))+'#=#','#!#errores#=#'+str(len(errores))+'#=#','#!#fallados#=#'+str(
len(fallas))+'#=#')
    """
```

Se utiliza el módulo “unittest” de Python para la ejecución de los tests. El argumento “exit=False” permite que se ejecute código luego de haberse corrido los tests, es decir todo el código que viene después de “u = unittest.main(exit=False)”. En versiones de Python menores a la 2.7, este argumento no es reconocido, por lo que la variable “final” no se forma de igual

manera. En “u.result.errors” y “u.result.failures” se almacenan tuplas que contienen los nombres de los tests que no pasó el código testeado. Se consultan estos nombres y se van agregando a una lista “errores” si el nombre proviene de “u.result.errors” o “fallas” si proviene de “u.result.failures”, y en cualquiera de los dos casos a otra lista, “totalmal”, si no se encuentran ya en ella, de manera de no repetir nombres. El largo de esta lista será igual a la cantidad de tests no aprobados. Estos resultados se imprimen en la salida junto a palabras clave apropiadas, por ejemplo “#!#totalmal#=#”, para ubicarlos luego fácilmente.

Con “texto”, “testeos” y “final” establecidas, se actualiza la variable “texto” y con ella se escribe el archivo temporal mencionado inicialmente, cuyo nombre se almacena en la variable “arch_Pyth” y se forma con la palabra “archivo”, más el nombre de usuario (cédula de identidad) del usuario que está trabajando, más un número aleatorio de entre tres y seis cifras, más la extensión “.py”.

```
texto = texto+'\n'+testeos+'\n'+final

a2 = open(arch_Pyth ,"w")

a2.write(texto)

a2.close()
```

Habiéndose creado y escrito el archivo, se lo ejecuta para que se corran los tests y se obtengan los resultados según lo descrito anteriormente, utilizando el intérprete Python del servidor en el que se encuentra la aplicación. Para ello se escribe, aún dentro de la función “ejecutar(texto,tarea)”:

```
if so == 'nt':

    proc = subprocess.Popen([arch_Pyth], stdout=PIPE, stderr=PIPE,
shell=True)

else:

    proc = subprocess.Popen('python '+ arch_Pyth, stdout=PIPE, stderr=PIPE,
shell=True)
```

“so” contiene la información del sistema operativo del servidor, por lo que la ejecución se hace dependiendo de ese valor. En cualquiera de los casos y si la ejecución se completa, la salida estándar y la de errores se capturan en un par de variables de la siguiente manera:

```
pstdout, pstderr = proc.communicate() # communicate() devuelve una tupla
con la salida estándar y errores
```

Como se ha mencionado en secciones anteriores, la ejecución podría no completarse si llegara a ser demasiado extensa, por ejemplo si se presentara en el código ingresado un bucle infinito. En ese caso, después de un tiempo determinado, se corta la ejecución y se redirige a una página donde se informa sobre el error de tiempo de espera agotado. El código que permite este comportamiento se explica en la sección Resolución de algunos problemas.

- Obtención de los resultados del testeo.

La función “ejecutar(texto,tarea)” invoca a “corregir(resultado)” pasándole como argumento la variable “pstdout”, o “pstderr” si la versión del intérprete Python del servidor es menor a 2.7. Esta función se encarga de buscar las palabras clave mencionadas antes (o las que correspondan en versiones de Python menores a 2.7), hacer el cálculo del puntaje obtenido y devolverlo junto a la cantidad de tests corridos y fallados.

```
def corregir(resultado):

    astring = str(resultado)

    if '#!#corridos#=#' in astring:

        division = astring.split('#=#')  # se divide la salida según el
símbolo especial #=#

        pruebasFloat = float(division[1])  # la cantidad de tests corridos
se encuentra en la posición 1, según la división anterior

        mal = float(division[3])  # la suma de tests con fallas y/o errores
(sin repetición) está en la posición 3
```

La función busca la palabra clave “#!#corridos#=#” en el texto que se le haya pasado como argumento, y si la encuentra, divide dicho texto según los caracteres especiales “#=#”. De esta manera se sabe en qué posiciones se encuentran las cantidades de tests corridos y fallados. Luego se realiza el cálculo del puntaje en términos de porcentaje:

```
puntaje = 100*(pruebasFloat-mal)/pruebasFloat
```

Si no llegaran a encontrarse las palabras clave por algún motivo, se desplegará un mensaje apropiado.

En versiones de Python menores a 2.7 se buscan otras palabras clave, pero el procedimiento es similar.

- Calificación del Alumno.

Una vez obtenido el puntaje a asignar, se guarda el mismo en la tabla de calificaciones correspondiente. Para ello, la función “ejecutar(texto,tarea)” invoca a “calificar(tarea,nota)”. Esta función verifica que el usuario pertenezca al grupo Alumno, en cuyo caso actualiza o inserta un nuevo registro con el nombre de usuario del Alumno y posteriormente actualiza la calificación con el valor de la variable “nota”, en la tarea que corresponda, según el número que se le haya pasado como argumento en la variable “tarea”.

```
def calificar(tarea, nota):

    if auth.has_membership('alumno'):

        ci = auth.user.username

        db.Nota.update_or_insert(username=ci)

        if tarea == 'Tarea 1':

            db(db.Nota.username==ci).update(tarea1=nota)

            return True
```

```
        elif tarea == 'Tarea 2':

            db(db.Nota.username==ci).update(tarea2=nota)

            return True

        elif tarea == 'Tarea 3':

            db(db.Nota.username==ci).update(tarea3=nota)

            return True
```

El código continúa y se repite para las diez tareas que trae la aplicación. Luego de actualizar la calificación, la función devuelve un valor Verdadero.

- Muestra de resultados (FIGURA VII-6 y FIGURA VII-7).

En "ejecutar(texto,tarea)", si no hubieron errores de compilación, se pregunta si la función "calificar(tarea,nota)" devolvió el valor Verdadero y, según sea la respuesta afirmativa o no, se establece el valor de una variable de sesión, "session.mensaje", que se mostrará al usuario en una nueva página:

```
calificado = calificar(tarea, puntaje)

if calificado:

    session.mensaje = 'TAREA: '+tarea+'\n\nTuviste '+str(int(tests-
fallos))+' de '+str(int(tests))+' tests correctos. \n\nObtuviste el
'+str(puntaje)+' % de los puntos.'

else: session.mensaje = 'No se pudo calificar tu tarea. Por favor ponte en
contacto con tu profesor.'
```

De similar forma, si existieron errores de compilación:

```
calificado = calificar(tarea, 0.0)

if calificado:

    session.mensaje = 'TAREA: '+tarea+'\n\nTuviste errores de compilación.
El puntaje asignado a esta tarea es 0.0, pero ánimo!, puedes intentarlo
nuevamente.'

    else: session.mensaje = 'No se pudo calificar tu tarea. Por favor ponte
en contacto con tu profesor.'
```

En las variables "session.salida" y "session.error" se guardan la salida debida a la ejecución del código ingresado (si se imprimió algo en pantalla) y los eventuales errores de compilación que hayan existido, respectivamente.

El contenido de las tres variables de sesión mencionadas, se mostrará, según corresponda, al usuario en la página de resultados (second.html). Al ser variables de sesión no es necesario devolverlas en un "return", la Vista igualmente puede acceder a ellas.

En second.html se muestra "session.error" o "session.salida" dependiendo de si hubieron o no errores de compilación:

```
{{#If para distinguir la salida y desplegarla según haya o no errores}}

{{if (session.error != ""):}}

      <h5> <img src="{{=URL('static','images/error.png')}}"/> <FONT
COLOR="#DD0C0C"> Error: </FONT> </h5></br>

      <textarea style="border-color:#; resize:none; font-size:12pt;
color:#000; width:100%; height:181px"; onfocus=; readonly="true";
wrap="soft">{{=session.error}}</textarea>

      <textarea style="border-color:#; resize:none; font-size:12pt;
color:#000; width:100%; height:190px"; onfocus="this.blur()";
readonly="true"; wrap="soft">{{=session.mensaje}} </textarea>

{{else:}}

      <h5> <img src="{{=URL('static','images/ok.png')}}"/> <FONT
COLOR="#5BF18B"> Salida: </FONT> </h5></br>

      <textarea style="border-color:#; resize:none; font-size:12pt;
color:#000; width:100%; height:181px"; onfocus=; readonly="true";
wrap="soft">{{=session.salida}}</textarea>

      <textarea style="border-color:#; resize:none; font-size:12pt;
color:#000; width:100%; height:190px"; onfocus="this.blur()";
readonly="true"; wrap="soft">{{=session.mensaje}} </textarea>

{{pass}}
```

Los corchetes denotan la presencia de código Python embebido en Html. Los mensajes, salidas o errores se muestran todos en áreas de texto no modificables.

En el lado derecho de la pantalla se muestra, en cualquier caso, el código ingresado por el usuario, que quedó almacenado en la variable “session.texto”.

```
{{block right_sidebar}}

<h5> <FONT COLOR="#424242"> Este es el código Python que ingresaste :
</FONT> </h5></br>

<Textarea id="no_editable", style="width:100%;
height:390px">{{=session.texto}} </Textarea>

{{end}}
```

El “id” igual a “no_editable”, indica que el área de texto tiene reconocimiento de sintaxis Python, pero es de solo lectura.

Resolución de algunos problemas

- Reconocimiento de sintaxis Python.

Esta funcionalidad se logró mediante un editor de texto con reconocimiento y resaltado de sintaxis Python. El editor utilizado fue EditArea, basado en JavaScript, de libre distribución bajo licencia GNU Lesser General Public License (Licencia Pública General Reducida de GNU). EditArea reconoce la sintaxis de varios lenguajes de programación, entre ellos Python, y presta

además funcionalidades como indicar los números de línea, poder usar la tecla "tab" para indentar, buscar palabras en el texto, entre otras.

Para llevar a cabo el funcionamiento de este editor, se cargaron los archivos que lo componen en un directorio dentro del de la aplicación (/static/edit_area), y se agregó el siguiente código en la Vista:

```
<script language="javascript" type="text/javascript"
src="{{=URL('static','edit_area/edit_area_full.js')}}"></script>

<script language="javascript" type="text/javascript">

editAreaLoader.init({

    id : "editable"

    ,syntax: "python"

    ,start_highlight: true

    ,font_size: 10

    ,replace_tab_by_spaces: 4

    ,show_line_colors: true

    ,word_wrap: true

    ,language: "es"

    ,allow_toggle: false

 });</script>
```

Se establecen aquí ciertos parámetros como el tipo de sintaxis a reconocer, el tamaño de la letra a utilizar, a cuántos espacios equivale la tabulación, etc., y quedan guardados bajo el identificador "editable". Un área de texto con identificador ("id") igual a "editable" tendrá las características establecidas anteriormente. El nombre de "editable" proviene del hecho que por defecto el editor permite escribir en él. Si se desea presentar cierto código en un área de texto con reconocimiento de sintaxis, pero sin que se pueda modificar el mismo, se debe agregar en el "script" descrito antes la línea:

```
is_editable: false
```

Como la presente aplicación muestra código no editable, en la presentación de resultados (FIGURA VII-6) o en la presentación de una solución (FIGURA VII-9) por ejemplo, se creó un nuevo tipo de área de texto bajo el identificador "no_editable" de igual forma a la anterior, pero con la diferencia de haberse agregado la línea mencionada "is_editable: false".

Al ser varias las páginas que utilizan estas áreas de texto, el "script" fue colocado en los archivos layout2.html y layout3.html de la Vista, de donde heredan su formato principal la mayoría de las páginas de la aplicación.

- Testeo simultáneo de código.

Ya se ha descrito la manera de realizar los testeos del código ingresado por un usuario (se forma un archivo temporal con extensión .py que contiene el código del usuario, más los tests, más una porción de código fijo). Ante la eventualidad que se quiera crear este archivo en el momento que ha sido creado para la corrección de código a otro usuario y no se ha eliminado aún, la aplicación nombra el archivo de distinta manera para cada usuario, evitando cualquier tipo de conflicto que pudiera ser causado por esta situación. Por lo tanto el nombre del archivo se forma comenzando con la palabra "archivo", siguiendo con el nombre de usuario actual y finalizando con un número aleatorio de entre tres y seis cifras, y se almacena en la variable "arch_Pyth" para su posterior uso.

```
ran = random.randint(100,999999)

arch_Pyth = "archivo"+str(auth.user.username)+str(ran)+".py"
```

- Ejecución con el intérprete Python del servidor.

La corrección de una tarea se efectúa al ejecutar el archivo mencionado en el punto anterior, con el intérprete Python del servidor donde se encuentra la aplicación. Para ello se eligió el módulo "subprocess" de Python, en particular la clase "subprocess.Popen" que permite ejecutar un subprograma en un nuevo proceso, dando la posibilidad de capturar sus salidas estándar y de errores. Además, al crearse un proceso nuevo, da la posibilidad, en caso de necesitarse, de forzar dicho proceso a finalizar, sin afectar la ejecución de cualquier otro proceso simultáneo.

El uso de esta herramienta debe llevarse a cabo de manera distinta en función del sistema operativo utilizado. Por este motivo la aplicación consulta el nombre del sistema operativo actual y, dependiendo de la respuesta, ejecuta el "subprocess" como corresponda.

```
so = os.name

if so == 'nt':

    proc = subprocess.Popen([arch_Pyth], stdout=PIPE, stderr=PIPE,
shell=True)

else:

    proc = subprocess.Popen('python '+ arch_Pyth, stdout=PIPE, stderr=PIPE,
shell=True)
```

La variable "so" contiene el nombre del sistema operativo. La distinción se hace entre Windows ("nt") o cualquier sistema basado en Unix (no se ha testeado el funcionamiento en sistemas Mac).

Se crea el nuevo proceso ("proc") y se abren "tuberías" ("PIPE") para ir almacenando la salida estándar y la de errores, que luego se capturan en un par de variables de la siguiente manera:

```
pstdout, pstderr = proc.communicate()
```

La función "communicate()" devuelve una tupla con la salida estándar y la de errores, las que se pueden utilizar ahora estando guardadas en "pstdout" y "pstderr" respectivamente.

- Bucles infinitos.

Se comprobó que si en el código escrito en el archivo con extensión .py que se ejecuta (mencionado antes), existía un bucle infinito o con muchas iteraciones (en el entorno de las 600), de manera que la ejecución tomara demasiado tiempo, la aplicación dejaba de responder, impidiendo que se pudiera seguir trabajando en ella. Para solucionar ese comportamiento, se utilizaron las características de la clase “subprocess.Popen” (este fue uno de los motivos de su elección), ya que un proceso que pertenece a la misma, permite ser terminado cuando se desee.

La implementación consiste en, luego de creado el proceso “proc” de la forma que se vio en el punto anterior, activar un temporizador (o “timer”), que pasado determinado tiempo (por defecto será de 9 segundos) invoque a la función “matar()”, que es la encargada de finalizar el proceso “proc”. Si este proceso finaliza normalmente antes de que la cuenta del temporizador llegue a su fin, la misma se cancela y no se invoca a “matar()”.

```
# Timer
TIEMPO_MAX=9.0
t=threading.Timer(TIEMPO_MAX,matar)
t.start()
# Se pregunta por el estado del proceso
esta_vivo=True
while esta_vivo:
    if proc.poll() != None:
        esta_vivo=False
        t.cancel()
```

Con “proc.poll()” se consulta si el proceso “proc” ya ha finalizado; si la respuesta es “None”, significa que aún no lo ha hecho. Por lo tanto una respuesta diferente a “None” indicará que el proceso finalizó y se podrá salir del bucle “while” estableciendo “esta_vivo” en Falso. En este caso se cancela el temporizador con “t.cancel()”. Si la ejecución del bucle “while” continúa y el temporizador terminó su cuenta, se invoca a “matar()”, que terminará el proceso “proc”, de manera de poder salir del bucle “while”.

```
def matar():
    proc.stdout.close()
    proc.stderr.close()
    proc.kill()
    return
```

En esta función se cierran las salidas estándar y de errores del proceso “proc” y posteriormente se finaliza el proceso con “proc.kill()”.

A continuación se consulta por el estado de la salida estándar “stdout” del proceso; si está cerrada es porque se invocó a la función “matar()”. En este caso se redirige al usuario a una página donde se informa sobre el error de tiempo de espera agotado con un mensaje apropiado, se califica al Alumno con cero puntos y se elimina el archivo que se ejecutó.

```
# Si stdout está cerrada en este punto es porque se llamó a la función
matar()

if proc.stdout.closed:

    pstderr="error de timeout"

    pstdout=""

    calificado = calificar(tarea, 0.0)

    session.flash = {'adv' : 'Tiempo de espera agotado'}

    # Se elimina el archivo creado

    os.remove(arch_Pyth)

    redirect(URL('timeout'))
```

En caso de no estar cerrada la salida aún, se continúa con el flujo normal de funcionamiento, que será capturar en un par de variables las salidas estándar y de errores del proceso “proc”.

- Ingreso a la aplicación con nombre de usuario.

Por defecto, Web2py pide como forma de acceso a una aplicación, el correo electrónico del usuario. En el caso de Webide2, se consideró más apropiado que el ingreso se realizara con el nombre de usuario. Para ello se agregó en el archivo db.py del Modelo la siguiente línea:

```
auth.define_tables(username=True)
```

- Tiempo de formularios.

La aplicación basa gran parte de su funcionamiento en el ingreso de datos mediante formularios web. Por defecto, las nuevas versiones de Web2py permiten tener abierto un formulario con la posibilidad de ingresar datos y que los mismos sean procesados, por un lapso máximo de cinco minutos. Pasado este tiempo, se puede continuar escribiendo en el formulario, pero al intentar enviar los datos para su procesamiento (con algún botón de tipo “submit”), se redirige al usuario a la misma página sin procesarlos y sin avisar de ninguna manera que no se ha hecho; lo que equivaldría a solamente refrescar la página. Como se considera que se tendrá que ingresar código de programación en la mayoría de los formularios de la aplicación, se extendió el “tiempo de vida” de los mismos, estableciéndolo en una hora. Esto se logró fijando en 60 el nuevo valor de la constante “SLEEP_MINUTES” en el archivo /web2py/scripts/session2trash.py:

```
SLEEP_MINUTES = 60
```

- Corrección automática.

Para llevar a cabo la corrección automática de las tareas, se analizaron y probaron algunas herramientas de testeo en Python. La que mejor se adecuó a las necesidades de la aplicación fue el módulo "unittest" y en particular su clase "TestCase", por simplicidad de uso para el docente y correcto funcionamiento para los fines buscados.

Se pretende comparar las salidas que tiene una función implementada por un estudiante, dadas determinadas entradas, con salidas preestablecidas por el Profesor. La clase "TestCase" posee cierta cantidad de métodos con los que el docente puede contar para chequear las salidas de las funciones que desee corregir. Los mismos se enumeran en la siguiente tabla:

Método	Verifica
assertEqual(a, b)	a == b
assertNotEqual(a, b)	a != b
assertTrue(x)	bool(x) es True
assertFalse(x)	bool(x) es False
assertIs(a, b)	a is b
assertIsNot(a, b)	a es not b
assertIsNone(x)	x es None
assertIsNotNone(x)	x es not None
assertIn(a, b)	a está en b
assertNotIn(a, b)	a no está en b
assertIsInstance(a, b)	isinstance(a, b)
assertNotIsInstance(a, b)	not isinstance(a, b)
assertRaises(exc, fun, *args, **kwds)	fun(*args, **kwds) arroja exc
assertRaisesRegexp(exc, re, fun, *args, **kwds)	fun(*args, **kwds) arroja exc y el mensaje concuerda con re
assertAlmostEqual(a, b)	round(a-b, 7) == 0
assertNotAlmostEqual(a, b)	round(a-b, 7) != 0
assertGreater(a, b)	a > b
assertGreaterEqual(a, b)	a >= b
assertLess(a, b)	a < b
assertLessEqual(a, b)	a <= b
assertRegexpMatches(s, re)	regex.search(s)
assertNotRegexpMatches(s, re)	not regex.search(s)
assertItemsEqual(a, b)	sorted(a) == sorted(b)
assertDictContainsSubset(a, b)	Todos los pares clave/valor en a existen en b

Tabla VII-2 – Métodos de la clase TestCase (Python Software Foundation, 2010).

Para llevar a cabo el testeo, el Profesor deberá crear una clase que herede de "unittest.TestCase" y contenga los tests necesarios para comparar una determinada cantidad de salidas. A continuación se muestra un ejemplo sencillo.

Se supone que se desea testear una función que debe hallar las raíces de un polinomio de segundo grado, dados sus coeficientes. Se define para ello una clase "TestPolinomio" que hereda de "unittest.TestCase", y dentro de ella un test llamado "test_buscar_raices" que evaluará el desempeño de la función llamada "buscar_raices". En "COEFICIENTES_RAICES" se

definen las entradas para las que se va a realizar el testeo y las salidas que las mismas deben producir en la función testeada. Luego se compara mediante "self.assertEquals(raices, raices_esperadas)" las salidas dadas por la función "buscar_raices", con las salidas preestablecidas (en este caso (0,0), (-1,2) y (-2,2)).

```
class TestPolinomio(unittest.TestCase):

    def test_buscar_raices(self):

        COEFICIENTES_RAICES = [

            ((1,0,0), (0, 0)),

            ((-1,1,2), (-1, 2)),

            ((-1,0,4), (-2, 2))]

        for coef, raices_esperadas in COEFICIENTES_RAICES:

            raices = buscar_raices(coef[0], coef[1], coef[2])

            self.assertEquals(raices, raices_esperadas)
```

Al escribir esta clase no es necesario importar el módulo "unittest", ya que la aplicación lo hace automáticamente al formar el archivo .py de ejecución (descrito en puntos anteriores).

Los nombres de los tests deben comenzar con la palabra "test" para que sean corridos.

La forma de obtener y procesar los resultados de los testeos fue tratada en la sección Descripción del código.

VII.2. Webide 3

VII.2.1. Descripción

Es una aplicación web que provee facilidades para el dictado de un curso designando profesores y habilitando alumnos a registrarse en él, a partir de un rol "administrador". Los profesores pueden cargar tareas (letra, tests, solución, fecha de inicio y finalización, etc.) subiendo archivos, elegir entre dos escalas de notas, fijar una nota mínima de aprobación, exportar notas a varios formatos de planilla. Los estudiantes pueden bajar la letra de la tarea, subir un archivo con su solución, ver la nota obtenida. El administrador puede borrar todos los registros y reinicializar el curso.

Esta aplicación no califica como un IDE (Integrated Development Environment) sino más bien como un sistema de "upload": los estudiantes no pueden ver los resultados del compilador ni tienen forma de corregir sobre la aplicación. No obstante es una aplicación que permite la corrección de tareas en forma automática registrando una calificación en el historial del estudiante.

VII.2.2. Funcionamiento

Se describirá el sistema de autenticación y luego el funcionamiento normal del programa según el usuario sea profesor o alumno.

Sistema de autenticación

Al ingresar el usuario verá la pantalla que se muestra en la figura siguiente donde tendrá la opción de ingresar a la aplicación autenticándose. De no haberse registrado tiene la opción de hacerlo.

FIGURA VII-29 – Página de inicio de Webide3.

Para registrarse, el administrador selecciona su propia contraseña cuando ingresa por primera vez a la aplicación, mientras que profesores y alumnos dispondrán de una contraseña cada uno; a los profesores les será dada por el administrador y a los alumnos por los profesores. Al ingresar en Registrarse se verá el siguiente formulario:

FIGURA VII-30 – Página de registro.

Una vez completados los datos, se envía el formulario y se inserta un nuevo registro en la base de datos, en la tabla Datos_Profesores o Datos_Alumnos según corresponda.

Para que los datos sean procesados deben pasarse las reglas de validación, de lo contrario aparecerá un mensaje de error. Por ejemplo, se pide que se ingrese un nombre, de lo contrario se le indicará que lo haga, tal situación se muestra en la figura siguiente:

FIGURA VII-31 – Error de validación de formulario.

Una vez registrado el usuario será redirigido a la interfaz correspondiente a su tipo.

Para identificarse los usuarios deben ingresar cédula y contraseña. Pasada la validación se los redireccionará a la interfaz apropiada.

Profesores

Ya registrados y una vez hayan ingresado, los profesores verán la siguiente página:

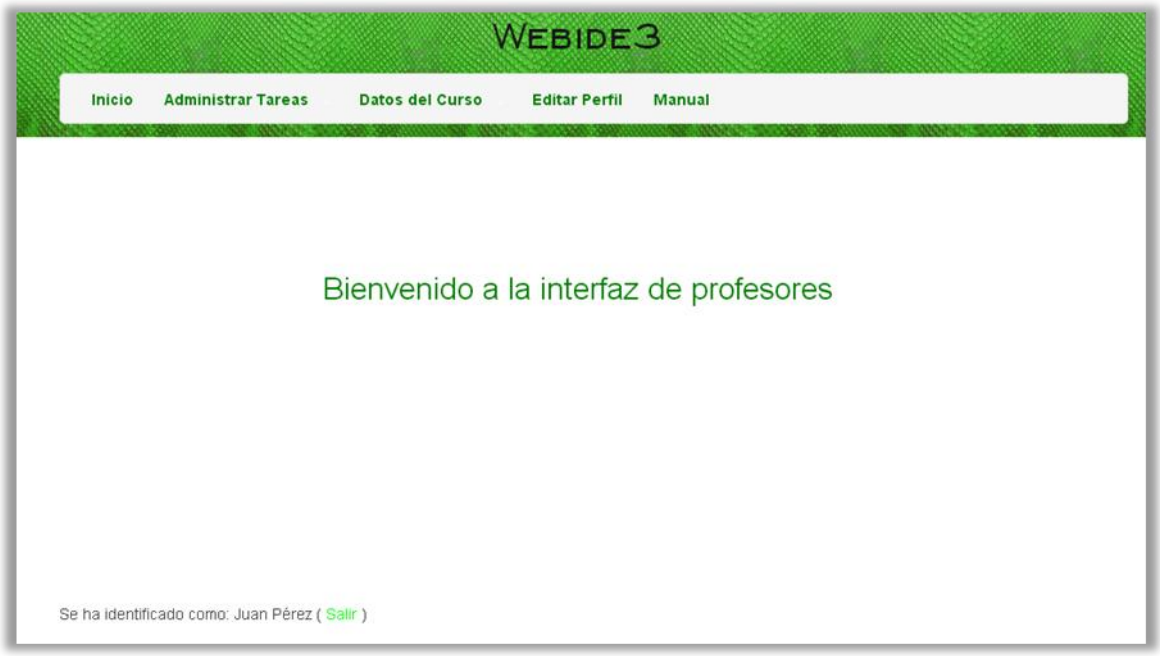

FIGURA VII-32 – Interfaz de profesores.

Lo primero que deben hacer es ingresar en "Datos del Curso" y luego en "Ingresar Contraseña del Curso", opción que les permitirá elegir la contraseña que le darán a los estudiantes para que éstos últimos puedan registrarse.

Para todo lo relacionado a las tareas deben entrar en "Administrar Tareas". Lo hacen a través del menú que se encuentra en la parte superior de la pantalla. Allí verán las opciones que se muestran en la figura siguiente:

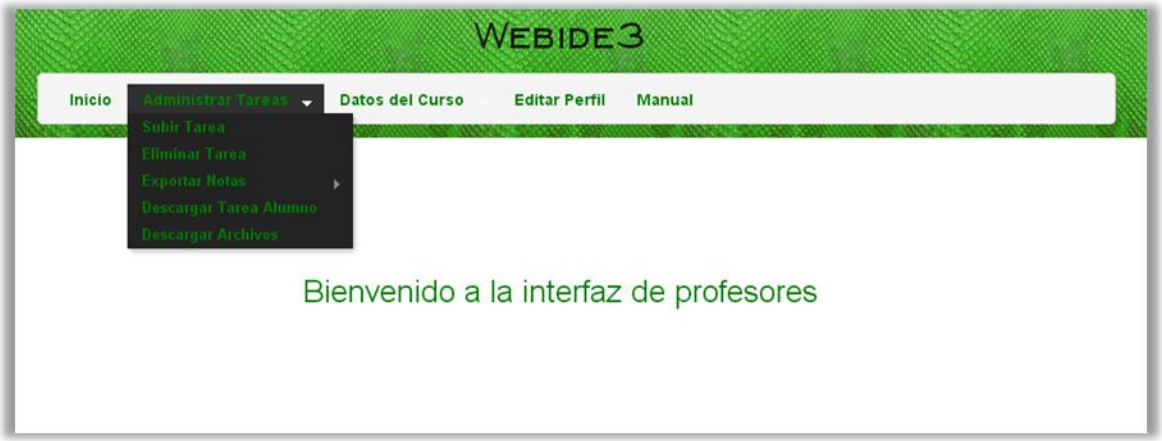

FIGURA VII-33 – Administrar tareas.

Para subir una tarea, entran a "Subir Tarea" y se desplegará el siguiente formulario:

FIGURA VII-34 – Subir tarea.

Ingresada la tarea los docentes han de esperar que los alumnos suban sus archivos. Pasado el plazo de recepción, ingresan a "Exportar Notas" y se desplegará un submenú donde han de elegir cuál es la tarea cuyas notas desean descargar:

FIGURA VII-35 – Exportar notas.

El menú es dinámico, sólo aparecen como disponibles para exportar las notas de aquellas tareas que se hayan ingresado hasta el momento (que en la figura son dos) y las notas finales calculadas como el promedio de las notas de todas las tareas.

Si por ejemplo se seleccionó la tarea 1, se verán las notas de la tarea 1 de aquellos alumnos que estén registrados, tal cual se muestra en la figura a continuación:

FIGURA VII-36 – Notas de una tarea.

Debajo de la tabla tienen la opción de descargar la misma en diferentes formatos, si por ejemplo eligen CSV la tabla será descargada en formato CSV como se muestra en la figura siguiente (se descarga en este caso con el nombre rows (6) pero el profesor le puede poner al archivo el nombre que desee):

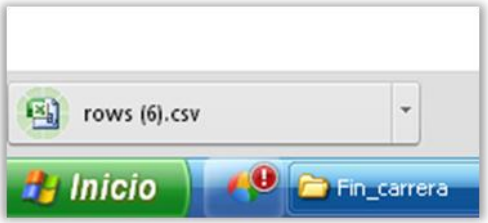

FIGURA VII-37 – Descarga de archivo con calificaciones.

Alumnos

Habiendo ya ingresado, los estudiantes verán la siguiente pantalla:

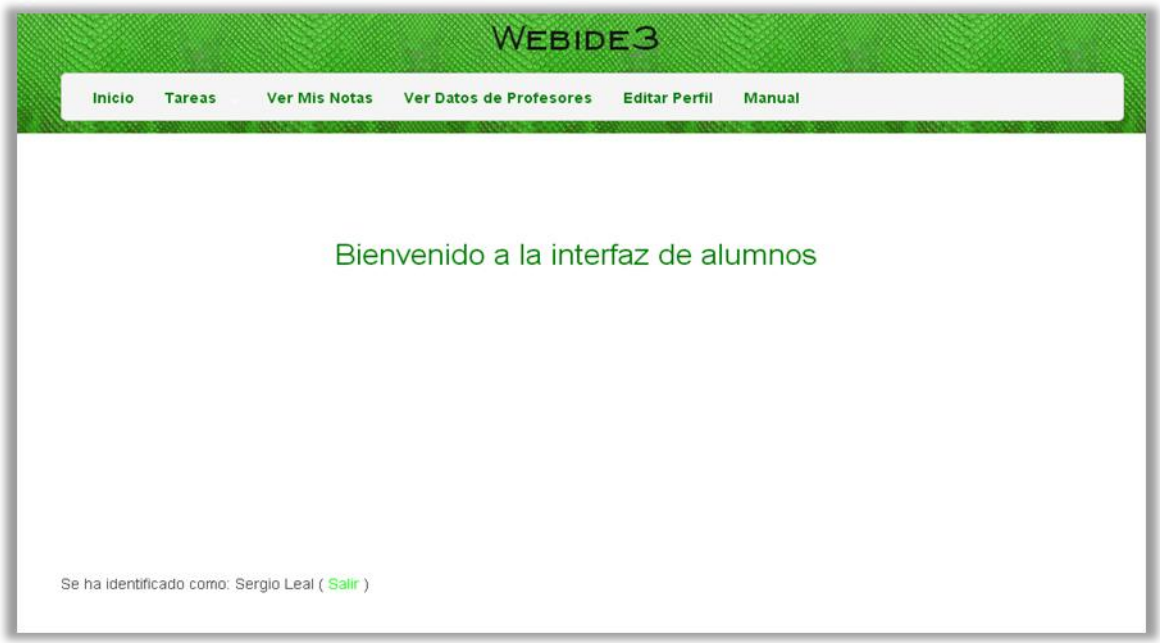

FIGURA VII-38 – Interfaz de alumnos.

Para subir una tarea deben ingresar a "Tareas" a través del menú principal. Se desplegará entonces un submenú conteniendo todas las tareas que los profesores han ingresado hasta el momento. Seleccionan cuál es la que desean subir y verán un nuevo submenú que les permite elegir entre descargar la letra y subirla:

FIGURA VII-39 – Menú tareas.

Si ingresan a "Subir" verán el siguiente formulario:

FIGURA VII-40 – Subir resolución a una tarea.

Enviado el formulario (si pasa las validaciones) la tarea es corregida y se obtiene una nota que se le muestra al estudiante; si, para fijar ideas suponemos obtuvo una calificación de 5 verá entonces:

FIGURA VII-41 – Calificación obtenida.

¿Qué sucede internamente?

Se actualiza la tabla Notas de la base de datos. Si por ejemplo el estudiante que hizo uso de la aplicación tiene cédula de identidad igual a 10 y la tarea que subió fue la 1 se tendrá lo siguiente:

FIGURA VII-42 – Asignación de la calificación para una cédula.

En este punto, por ejemplo el estudiante de cédula de identidad 3 no ha subido ninguna tarea y por tanto su nota es 0. Veamos qué sucede cuando sube su archivo, si por ejemplo se saca 10 se tendrá:

FIGURA VII-43 – Asignación de la calificación para una nueva cédula.

Si el estudiante de cédula 10 no está satisfecho con su nota y desea obtener una mejor calificación lo que hace es subir otra vez la tarea y se actualizará la tabla. Puede hacerlo mientras esté habilitada la tarea. Si por ejemplo se sacó 10 esta vez, se verá lo siguiente:

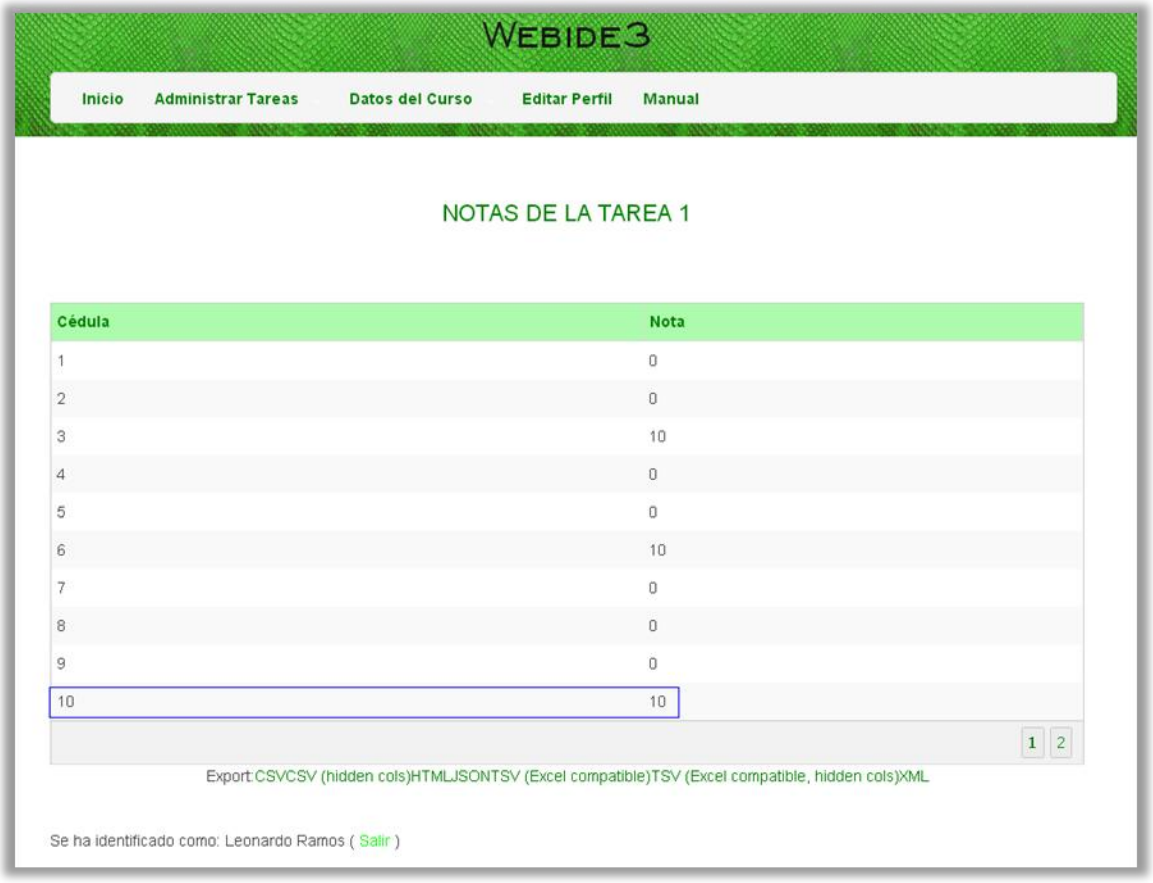

FIGURA VII-44 – Actualización de la calificación para una cédula.

La situación es completamente análoga para el resto de las tareas.

VII.2.3. Características generales y funcionales

Las ventajas y limitaciones son muy similares a las de WebIDE 2: es una aplicación libre, autocontenida, independiente del sistema operativo, su diseño no es óptimo etc. No obstante se destacarán algunas características particulares.

Ventajas

- El número de tareas no es fijo, el docente puede ingresar la cantidad de tareas que desee e incluso eliminar las mismas de acuerdo a lo que crea conveniente.
- Si bien el diseño no es óptimo, sí permite una fácil adaptación de la aplicación para otros usos. Si se quisiera por ejemplo implementar un corrector ortográfico en lugar de la corrección de archivos en Python, alcanza con cambiar el controlador “corrector” adecuándolo a las nuevas necesidades; el sistema de autenticación, la asignación de notas y las funcionalidades que corresponden a los usuarios permanecen incambiadas.

Limitaciones

- Las correcciones que los docentes y alumnos precisen realizar de sus archivos deben realizarlas fuera de la aplicación, esto es una clara consecuencia de que el sistema no es un IDE.
- Las escalas de notas vienen por defecto con la aplicación.

VII.2.4. Diseño

Primero se verá el flujo de trabajo, luego los componentes básicos necesarios (entendiendo por tales tablas en la base de datos, funciones en los controladores y vistas) y finalmente el proceso básico representativo de la parte más importante de la aplicación.

Flujo de trabajo

Tomando como punto de partida que los usuarios ya están registrados y han ingresado, el flujo básico de trabajo es el siguiente:

- Profesor:
  - Carga en la aplicación los datos y archivos necesarios para la realización de tareas.
  - Finalizado el tiempo estipulado por él mismo exporta la tabla de calificaciones.
- Alumno:
  - Selecciona la tarea que le corresponde realizar.
  - En la página de la tarea seleccionada, puede descargar la letra del problema y entregarla.
  - Enviada la tarea observa los resultados obtenidos.
  - Si quiere subir nuevamente la tarea repite estos mismos pasos.

Componentes principales

Siguiendo Web2py el paradigma MVC dividiremos el análisis en modelo, vista y controlador.

Modelo

Se debe tener una base de datos para guardar información relevante, le llamaremos “db”.

Se listan a continuación las tablas que serán necesarias:

1) Una llamada “Claves_Globales” para almacenar las contraseñas generales del curso; las que le serán dadas al administrador, profesores y alumnos para registrarse. Se supondrá un solo administrador, por tanto la clave primaria será la contraseña del mismo.
2) Una llamada “Datos_Admin” que almacene los datos del mismo: Nombre, Apellido, CI, Mail y Contraseña. La clave primaria será la cédula de identidad.
3) Una llamada “Datos_Profesores” análoga a “Datos_Admin” pero que almacene datos de los profesores.
4) Una llamada “Datos_Alumnos” análoga a “Datos_Admin” pero que almacene datos de los alumnos.
5) Una llamada “Notas” para almacenar las calificaciones obtenidas por los estudiantes. Tendrá como campos la CI del estudiante para identificar al mismo, un identificador de tarea para identificar la tarea y la Nota que pertenece a la tarea. Además se agregará un campo conteniendo el código que ingresó el alumno. Se hace notar que la CI por sí sola no es clave primaria, ya que tendremos varios registros con la misma cédula, debido a que cada alumno sube un archivo para cada una de las tareas. La clave primaria está compuesta por los campos CI e identificador de tarea.

6) Una llamada “Resultados_Finales” con los campos CI, la nota final calculada de acuerdo a algún criterio (que será el promedio) y un resultado indicando si se ha aprobado o no el curso. La clave primaria será la cédula de identidad.
7) Una tabla llamada “Tareas” donde se guardan los archivos y datos que los docentes suben para cada tarea: letra, tests de corrección, posible solución, escala de calificaciones y fechas de inicio y fin. Su clave primaria será un campo llamado “Id_Tarea” (identificador de tarea).
8) Tablas para guardar las escalas de calificaciones disponibles. Se les llamará Escala_i donde i indica el número de escala. Tendrán los campos Valor_Inicial y Valor_Final que definen los rangos y Nota que es la nota que le corresponde a determinado rango. La clave primaria será un índice autonumérico Id. Se definirán arbitrariamente dos escalas.

Se tendrá entonces el siguiente esquema (donde en verde claro se destaca la clave primaria de cada tabla):

MODELO
Claves_Globales (Clave_Admin, Clave_Profes, Clave_Alumnos) Datos_Admin (Nombre, Apellido, CI, Mail, Clave) Datos_Profesores(Nombre, Apellido, CI, Mail, Clave) Datos_Alumnos (Nombre, Apellido, CI, Mail, Clave) Notas (CI, Id_Tarea, Nota, Tarea) Resultados_Finales (CI, Nota, Resultado) Tareas (Id_Tarea, Letra, Tests, Posible_Sol, Escala, Fecha_Inicio, Fecha_Fin) Escala_i (Id, Valor_Inicial, Valor_Final, Nota)

Tabla VII-3 – Modelo en Webide3.

Se deberán realizar las validaciones apropiadas, que permitan evitar, o al menos minimizar la ocurrencia de errores. Por ejemplo, cuando el alumno sube una tarea se pedirá que efectivamente lo haga, de lo contrario aparecerá un mensaje de error, como podría ser “por favor suba una tarea” y la operación no será efectuada.

Controladores

Se tendrán tres usuarios de diferente índole: administrador, profesores y alumnos. Se creará un controlador para cada uno de ellos. Se llamarán admin, profesores y alumnos

respectivamente. Habrá también un controlador por defecto, que se llamará “default”, otro que se encargará de la autenticación y se llamará “aut” y otro que corregirá la tarea y asignará la nota que se llamará “corrector”. En la tabla siguiente se listan los mismos:

CONTROLADORES
Default
Aut
Admin
Alumnos
Profesores
Corrector

Tabla VII-4 – Controladores en Webide3.

A continuación se verá qué funciones es necesario incluir en cada controlador.

Default

Tendrá la función “index” (índice) y eventualmente alguna otra función que pueda llegar a requerirse.

Se implementará entonces:

- Index

Aut

Debe haber una página inicial que le permita al usuario o bien registrarse o identificarse si ya se registró.

No cualquier persona podrá registrarse; para poder hacerlo le será entregada a los usuarios una clave general, que luego podrán cambiar por una personal. Dicha clave es diferente según el tipo de usuario; el administrador puede elegirla, a los profesores les será entregada por el administrador y a los alumnos por los profesores.

La primera vez que alguien ingresa deberá registrarse. Para hacerlo, el usuario dispondrá de un formulario a través del cual ingresará sus datos (nombre, apellido, mail y cédula de identidad) y la contraseña general recibida; es a través de la última que se distingue entre los tipos de usuarios, y de esta forma se puede insertar el nuevo registro en la tabla que corresponde (por ej. si es alumno ha de ser insertado en la tabla Datos_Alumnos). Luego serán redirigidos a la interfaz que les corresponde.

Ya registrados, los usuarios podrán ingresar cuando lo deseen identificándose.

Se implementarán las siguientes funciones:

- Registrarse.
- Identificarse.
- Agregar usuario.

Admin

El Administrador tendrá las opciones de elegir la contraseña que le dará a los profesores para que puedan registrarse, ver datos de los profesores, editar su perfil y borrar la base de datos.

Dispondrá de una interfaz para acceder rápidamente a la opción deseada.

Se implementarán entonces las funciones siguientes:

- Interfaz.
- Fijar contraseña para el registro de profesores.
- Ver datos de los profesores
- Editar Perfil.
- Borrar base de datos.

Profesores

Deben de disponer de una interfaz apropiada y poder editar su perfil cuando lo deseen.

Se requerirá de un formulario a través del cual los profesores puedan subir los archivos y datos de interés de las tareas, los mimos serán almacenados en la tabla “Tareas” de la base de datos.

También deben disponer de una opción que les permita descargar los archivos que subieron.

Habrá una función para exportar la tabla de notas de cada tarea una vez finalizadas las mismas.

Se debe de estar preparado para la situación en que un alumno, una vez culminada la tarea y la corrección, requiera una revisión de la misma. Para ello se tendrá una función que a partir del ingreso de los datos CI e identificador de tarea devuelva el archivo entregado por el estudiante.

Habrá funciones auxiliares para hacer más completa la aplicación. Por ejemplo, los docentes podrán ver los datos de los alumnos, algo necesario para poder comunicarse con los mismos.

Agregamos así las siguientes funciones:

- Interfaz.
- Editar perfil.
- Formulario para subir tareas.
- Descargar archivos de las tareas.
- Descargar archivos de los estudiantes.
- Ver datos de los alumnos.

Alumnos

Deben de disponer de una interfaz apropiada y poder editar su perfil cuando lo deseen.

Será necesario disponer de un formulario que les permita subir las tareas.

Se deben tener funciones para que los estudiantes sean capaces de descargar la letra de las tareas y ver en todo momento sus calificaciones.

Se agregaron por tanto las siguientes funciones:

- Interfaz.
- Editar perfil.
- Formulario para subir tareas.
- Descargar letra de la tarea.
- Ver notas.

Corrector

Luego de ingresada la tarea, la misma será corregida haciéndola pasar por un conjunto de tests. Dependiendo de la cantidad de tests aprobados es que se asignará una calificación al estudiante. Necesitamos una función que se encargue de esto.

Asimismo debe haber una función que actualice la tabla de notas con la nueva calificación obtenida.

Se agregan las siguientes funciones:

- Obtener nota.
- Actualizar base de datos.

Vale la pena aclarar que las aquí descritas no son todas las funciones que contendrá el programa, las mismas llamarán a otras auxiliares para resolver los diferentes puntos, pero sí son representativas del esquema lógico que se quiere lograr.

Vistas

El objetivo de las vistas es el de proporcionarle una interfaz amigable al usuario que le permita una fácil interacción con la aplicación y rápidamente los direccione a dónde deseen ir.

Habrá formularios para permitir el ingreso de datos y archivos, como lo son el de identificación o el de subir tareas para los profesores.

Habrá otras vistas que muestran tablas para permitir la elección y descarga de archivos cargados en la base de datos.

Otras vistas se utilizarán para pedir confirmaciones, por ejemplo para saber si se está seguro de borrar la base de datos o para mostrar mensajes que confirmen que una operación se ha realizado correctamente (como por ejemplo el cambio de una contraseña).

Descripción del proceso básico

Considerando el flujo de trabajo básico para el Profesor y el Alumno, el proceso básico es el siguiente:

- Para el Profesor:

  - Mostrar página de bienvenida con enlaces a páginas para subir tareas, editar perfil, exportar notas, etc.
  - Si se ingresó a subir tarea:
    - Mostrar formulario para el ingreso de datos y archivos de las tareas.
    - Guardar la información ingresada.
  - Si se ingresó a exportar notas:
    - Mostrar tabla con las calificaciones y enlaces de descarga en diferentes formatos.
  - Si se ingresó a descargar archivo del estudiante:
    - Mostrar formulario para que se ingresen los datos del alumno que requiere ver su tarea y luego un enlace de descarga de la misma.
- Para el Alumno:
  - Mostrar página de bienvenida con enlace a página de selección de tareas.
  - Desplegar página de elección de tareas y redirigir a la página de la tarea seleccionada.
  - Verificar si la tarea elegida está habilitada.
  - Si la tarea no está habilitada: No permitir el acceso a la misma, informar al alumno de tal situación.
  - Mostrar formulario para que el alumno suba su tarea.
  - Capturar la entrada dada por el usuario.
  - Guardar una copia del código ingresado en un archivo en el servidor, en un directorio adecuado dentro del de la aplicación.
  - Crear un archivo con extensión .py, formado por el código ingresado por el alumno y los tests cargados previamente por el profesor.
  - Ejecutar dicho archivo con el intérprete Python del servidor donde se encuentra la aplicación.
  - Si la ejecución lleva demasiado tiempo, de manera que la aplicación deja de responder (por ejemplo si se da un bucle infinito en el código ingresado):
    - Cortar la ejecución y redirigir a una página donde se informe sobre el error de tiempo de espera agotado (o “timeout”) y que se asignará una calificación igual a cero.
    - Calificar al Alumno (guardar registro en la tabla destinada a ello) con cero puntos en la tarea correspondiente.
  - Capturar la salida y los errores provenientes de la ejecución en variables apropiadas. Calcular en base a estos datos el porcentaje de tests correctos.
  - Asignar un puntaje de acuerdo a una escala de calificaciones.
  - Actualizar la tabla Notas con la nueva calificación.
  - Eliminar el archivo que fue creado y ejecutado.
  - Mostrarle al alumno su calificación.

VII.2.5. Diagrama de componentes UML

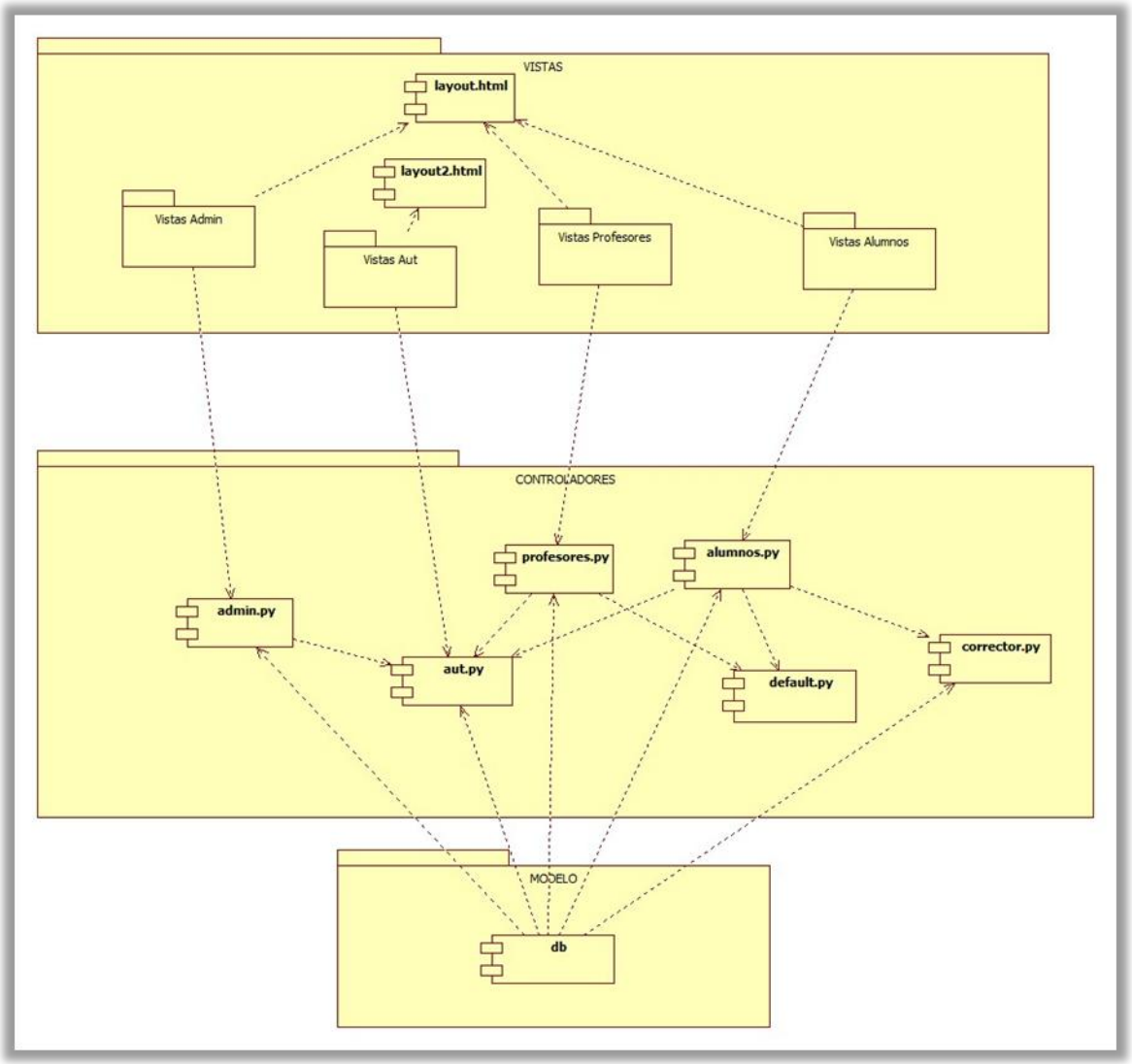

FIGURA VII-45 – Diagrama de componentes Webide3 – 1.

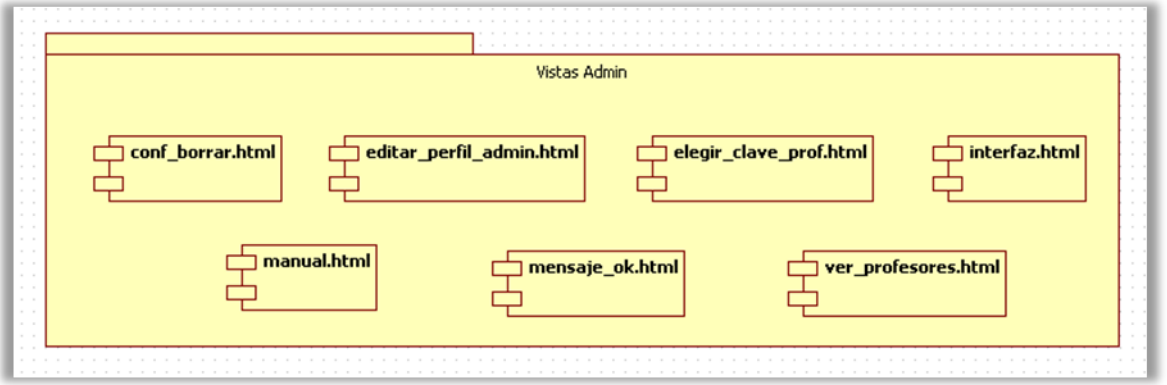

FIGURA VII-46 – Diagrama de componentes Webide3 – 2.

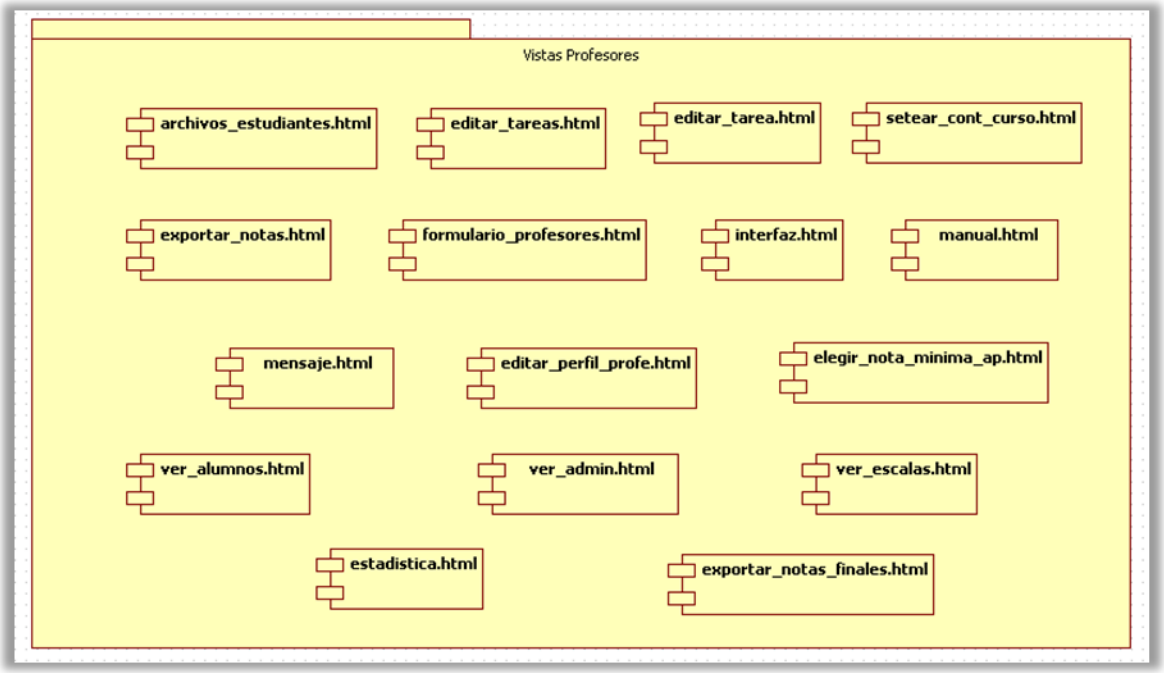

FIGURA VII-47 – Diagrama de componentes Webide3 – 3.

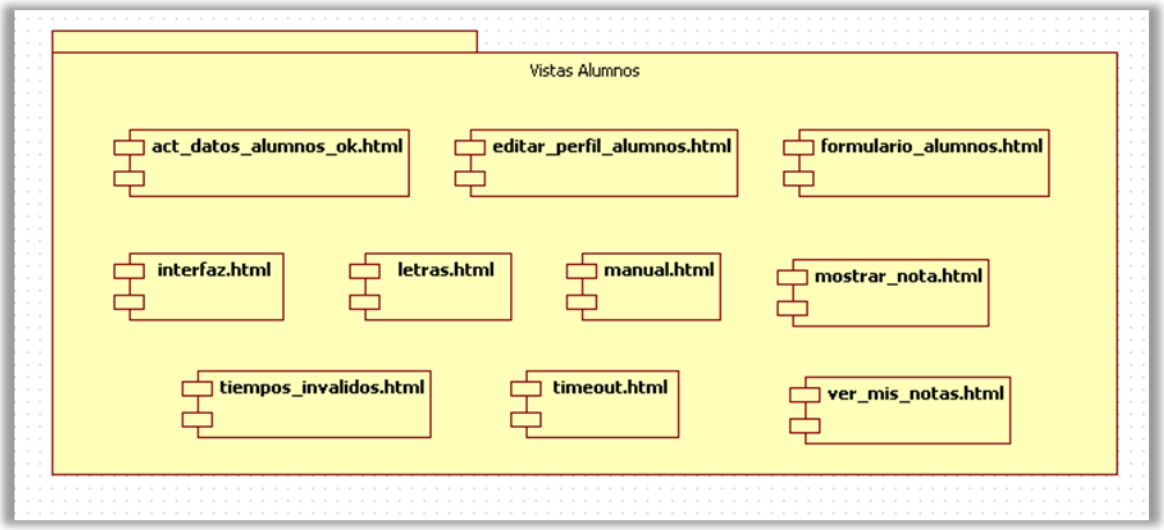

FIGURA VII-48 – Diagrama de componentes Webide3 – 4.

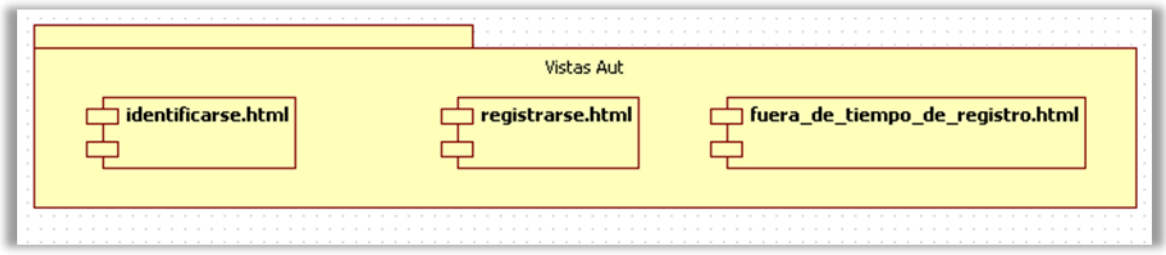

FIGURA VII-49 – Diagrama de componentes Webide3 – 5.

VII.2.6. Implementación

Se verá cómo se implementó el flujo más importante del programa, que es cuando el alumno se identifica, sube su tarea, la misma se corrige y se asigna automáticamente la calificación obtenida.

- Identificación

El estudiante se identifica completando un formulario con su cédula de identidad y contraseña. Se usa el comando SQLFORM.factory para implementarlo ya que no es necesario realizar operaciones de inserción/actualización con ninguna base de datos en particular. Los datos ingresados se almacenan en las variables form.vars.CI y form.vars.Clave respectivamente. Se escribe en la función "identificarse" del controlador "aut":

```
form = SQLFORM.factory (Field ('CI','integer',requires = IS_NOT_EMPTY("Por
favor ingrese su cédula"),

                        Field ('Clave', 'password', requires = IS_NOT_EMPTY
("Por favor ingrese su clave")))
```

Y en la vista correspondiente se presenta el formulario escribiendo:

```
{{ =form }}
```

A continuación se valida que los datos sean correctos. Se comienza preguntando si la cédula pertenece o no a un alumno registrado. De ser una cédula válida se pregunta si la contraseña ingresada es la que le corresponde a dicho alumno. De pasarse ambas validaciones se guarda en la variable session.CI la cédula del estudiante (que servirá para saber de aquí en más quién inició la sesión) y se redirecciona a la interfaz de alumnos. En caso contrario no se envían datos y se despliega un mensaje de error.

```
#selecciono el alumno por la cédula ingresada, los datos de alumnos se
encuentran en la tabla Datos_Alumnos

esta_alumno = db (db.Datos_Profesores.CI == (form.vars.CI)).select()

#si la cédula pertenece a la tabla la selección no será vacía (tendrá
longitud no nula)

If len (esta_alumno) > 0:

     #el alumno pertenece

     # selecciono la clave que le corresponde al alumno que ingresó

     rows = db(db.Datos_Alumnos.CI==form.vars.CI).

                              select (db.Datos_Alumnos.Clave)

     #pregunto si la clave ingresada (que está en form.vars.Clave) y la que
debe tener coinciden

     if form.vars.Clave == rows[0].Clave:

     # coinciden por lo que guardo las variables necesarias y redirecciono

           session.CI = form.vars.CI
```

```
            redirect (URL(request.application, 'alumnos', 'interfaz'))
     else:
            # si no es así se indica con un mensaje de error
            response.flash = 'Cédula y contraseña no coinciden'
else:
     # cédula incorrecta
     response.flash = 'Datos incorrectos'
```

- Elegir Tarea

A través de una opción disponible en el menú principal el estudiante elige la tarea que desea entregar. Cuando lo hace se almacena en la variable session.Id_Tarea el número de tarea seleccionado. El menú es dinámico, sólo aparecen como disponibles para seleccionar aquellas tareas que ya haya ingresado el docente.

Para generar el menú se forma una lista con todos los identificadores de tarea, que se encuentran en la tabla Tareas y al ingresar a una en particular se direcciona a la URL dónde se subirá el archivo, enviándole como información el número de tarea. Luego de direccionar se guardará el número de tarea en la variable session.Id_Tarea puesto a que se necesitará luego. El código es el siguiente:

```
# selecciono los identificadores numéricos de las tareas ya ingresadas
num_tareas_rows = db(db.Tareas).select(db.Tareas.Id_Tarea,
                                       orderby = db.Tareas.Id_Tarea)
#formo una lista llamada ul_aux con links a la entrega de tareas
#a través de args en el comando URL es que envío el número de tarea
ul_aux = []
for num_tareas in num_tareas_rows:
     ul_aux +=  [ ('Tarea ' + str (num_tareas.Id_Tarea), False,
                  URL ('exportar_notas', args = num_tareas.Id_Tarea]) ) ]
```

- Subir Tarea

Para subir la tarea se presenta un formulario. La tarea ingresada se guardará en la tabla Notas por lo que se utilizó el comando SQLFORM para implementarlo. No obstante, como el alumno ya se identificó y eligió el número de tarea, no es necesario hacer que los ingrese nuevamente, serán fijados recordando que se encuentran en las variables session.CI y session.Id_Tarea. Esta asignación debe hacerse previo a que el formulario sea aceptado. Luego se guarda la referencia a la tarea del alumno en una variable llamada session.entrada y se redirecciona a la función que calcula y asigna la nota llamada "obtener_nota" (que se encuentra en el controlador "corrector"). Se escribe en la función "formulario_alumnos" del controlador "alumnos":

```
# formulario para subir la tarea

form = SQLFORM(db.Notas, fields=['Tarea'], submit_button='Enviar')

# fijo CI e Id_Tarea

form.vars.CI = session.CI

form.vars.Id_Tarea = session.Id_Tarea

# guardo la referencia hacia donde se encuentra la tarea del alumno

session.entrada = form.vars.Tarea

# redirecciono a obtener_nota

redirect (URL (request.application, ’corrector’, ’obtener_nota’))
```

En este punto se conoce quién es el estudiante (a través de session.CI), qué tarea va a subir (a través de session.Id_Tarea) y cuál es el código que ingresó (a través de session.Tarea), por lo tanto estamos en condiciones corregir el archivo, obtener la nota y asignarla. De esto se encarga la función “obtener_nota” del controlador “corrector”.

“obtener_nota” debe realizar los siguientes pasos: armar el archivo de corrección, ejecutar el código y capturar la salida, calcular el porcentaje de tests correctos y la nota acorde a una escala preestablecida y finalmente actualizar la base de datos.

- Armar el archivo de corrección

Se hace en la función “obtener_nota” del controlador “corrector”.

El código ingresado se guardará en una carpeta en el servidor llamada Temp con nombre igual a la cédula de identidad. Se hace así para, aprovechando que la cédula es única, evitar que múltiples usuarios que acceden simultáneamente a la aplicación escriban sobre el mismo archivo. Luego, se escriben en el mismo, el código del estudiante junto con los tests de corrección que había ingresado el profesor. El código es el siguiente:

```
# establezco la ruta dónde se guardará el archivo

arch= "applications/"+request.application+"/Temp/"+session.CI+".py"

# armo el archivo, comienzo por guardar el código del estudiante

archivo = open (arch, "w")

arch_alum = "applications/"+request.application+"/uploads/"+str
(session.Tarea)

alum = open(arch_alum, 'r')

codigoalum = alum.readlines()

archivo.writelines(codigoalum)

# selecciono los tests escritos por el profesor que están almacenados en
Tareas

rows = db(db.Tareas.Id_Tarea==session.Id_Tarea).select(db.Tareas.Tests)
```

```
# le agrego al archivo los tests, es análogo a cómo se hizo con el código
del alumno
```

- Ejecutar código y capturar salida

Se utiliza el método subprocess. Se distingue entre sistemas operativos. La salida se almacena en la variable pstdout en caso de una correcta ejecución y en pstderr en caso de que haya errores.

```
# se corre el subproceso distinguiendo según el sistema operativo

so = os.name

if so == 'nt':

     proc = subprocess.Popen([arch_Python], stdout = PIPE, stderr = PIPE,

                                              shell = True)

else:

     proc = subprocess.Popen ('python' + arch_Python, stdout = PIPE,

                                       stderr = PIPE, shell = True)

# guardo la salida en pstdout y los errores en pstderr

pstdout, pstderr = proc.communicate()
```

- Calcular el porcentaje de tests correctos

De esto se encarga la función "corregir" del controlador "corrector". Recibe como entrada el resultado de la ejecución del archivo de corrección y calcula el porcentaje de test correctos. Se la llama desde "obtener_nota" pasándole como parámetro la salida del subprocess:

```
if pstderr == '':

     puntaje = corregir (pstdout)

else:

     puntaje = corregir (pstderr)
```

En este punto se encuentra en la variable local "puntaje" el porcentaje de tests correctos.

La función "corregir" se describe más abajo, en el ítem "Funciones Auxiliares".

- Obtener nota acorde a una escala preestablecida

De esto se encarga la función "calificar" del controlador "corrector". Recibe como entrada el porcentaje de tests correctos (que es puntaje, la salida de la función "corregir") y asigna la nota de acuerdo a una escala preestablecida. La nota obtenida se guarda en la variable session.calif. Se llama desde "obtener_nota" escribiendo:

```
session.calif = calificar(puntaje)
```

La función "calificar" se describe más abajo, en el ítem "Funciones Auxiliares".

- Actualizar Nota

De esto se encarga la función “actualizar_nota”. Se llama desde “obtener_nota” pasándole como parámetros session.CI, session.Id_Tarea y session.calif que son los datos necesarios para saber qué registro de la tabla Notas actualizar.

```
actualizar_nota (session.Id_Tarea, session.CI, session.calif)
```

Esta función se describe más abajo, en el ítem “Funciones Auxiliares”.

- Mostrar Nota

Luego de obtener la nota se direcciona al alumno a una página que le muestra al mismo la calificación obtenida, escribiendo:

```
# redirección a mostrar_nota

aux = URL (request.application, 'alumnos', 'mostrar_nota')

redirect (aux)
```

Recordando que la calificación obtenida se encuentra en la variable session.calif, se escribe en la vista de alumnos “mostrar_nota”:

```
{{=session.calif}}
```

- Funciones auxiliares
    - Corregir

Para implementarla se hará previamente la siguiente observación.

Cuando se ejecuta un unittest en Python, las salidas posibles son:

1) Si todos los tests fueron correctos:

RAN “cantidad de tests” in “cantidad de tiempo”s

OK

2) Si hubo fallas

“Mensajes de error dependiendo del tipo de fallas”

RAN “cantidad de tests” in “cantidad de tiempo”s

FAILED (failures= ”cantidad de fallas”)

3) Si hubo errores

“Mensajes de error dependiendo del tipo de errores”

RAN “cantidad de tests” in “cantidad de tiempo”s

FAILED (errors= ”cantidad de errores”)

4) Si hubo errores y fallas

“Mensajes de error dependiendo del tipo de errores y de fallas”

RAN “cantidad de tests” in “cantidad de tiempo”s

FAILED (failures = “cantidad de failures”, errors = “cantidad de errores”)

Se observa que hay palabras clave, a saber: RAN, errors y failures.

Luego de “RAN” aparece la cantidad de tests, luego de “failures” la cantidad de fallas (si las hay) y luego de “errors” la cantidad de errores (si los hay).

Por tanto, para saber la cantidad de tests ejecutados se busca el número luego del “RAN”, para saber la cantidad de fallas el número luego del “failures=” y para saber la cantidad de errores el número luego del “errors=”. Obtenidos estos datos se calculará el porcentaje de aciertos como:

```
(Total Tests – (Cant. Fallas + Cant. Errores)/Total Tests)*100%
```

Para encontrar las palabras clave, se utilizó la función rfind de Python, que devuelve el índice donde comienza la última aparición de la palabra buscada dentro de un string y -1 en caso de no encontrarla.

Por ejemplo, para hallar la cantidad de fallas, se escribe lo siguiente:

```
# la salida del unittest se encuentra en la variable astring

# busco la palabra “failures=”

fail = astring.rfind('failures=')

if fail != -1:

     # si fail es distinto de -1 quiere decir que se encontró la palabra

     # voy a formar una palabra, llamada “fallas” donde guardaré el nº de
fallas

     j = 0

     fallas = “”

     # “failures=” tiene 9 caracteres, me paro luego de ellos y de ahí
hasta el “)” es que se encuentra el valor que quiero

     while astring [fail+9+j] != ")":

            fallas = fallas + astring [fail+9+j]

     j = j + 1

     # convierto el string donde tengo el nº de fallas a entero

     fallasInt = int (fallas)

else:
```

```
     # si no se encontró la palabra entonces no hay fallas
     fallasInt = 0
```

- Calificar

Recibe como entrada el porcentaje de tests correctos que se guardará en la variable local "puntaje". Luego se obtiene la escala de calificaciones que seleccionó previamente el docente, y según a qué rango de esa escala pertenezca el puntaje se asigna la nota. El código es el siguiente:

```
# selecciono la escala de calificaciones
rows = db (db.Tareas.Id_Tarea == session.Id_Tarea).
                        select (db.Tareas.Escala_de_Calificaciones)
escala = rows[0].Escala_de_Calificaciones
# asigno la nota según la escala
if escala == 1:
     Filas = db ((db.Escala_1.Valor_Inicial <= puntaje) &
            (db.Escala_1.Valor_Final >= puntaje)). select (db.Escala_1.Nota)
# así sigue según el número distinto de escalas…
# finalmente obtengo la nota y la retorno
nota = filas[0].Nota
return nota
```

- Actualizar Nota

Recibe como parámetros de entrada CI, Id_Tarea y Nota. Actualiza el campo Nota de la tabla Notas correspondiente al registro identificado por CI e Id_Tarea. El código es el siguiente:

```
db ((db.Notas.CI==CI) & (db.Notas.Id_Tarea == Id_Tarea)). update
(Nota=Nota)
```

VII.3. Comparación entre Webide 2 y Webide 3

Las aplicaciones descritas anteriormente en este capítulo presentan varios puntos en común. Si bien lo hacen de maneras distintas, implementan la misma funcionalidad básica, que es la corrección automática de tareas de programación; poseen interfaces similares y dan la posibilidad de manejar el mismo tipo de registros. No obstante, pueden diferenciarse entre ambas, algunas características de importancia. A continuación se resumen y señalan las mismas en un esquema comparativo.

WEBIDE 2	WEBIDE 3
El sistema de agregado de usuarios a grupos puede llegar a ser poco práctico en la improbable situación de que se inscriban demasiados usuarios que no pertenezcan al curso.	La contraseña de registro deben proporcionarla los administradores y docentes de forma segura por sus propios medios.
Tanto el administrador como los profesores pueden ingresar y eliminar usuarios.	No se permite insertar y/o eliminar registros por parte del administrador y los docentes.
No hay restricciones de tiempo en cuanto a la posibilidad de registrarse.	Una vez que el profesor sube la primer tarea ya no es posible que los alumnos continúen registrándose.
El número de tareas es fijo.	El docente puede subir y/o eliminar tareas en cualquier momento según crea conveniente.
Es altamente interactiva, pueden corregirse las letras y códigos de las tareas directamente en la aplicación.	Es poco interactiva, las correcciones deben realizarse fuera de la aplicación.
No es posible descargar los archivos correspondientes a letras de tareas, códigos ingresados, etc.	Es posible descargar todos los archivos ingresados.
Se realiza un informe de errores de compilación y/o ejecución.	No se realiza un informe de errores.
No se realiza una estadística de los resultados del curso.	Se realiza una estadística en base a los resultados finales del curso.

Tabla VII-5 – Comparación entre Webide2 y Webide3.

Webide 2 es la aplicación que cumple con la definición de IDE y por ende sería la más apropiada para implementar en un curso de programación, considerando los objetivos primarios de este proyecto, pero posee algunas limitaciones que Webide 3 tiene solucionadas. Las más trascendentes son la cantidad fija de tareas y la imposibilidad de descargar los archivos que contienen las letras de las tareas, soluciones y código ingresado por los alumnos, si se deseara.

VII.4. Propuesta a futuro

Luego del análisis realizado y considerando las características de cada aplicación, se puede concluir que la evolución de la solución debe ir en la dirección de Webide 2, que ya incorpora lo desarrollado en Webide 1, agregando algunas de las funcionalidades de Webide 3, como, principalmente, la cantidad variable de tareas según las necesidades del docente o del curso en el que se vaya a implementar y la posibilidad de descargar archivos de interés.

Mirando más adelante y estudiando, por un lado, las necesidades que podrían surgir en el dictado de un curso de programación, y por otro, las eventuales mejoras en el desempeño de una aplicación del estilo descrito en el párrafo anterior, se ve la posibilidad de incluir funcionalidades que no han sido contempladas hasta aquí, y podrían ser beneficiosas tanto para docentes como para estudiantes. Una de ellas es la existencia de un foro para intercambiar opiniones, formas de resolución de problemas, etc., entre alumnos y con la participación de profesores. Incluso podría llegarse a la creación de una wiki, donde fuera posible dejar registradas experiencias en problemas concretos, por ejemplo. También sería de utilidad agregar una sección para que los docentes suban material de estudio y los alumnos puedan descargarlo o consultarlo en línea. Podría desarrollarse más profundamente un sistema de estadísticas, donde quedaran reflejados, además de datos sobre calificaciones y resultados globales de cursos, otros sobre tiempo de resolución de tareas, calculados por usuario y por tarea. Por otro lado, podría considerarse la facilidad que se tiene, mediante la herramienta Web2py en la que están basadas las aplicaciones, de traducir a distintos idiomas las interfaces y los mensajes desplegados mientras se trabaja en ellas.

Son mejoras que podrían colaborar en buena medida con el dictado de un curso y que son realizables a un bajo costo.

VIII. Implementación de las aplicaciones en Moodle

VIII.1. Agregado de recursos y actividades

Para implementar las aplicaciones Webide en un curso de Moodle, cualquiera de ellas sea, por ejemplo en un curso llamado “Pyhton”, se debe crear un nuevo recurso. Para implementar un acceso rápido, por ejemplo a WebIde 1, se debe realizar lo siguiente:

1) Crear la etiqueta del recurso
2) Agregar el recurso web o actividad
3) Confirmar funcionamiento

Para tener acceso y crear recursos se debe tener los permisos necesarios siendo administrador del curso en Moodle. Luego de ingresado como administrador del curso Python, se debe agregar la etiqueta del recurso. A continuación se ingresa en modo edición, el cual permite crear nuevas actividades, definiendo la fecha que se publicará el nuevo recurso, como se muestra en la imagen siguiente:

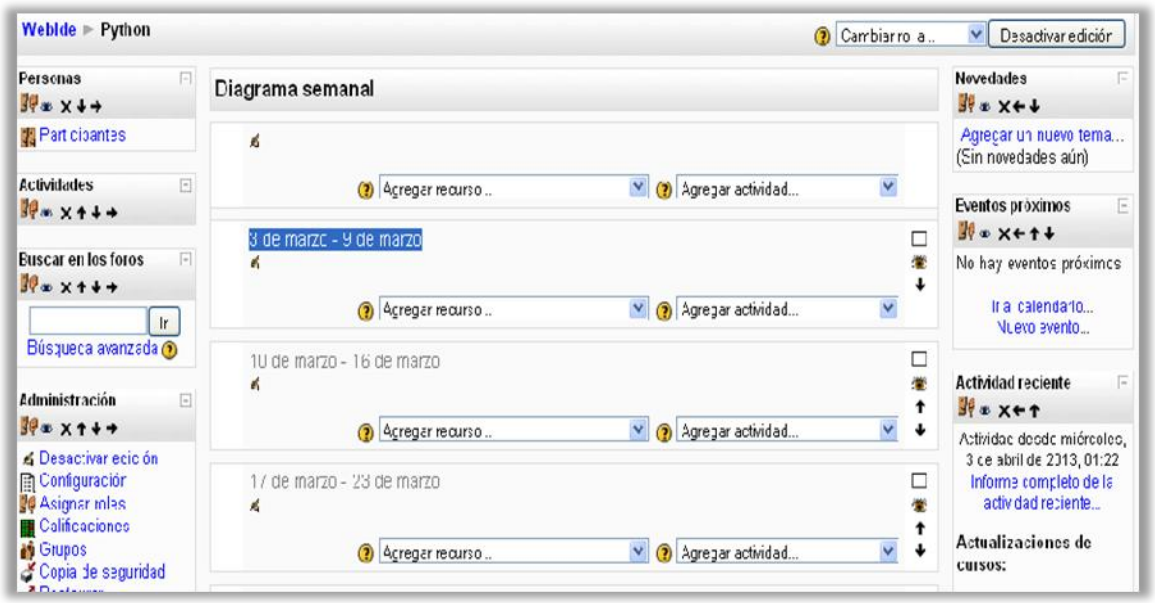

FIGURA VIII-1 – Modo edición y elección de fecha.

Luego se debe seleccionar la opción “Insertar etiqueta” para crear la etiqueta del recurso. En este caso la llamaremos a la etiqueta “Compilador Python en línea”.

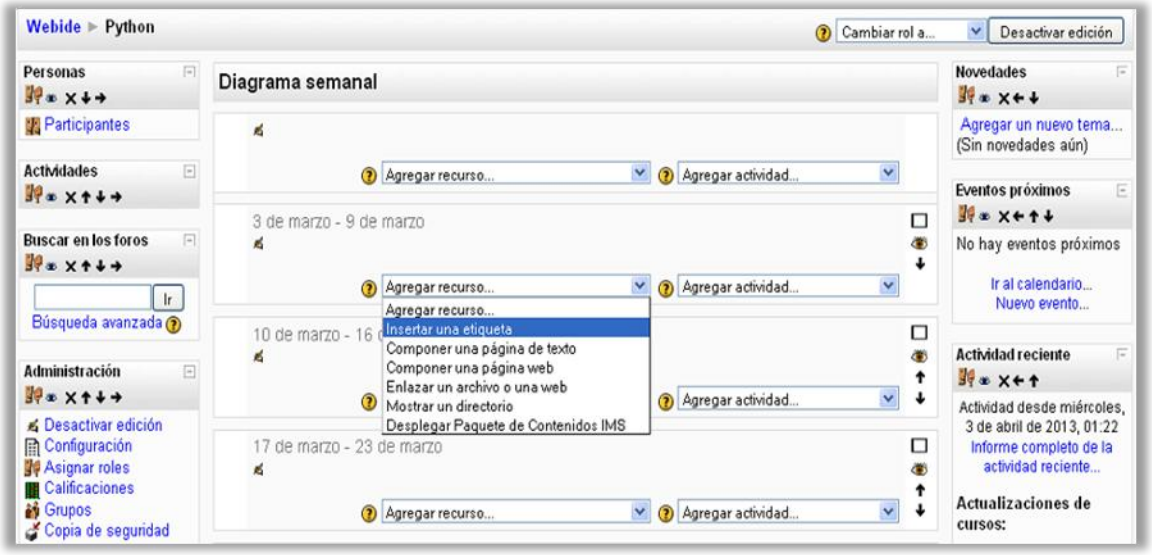

FIGURA VIII-2 – Creación de etiqueta.

Luego se debe seleccionar otro recurso, en este caso se desea un enlace a la dirección del compilador Web, seleccionando “Enlazar un archivo o una web”.

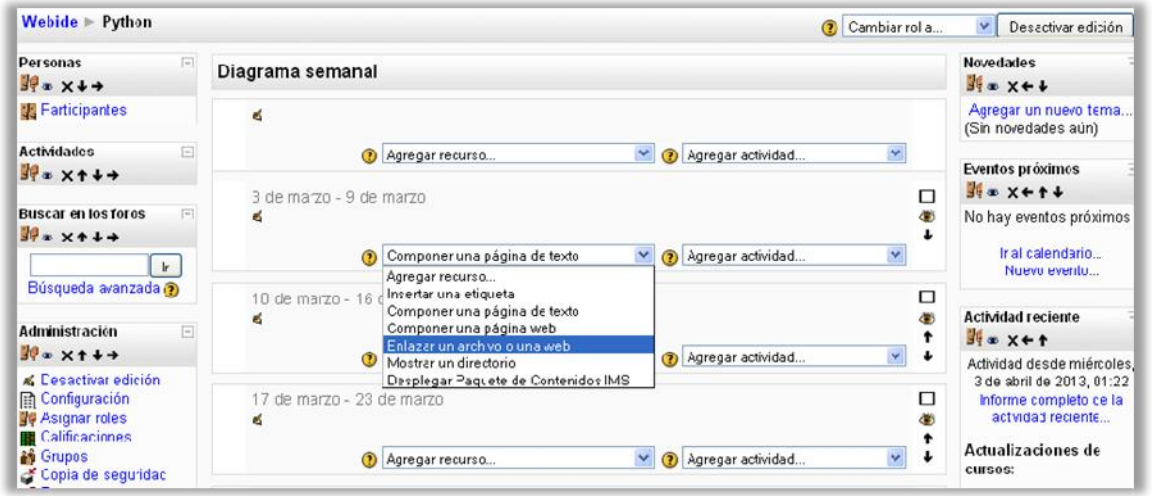

FIGURA VIII-3 – Creación de enlace a Webide 1.

De esta manera se nombra el enlace como “Compilador Python”, y se fija la dirección web donde se encuentra la página de bienvenida del compilador Webide 1, de igual modo se realiza para Webide 3, pero además se agrega una actividad (“Actividad no en línea”) que no será vista por los estudiantes, en este caso se asignará la calificación de la tarea 1 (por los profesores), a través de la importación de un archivo CSV (ver figura siguiente).

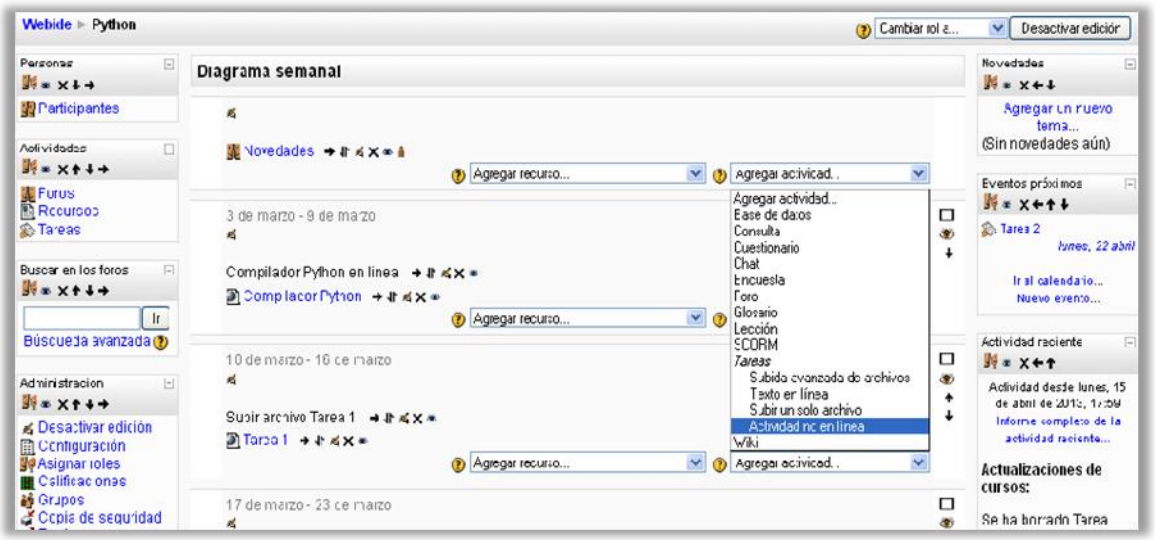

FIGURA VIII-4 – Creación de una actividad.

La manera de hacer invisible una actividad a los estudiantes (siempre vista por los profesores), es activando la opción “ocultar” de ésta actividad, quedando de la siguiente manera:

FIGURA VIII-5 – Vista de etiquetas, enlaces y actividades ocultas.

Luego de este proceso, cuando el estudiante ingresa al curso, ve la figura siguiente, pero no se le presenta la actividad “Calificación Tarea 1”, ya que es solamente vista por los profesores.

FIGURA VIII-6 – Vista desplegada al alumno.

Ingresando al enlace "Compilador Python" es redirigido al compilador Webide 1, e ingresando a "Tarea 1" es redirigido a la aplicación Webide 3 para subir el archivo.

La manera de importar las calificaciones a través de un archivo CSV para fijar las calificaciones en Moodle, se explica a continuación.

VIII.2. Importación de calificaciones mediante archivo CSV

Como se ha visto, Moodle tiene la posibilidad de crear actividades dentro de un curso, como subir archivos, crear tareas en línea, tareas no en línea, etc. La calificación de la actividad, puede hacerse de forma manual (directamente) o automática, y además existe la posibilidad de importar calificaciones por medio de archivos. Los tipos soportados para importar calificaciones son CSV y XML (Moodle 1.9).

¿Por qué la elección de archivo CSV?

La elección de calificar mediante archivo CSV se debe a sus ventajas frente a XML al importarlo en Moodle, teniendo en cuenta que cualquiera de los dos formatos es soportado por Moodle y Web2py. La diferencia radica en que Moodle permite pre-visualizar la importación de calificaciones, es decir una vista previa antes de importar, permitiendo seleccionar las columnas a mapear de un archivo CSV, y en XML no presenta esta facilidad, simplemente importa directamente.

Procedimiento:

1) Tener el archivo CSV a importar.
2) Acceder como administrador del curso con los permisos adecuados.
3) Acceder al menú Administración →Calificaciones.
4) Seleccionar del menú Importar → Archivo CSV.
5) Seleccionar archivo, codificación y separador.
6) Mapear Usuario con Columna_i y mapear Tarea con Columna_k

7) Botón Subir calificaciones.
8) Verificar que las calificaciones fueron almacenadas correctamente.

En la siguiente figura se muestra cómo seleccionar el tipo de archivo a importar en Moodle.

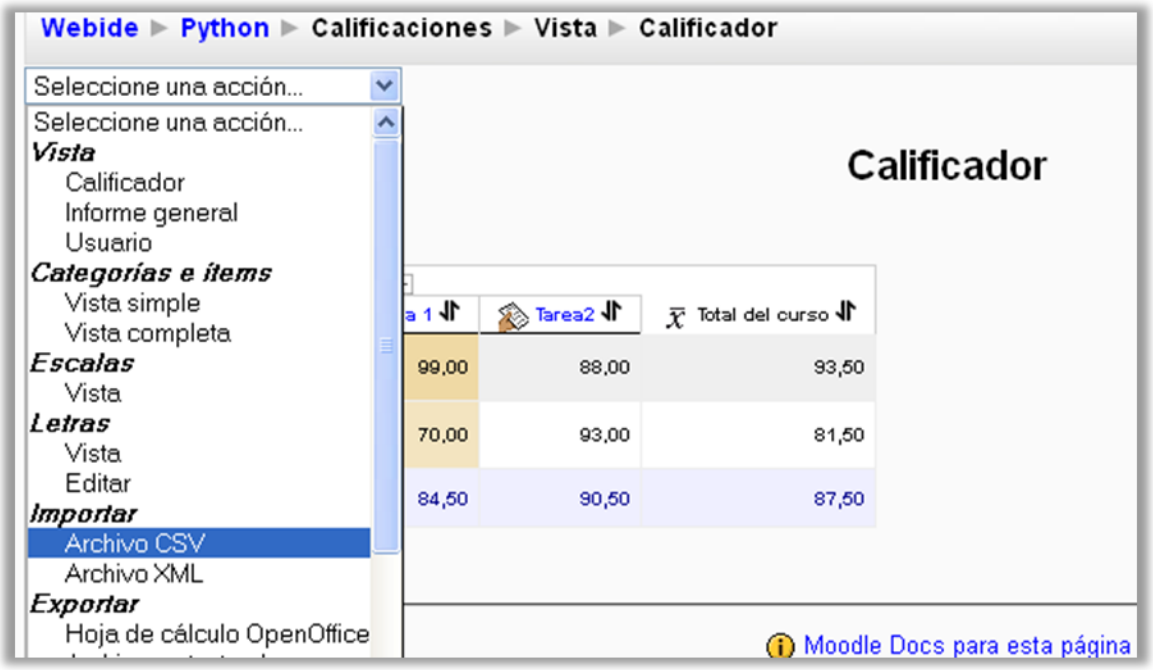

FIGURA VIII-7 – Importar calificaciones desde archivo CSV en Moodle.

Luego de elegir el formato del archivo a importar, como se muestra en la figura anterior, se despliega una serie de parámetros para seleccionar. Aquí se debe seleccionar el archivo que desea importar ("examinar"), tipo de separador (en este caso la "coma"), el tipo de codificación ("UTF-8"), y otros dos parámetros que quedan fijados por defecto ("escalas de texto" y "previsualizar filas").

FIGURA VIII-8 – Parámetros para importar archivo CSV.

Luego de seleccionar lo antes mencionado, se deben subir las calificaciones. Para poder pre-visualizar el archivo, deberá acceder al botón “Subir calificaciones”, lo que desplegará lo siguiente:

FIGURA VIII-9 – Mapeo de usuarios y calificaciones.

Luego de este proceso, si todo es correcto, se informa que la importación se ha realizado con éxito. Luego se despliegan las calificaciones.

FIGURA VIII-10 – Calificaciones almacenadas en Calificador.

Se debe tener en cuenta que para subir un archivo CSV, es necesario contar con el archivo en este formato con los separadores correspondientes, como lo es la coma en este caso. También considerar la codificación que se utiliza, por ejemplo en este caso se codificó con UTF-8. En general si no se especifican correctamente estos parámetros del archivo, pueden darse inconvenientes. Otro detalle a tener presente, es que el usuario para poder importar archivos, deberá poseer dos permisos (puede ser el administrador, o profesor del curso), Permiso General para Importar Calificaciones, y Permiso para Importar Calificaciones en Formato Determinado.

Permisos:

```
moodle/grade:import ("Import grades") = Allow

gradeimport/csv:view ("Import grades from CSV") = Allow
```

IX. Conclusiones

Finalizado el proyecto WebIDE, podemos afirmar que se ha cumplido con el primero de los objetivos propuestos inicialmente. Desarrollamos un compilador web (Webide 1) que funciona de la manera que esperábamos. Somos conscientes que aún puede mejorarse en el aspecto funcional y sobre todo desde el punto de vista del diseño y código de programación, pero al tratarse de una primera versión, lo fundamental es que satisface los requisitos para el funcionamiento deseado.

En lo que concierne al segundo objetivo, si bien no conseguimos llevar a cabo la creación de un objeto de aprendizaje con las características deseadas, desarrollamos una solución alternativa que satisface la necesidad planteada en el inicio del proyecto, incluso con alguna ventaja respecto a la posibilidad descartada. Por ejemplo, las aplicaciones creadas no necesitan de la existencia de una plataforma de aprendizaje para funcionar, garantizando su independencia, pero de todas maneras pueden intercambiar con ella información a través de archivos, si es necesario. Webide 3 no cumple con la definición de IDE, pero implementa una manera eficaz de manejar un sistema de carga y de corrección de tareas.

En retrospectiva, nos damos cuenta que no realizamos un correcto análisis de requerimientos para el desarrollo de una solución alternativa, una vez descartada la opción del objeto de aprendizaje, lo que nos llevó a tomar un camino equivocado e implementar dos aplicaciones que si bien son diferentes, tienen varios puntos en común. Ambas aplicaciones funcionan correctamente considerando para lo que están diseñadas, pero observamos que la solución óptima sería implementar las funcionalidades de las dos en una sola. El camino a seguir sería incorporar las funcionalidades principales de Webide 3 a Webide 2. En nuestro caso, dadas las condiciones actuales, esta modificación demandaría cierto tiempo del que no disponemos, y deberíamos recurrir a solicitar extender el límite de la entrega del proyecto, hecho que no se justifica considerando que el producto final se compone de prototipos.

Igualmente, pensamos que los prototipos son suficientemente elaborados como para servir de base a un desarrollo con calidad de producción que pueda ser probado en un curso real. Un desarrollo a partir de los prototipos construidos, para prueba en clase o para uso en un curso formal, debería comenzar con una revisión o complemento de la etapa de análisis, que permitiera llegar a una especificación de requerimientos, es decir, establecer claramente lo que se espera de la aplicación, y cuáles serán las pruebas con que se deberá demostrar que la misma efectivamente cumple lo requerido.

Posiblemente en una situación real, la participación de un cliente, que podría ser un profesor o un grupo de profesores a punto de dar un curso, hubiera aportado hacia un análisis más certero de los requerimientos, colaborando así con un mejor desarrollo de la solución.

Por otra parte, debemos mencionar que fue importante el aporte del curso Gestión de Proyecto brindado por la Facultad de Ingeniería al comienzo de este emprendimiento. Nos fue de utilidad para organizarnos y establecer los fundamentos para el desarrollo del proyecto.

Pudimos superar algún tipo de inconveniente que se dio en el transcurso de esta tesis, como el hurto de la laptop de uno de los integrantes, gracias a las precauciones tomadas desde un principio, en este caso el respaldo sistemático de información (que calificaría como “control de riesgos”), por lo que el hecho no repercutió de manera trascendente sobre el trabajo.

Hemos adquirido cierta experiencia en Python, un lenguaje considerado a nuestro entender en crecimiento, y en el paradigma de programación Modelo Vista Controlador (MVC), de uso notoriamente extendido por el mundo actualmente.

Obtuvimos, además y principalmente, experiencia en el desarrollo de un proyecto, el primero para nosotros de esta magnitud. Logramos, analizando lo realizado a lo largo del año en que se llevó a cabo la tesis, identificar los puntos en los que no estuvimos acertados y considerar los caminos que deberíamos haber tomado en cada caso.

X. Bibliografía

Abelló, Urpí, Rodríguez, Estévez. (n.d). Extensión de Moodle para facilitar la corrección automática de cuestionarios y su aplicación en el ámbito de bases de datos. 07.extensión.pdf. Descargado el 28 mayo 2012 .Universidad Politécnica de Catalunya (UPC) , Universitat Oberta de Catalunya (UOC).

Di Pierro, M. (2007). Libro de web2py en español. Versión 1. Consultado abril 2012 – marzo 2013. http://www.latinuxpress.com/books/drafts/web2py.

Exe. Exe Exelearning. (n.d). Consultado 15 octubre 2012. http://exelearning.org/wiki.

Gonzalez Barbone, V., Anido Rifon, L.. (2009). From SCORM to Common Cartridge: A step forward, History and support. Elsevier.

González Duque, R.. (n.d). Python para todos. Creative Commons Reconocimiento 2.5 España.

IMS Global. (n.d). Consultado 15 octubre 2012. http://www.imsglobal.org/specificationdownload.cfm.

IMS Global. (n.d). Join the Common Cartridge & Learning Tools Interoperability™ Alliance. Consultado 20 octubre 2012. http://www.imsglobal.org/cc/jointhealliance.cfm.

IMS Global. (n.d).Learning Tools Interoperability™. Consultado 14 setiembre 2012. http://www.imsglobal.org/lti/.

IMS Global. Página oficial. (n.d). Consultado 15 agosto 2012. http://www.imsglobal.org.

IMS Global. (2010). IMS Global Learning Tools Interoperability™ Basic LTI Implementation Guide . Consultado 15 octubre 2012. http://www.imsglobal.org/lti/blti/bltiv1p0/ltiBLTIimgv1p0.html.

IMS Global. (2011). IMS GLC Common Cartridge Profile: Implementation. Consultado 15 octubre 2012. http://www.imsglobal.org/cc/ccv1p2/imscc_profilev1p2-Implementation.html.

IMS Global. (2012).IMS Global Learning Tools Interoperability™ Implementation Guide. Consultado 19 agosto 2012. http://www.imsglobal.org/LTI/v1p1/ltiIMGv1p1.html.

Jesukiewicz, P. (2009). SCORM 2004 Content Agreggation Model (CAM) Versión 1.1 (4ta edición).

Jesukiewicz, P. (2009). SCORM 2004 Run-Time Environment (RTE)Versión 1.1 (4ta edición).

Jesukiewicz, P. (2009). SCORM 2004 Sequencing and Navigation (SN) Versión 1.1 (4ta edición).

Lara Fuillerart, J. (2009). Manual de referencia para el profesorado. Versión 1.9. Consultado 17 octubre 2012. http://www.issuu.com/ostos/docs/16990042-moodle-manual-de-referencia-para-profesor.

Massa, E.. (2012). *Objetos de Aprendizaje: Metodología de desarrollo y evaluación de la calidad*. Universidad Nacional de la Plata, Facultad de Informática.

Moodle. (n.d). Importar calificaciones. Consultado 5 agosto 2012. http://docs.moodle.org/all/es/Importar_calificaciones.

Python Programming Language – Official Website. Consultado abril 2012 – marzo 2013. http://docs.python.org/dev/library.

Python Software Foundation. (2010). *Python v2.7.4 documentation*. Consultado abril 2012 – marzo 2013. http://docs.python.org/2/library

Python Software Foundation. (2012). *Python v3.3.1 documentation*. Consultado diciembre 2012 – marzo 2013. http://docs.python.org/3/library

San Crístobal, E.. (2010). *Metodología, estructura y desarrollo de interfaces intermedias para la conexión de laboratorios remotos y virtuales a plataformas educativas*. Escuela Técnica Superior de Ingenieros Industriales de la Universidad Nacional de Educación a Distancia.

Universidad Carlos III de Madrid. (2008).Software Libre para la creación de material docente. Consultado 15 julio 2012. http://www.uc3m.es/portal/page/portal/biblioteca/sobre_la_biblioteca/servicios/taller_aula/profesores/noticias/Software-Libre-para-la-creacion-de-material-docente%5B2%5D.pdf

Wikipedia. Base de datos. Consultado 2 agosto 2012. http://es.wikipedia.org/wiki/Base_de_datos.

Wikipedia. Entorno de desarrollo integrado. Consultado 30 Mayo 2012. http://es.wikipedia.org/wiki/Entorno_de_desarrollo_integrado.

Wikipedia. Estado del arte. Consultado 1 agosto 2012. http://es.wikipedia.org/wiki/Estado_del_arte.

Wikipedia. Google Chrome. Consultado 1 agosto 2012. http://es.wikipedia.org/wiki/Google_chrome.

Wikipedia. Html. Consultado 3 mayo 2012. http://es.wikipedia.org/wiki/Html.

Wikipedia. JavaScript. Consultado 3 mayo 2012. http://es.wikipedia.org/wiki/JavaScript.

Wikipedia. Lenguaje Unificado de Modelado. Consultado 9 noviembre 2012. https://es.wikipedia.org/wiki/Lenguaje_Unificado_de_Modelado.

Wikipedia. Modelo Vista Controlador. Consultado 3 mayo 2012. http://es.wikipedia.org/wiki/Modelo_Vista_Controlador.

Wikipedia. Moodle. Consultado 4 agosto 2012. http://es.wikipedia.org/wiki/Moodle.

Wikipedia. Mozilla Firefox. Consultado 14 junio 2012. http://es.wikipedia.org/wiki/Mozilla_firefox.

Wikipedia. Núcleo Linux. Consultado 23 junio 2012. http://es.wikipedia.org/wiki/Núcleo_Linux.

Wikipedia. Objeto de aprendizaje. Consultado 22 abril 2012. http://es.wikipedia.org/wiki/Objeto_de_aprendizaje.

Wikipedia. Python. Consultado 12 abril 2012. http://es.wikipedia.org/wiki/Python.

Wikipedia. SCORM. Consultado 22 abril 2012. http://es.wikipedia.org/wiki/SCORM.

Wikipedia. StarUML. Consultado 9 noviembre 2012. http://en.wikipedia.org/wiki/StarUML.

Wikipedia. Ubuntu. Consultado 23 junio 2012. http://es.wikipedia.org/wiki/Ubuntu.

Wikipedia. Windows Internet Explorer. Consultado 14 junio 2012. http://es.wikipedia.org/wiki/Windows_Internet_Explorer.

Wikipedia. Xampp. Consultado 4 agosto 2012. http://es.wikipedia.org/wiki/Xampp.