PROYECTO FINAL DE CARRERA

CATALYDE: UN ENTORNO DE INICIACIÓN A LA PROGRAMACIÓN CON FINES EDUCATIVOS

Realizado por: Stanciu, Catalin Costin
Dirigido por: España Boquera, Salvador
Pastor Pellicer, Joan

Universidad Politécnica de Valencia,
Valencia, Febrero 2013

Ley de Alzheimer de la programación:
si lees un código que escribiste hace más de dos
semanas es como si lo vieras por primera vez.

— Dan Hurvitz

RESUMEN

En el presente proyecto se ha diseñado e implementado una herramienta de apoyo al aprendizaje en asignaturas de introducción a la programación. Esta herramienta, denominada *Catalyde*, proporciona una interfaz de usuario accesible mediante un navegador web en el que los alumnos pueden acceder al boletín de prácticas, editar las soluciones a los ejercicios propuestos, compilarlos y ejecutarlos. Si bien ya existen entornos de desarrollo vía web, lo que hace novedoso este entorno para la enseñanza de la programación es la posibilidad de que el alumno pueda resolver las distintas partes de los ejercicios de manera incremental. Es decir, el alumno puede marcar qué partes del código han sido desarrolladas por él y la herramienta utiliza la solución propuesta por el profesor (que permanece oculta) para las partes que todavía no ha completado. De ese modo el alumno puede entender mejor lo que se pide (porque puede probar el programa desde el inicio) y la validez de lo que va realizando sin tener que esperar a completarlo todo al final. También permite evaluar de manera automática la corrección de algunos ejercicios por medio de tests.

Palabras clave: herramienta, desarrollo, aplicación web, cliente-servidor, pedagógico, programación, nodejs, mongodb.

ABSTRACT

In this project, a novel tool called *Catalyde* has been designed and implemented. This tool is intended to guide students of first courses of Computer Science to develop their programming skills. By means of a web interface, users are able to edit the solutions of exercises proposed by their teacher, to compile and to execute them. One of the unique features of this environment is the possibility of developing solutions in an incremental way by marking and unmarking which parts of the solution are due to the students and which parts should made use of the hidden teacher's solution. In this way, students can test the program and can better understand what it is expected from them. They can also check parts of the code without waiting for the complete solution. The tool can also automatically check the validity of some exercises by means of tests.

Keywords: tool, development, web application, client-server, pedagogical, programming, nodejs, mongodb.

La programación es una carrera entre los desarrolladores, intentando construir mayores y mejores programas a prueba de idiotas, y el universo, intentando producir mayores y mejores idiotas. Por ahora va ganando el Universo.

— Rich Cookhave

AGRADECIMIENTOS

En primer lugar quisiera agradecer a mis directores de proyecto, Salvador España Boquera y Joan Pastor Pellicer, todo el apoyo, el interés y el trabajo que han realizado para poder llevar a cabo este proyecto y por todos aquellos momentos en los que me han animado a seguir adelante.

A María José Castro, que me ha dado la oportunidad de empezar un proyecto estando cerca de un grupo de investigación, permitiéndome aprender de gente experimentada.

Agradecer a mis compañeros del laboratorio, Ximo, Joan y Marcos por todo los momentos compartidos durante el desarrollo del proyecto.

A mis compañeros de curso, que durante los años de universidad me han alegrado los días con su buen humor.

A mis padres que siempre han estado allí y me han animado, que si no fuera por ellos nunca hubiera llegado hasta aquí.

Por último a Amparo que siempre ha estado a mi lado apoyándome y animándome en todo, ofreciéndose a ayudarme en todo momento. Por ser mi ángel de la guarda.

A todos vosotros, Gracias.

ÍNDICE GENERAL

1 INTRODUCCIÓN 1
- 1.1 Motivación 1
- 1.2 Objetivos 3
- 1.3 Estructura de la memoria 5

2 ENTORNOS Y PLATAFORMAS DE DESARROLLO 7
- 2.1 Situación actual 7
- 2.2 Cursos y entornos web para aprender a programar 8

3 ANÁLISIS DE LA SOLUCIÓN 15
- 3.1 Especificación de requisitos software 16
  - 3.1.1 Requisitos funcionales 16
  - 3.1.2 Requisitos no funcionales 17
- 3.2 Casos de uso 18
  - 3.2.1 Sistema principal 19
  - 3.2.2 Sistema administración 25

4 DISEÑO DE LA SOLUCIÓN 31
- 4.1 Modelo Cliente y servidor 31
- 4.2 Modelo de datos y conceptual 35
- 4.3 Diseño de la interfaz 38
  - 4.3.1 Inicio de Sesión 38
  - 4.3.2 Vista general de la aplicación 39
  - 4.3.3 Abrir una práctica 40
  - 4.3.4 Cerrar una práctica 42
  - 4.3.5 Mover una práctica 43
  - 4.3.6 Menú de la aplicación 43
- 4.4 Modelo de ejecución y compilación 45
- 4.5 Definición de una práctica 51

5 TECNOLOGÍAS APLICADAS 55
- 5.1 Tecnologías usadas en la parte cliente 56
  - 5.1.1 Termlib 56
  - 5.1.2 jQquery 56
  - 5.1.3 jQuery UI 56
  - 5.1.4 CodeMirror 57
- 5.2 Tecnologías usadas en el servidor 57
  - 5.2.1 Nodejs 57
  - 5.2.2 Modulos de Node.js 58
  - 5.2.3 MongoDB 60
  - 5.2.4 GDB 60
  - 5.2.5 Apparmor 60
- 5.3 Herramientas adicionales 61
  - 5.3.1 SVN 61
  - 5.3.2 Aptana Studio 61
  - 5.3.3 StarUML 62

5.3.4 Draw.io 62
5.4 Integración de tecnologías 63
6 SEGURIDAD, ESCALABILIDAD Y EFICIENCIA 67
6.1 Escalabilidad 67
6.2 Seguridad 69
7 CONCLUSIONES Y TRABAJO FUTURO 73
7.1 Conclusiones 73
7.2 Trabajo futuro 74
8 ANEXOS 77
8.1 Anexo 1: Creación de la práctica 77
8.2 Anexo 2: Ejemplo de uso 83
8.3 Poster enviado a JDINF'12 87
REFERENCIAS 89

ÍNDICE DE FIGURAS

Figura 1 Captura de la interfaz de udacity. 11
Figura 2 Captura de la interfaz de codeacademy. 12
Figura 3 Checkio, plataforma donde en forma de juego se plantean retos de programación. 13
Figura 4 E-Lab system es otro entorno web para la validación automática de problemas de programación Delev and Gjorgjevikj [19]. 14
Figura 5 CAP: corrector automático de programas para la evaluación continua y el aprendizaje autónomo descrito en Sapena et al. [38]. 14
Figura 6 Actores del sistema. 18
Figura 7 Casos de uso relacionados con la gestión de la sesión 19
Figura 8 Caso de uso: Listar grupos 20
Figura 9 Caso de uso iniciar pantalla completa. 23
Figura 10 Casos de uso relacionados con la gestión del código escrito por el alumno. 24
Figura 11 Casos de uso relacionados con la gestión de usuarios en modo administración 25
Figura 12 Casos de uso relacionados con la gestión de prácticas en modo administración. 27
Figura 13 Casos de uso relacionados con el usuario de tipo administrador. 29
Figura 14 Cliente-Servidor 32
Figura 15 Modelo de objetos del cliente 34
Figura 16 Esquema de objetos/datos 37
Figura 17 Boceto de inicio de sesión 39
Figura 18 Errores al proporcionar datos incorrectos en el inicio de sesión. 39
Figura 19 Boceto principal de la aplicación 40
Figura 20 Boceto de la práctica abierta 41
Figura 21 Acción abrir práctica 41
Figura 22 Acción cerrar práctica 42
Figura 23 Organizar prácticas abiertas 43
Figura 24 Boceto opciones con detalle 44
Figura 25 Boceto archivos con detalle 44
Figura 26 Diagrama del flujo de ejecución de un programa 48
Figura 27 Diagrama del módulo de ejecución 50
Figura 28 Ejemplo de los tres tipos de bloques que soporta el sistema. 52

Figura 29 Muestra de un trozo de una plantilla de código. 53
Figura 30 Esquema de los elementos que componen una práctica en relación a la creación de la misma. 53
Figura 31 Tecnologías utilizadas 55
Figura 32 Figura de muestra de Aptana Studio 61
Figura 33 Figura de muestra de Draw.io 63
Figura 34 Servidor: Interacción entre NodeJs y todos los módulos utilizados 63
Figura 35 Cliente: Tecnologías utilizadas en el cliente 64
Figura 36 Gráfico que muestra la relación entre las tecnologías. 65
Figura 37 Página de inicio de sesión de la aplicación Catalyde. 83
Figura 38 Mensajes de error que se pueden obtener al iniciar sesión. 84
Figura 39 Vista general de la aplicación. 84
Figura 40 Opciones de la barra de menú. 84
Figura 41 Ejemplo de práctica abierta. 85
Figura 42 Programa en ejecución. 86
Figura 43 Ejemplo de error de compilación. 86

ACRÓNIMOS

API Application Programming Interface

UML Unified Modeling Language

MDA Model Driven Architecture

MOOC Massive Open Online Course

MS-DOS Microsoft Disk Operating System

NPM Node Package Manager

LLVM Low Level Virtual Machine

GCC GNU Compiler Collection

NOSQL Not Only Structured Query Language

SQL Structured Query Language

BSON Binary JavaScript Object Notation

JSON JavaScript Object Notation

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

HTML HyperText Markup Language

XML eXtensible Markup Language

TCP Transmission Control Protocol

MVC Model-View-Controller

CSS Cascading Style Sheets

1 INTRODUCCIÓN

1.1 MOTIVACIÓN

Aprender a programar es una tarea muy compleja que requiere la adquisición de conocimientos y habilidades que difícilmente se pueden realizar sin la propia experimentación del alumno. La parte práctica de las asignaturas de programación tiene un papel muy importante, tanto o más que las propias clases de teoría, ya que es cuando el alumno demuestra el entendimiento de los conceptos teóricos y los interioriza.

Es natural preguntarse si la forma en la que se aprende a programar es la más adecuada. Actualmente los alumnos utilizan durante las prácticas el mismo tipo de herramientas que utilizan los programadores profesionales, si bien esto último es una afirmación muy general, puesto que dichos programadores utilizan un amplio abanico de herramientas y de opciones. La idea principal es que el alumno terminará usando dichas herramientas, luego parece natural que las use desde el principio.

En este trabajo se han planteado las ventajas de un entorno especial para aprender a programar y se han sospesado los beneficios e inconvenientes de dicho entorno.

Es bien sabido que lo más difícil al aprender a programar es entender la parte lógica, saber cómo unir las piezas del repertorio que ofrecen los lenguajes de programación. El resto de aspectos como la sintaxis o detalles concretos de un lenguaje de programación son accesorios y casi más un "mal necesario" que un fin en sí mismo. Sin embargo, a nivel práctico puede que el alumno necesite más tiempo para aprender estas reglas sintácticas que para aprender la "lógica" de un programa (la parte algorítmica). Dicho de otro modo: si el alumno dispone de un tiempo limitado para estudiar o para realizar las prácticas, todo el tiempo que le quite darse cuenta de que falta un punto y coma, es tiempo que no utiliza para entender que no puede usar una variable antes de darle un valor. Este tipo de cosas se aprenden normalmente por experiencia propia. Se retiene más cometiendo un fallo que escuchando las indicaciones del profesor para evitarlo.

Programar supone ser capaz de convertir un modelo mental de "lo que tiene que hacer o resolver un programa" en el diseño y la posterior implementación que lo realice. En muchas ocasiones, el alumno tiene una visión muy incompleta y vaga de lo que está pidiendo el profesor.

Si el alumno pudiese probar el programa que va a implementar se podría hacer una mejor idea de lo que le están pidiendo. Esto, ciertamente, no ocurre en el mundo real, pero podríamos decir que es como las ruedecitas de una bicicleta: los ciclista profesionales ya no las necesitan.

Por tanto, lo que se propone en este trabajo es el diseño y la implementación de un entorno que, por sus características, no se limita a ser un simple entorno de desarrollo, sino que incluye características únicas con orientación pedagógica especialmente pensadas para aprender a programar.

La herramienta que se propone convierte al profesor en un usuario más activo durante la realización de las prácticas, ya que la práctica contiene más "lógica subyacente". La idea general es que el alumno tiene incrustado en el boletín zonas interactivas que pueden ser meros cuestionarios tipo test, botones de configuración o zonas de edición de código. El código que rellena el alumno no tiene por qué ser un programa completo, sino que puede ser el cuerpo de una función o una parte concreta a resolver. El alumno podrá probar, si el profesor así lo desea, cómo se comporta el programa que tiene que realizar. Para tal fin, el profesor ha escrito una solución del programa a la que el alumno no tiene acceso.

En este punto el lector podría pensar que nada impide, en un entorno convencional, que el profesor le proporcione al alumno una versión compilada de una solución del ejercicio. Si bien esto es cierto, lo que hace única a esta herramienta es la posibilidad de poder marcar y desmarcar las distintas zonas que escribe el alumno para que él mismo pueda resolver una parte y validarla sin tener que esperar a terminar absolutamente todo el programa. De este modo, el alumno puede validar las distintas zonas del programa poco a poco y ver los errores de manera más local e inmediata. Aunque esto no es del todo "realista", se podría considerar como las ruedecitas de las bicicletas que se irían quitando progresivamente.

Otra forma de evaluar el código escrito por los alumnos podría ser un oráculo, es decir, que el profesor construiría una serie de tests que se ejecutarían con el programa y esto indicaría si funciona correctamente Douce et al. [21]. Es parecido a lo que se utiliza en muchos concursos de programación para validar la solución de un problema.

El entorno que se explicará a continuación tendría especial interés para alumnos que aprenden a programar en ingenierías que no son necesariamente “informática”. La mayoría de estas carreras tienen una asignatura de iniciación a la programación y normalmente el lenguaje de programación aprendido es C, por su utilidad en la industria. Para poner un ejemplo concreto, en la Escuela Técnica Superior de Ingeniería del Diseño (ETSID) se utilizó Turbo C++ de Borland durante muchos años, incluso cuando ya se consideraba arcaico por ser un entorno Microsoft Disk Operating System (MS-DOS)

en máquinas con el sistema operativo Windows. Actualmente se utiliza Dev-C++[1] y es importante mencionar que en muchas ocasiones el profesor tiene escasa capacidad para decidir el entorno, ya que puede venir impuesto por limitaciones en la elección del sistema operativo. Este ejemplo ha motivado la realización un entorno web. A continuación se resumen los principales motivos para optar por un entorno web:

- La no dependencia del sistema operativo, ya que por medio de los navegadores web se consigue la ejecución multiplataforma.
- El poder monitorizar la actividad del alumno cuando realiza ejercicios en casa, ya que se guardan históricos de acceso a la aplicación.
- El poder utilizar un modo examen en el que se tenga la monitorización del alumno[2] en todo momento evitando copias por correo o chat.
- Al ser entorno web, se permite el acceso a distancia, por tanto incluso los cursos podrían ser no presenciales.

1.2 OBJETIVOS

El objetivo principal del presente Proyecto Final de Carrera es llevar a cabo la implementación de un entorno de desarrollo, especialmente dedicado a la programación, para proporcionar al usuario la infraestructura necesaria para el aprendizaje de la tarea de programar. Este objetivo se puede desglosar en varios puntos que serán los siguientes:

- Que el alumno pueda acceder a las prácticas correspondientes a su grupo y pueda editar su código, compilarlo y ejecutarlo. El resultado quedará guardado entre sesiones y podrá acceder de nuevo o guardarlo en local en caso de estar permitido por la configuración.
- Que el alumno pueda trabajar con los programas de manera incremental, es decir, el profesor podrá dividir el código en apartados y el alumno podrá activar o desactivar su propio código de modo que pueda ejecutar el programa independientemente de que se hayan completado todos los apartados. Este mecanismo permite disponer de una realimentación o "feedback" sin tener que esperar a la finalización de todo el programa para poder verificar el funcionamiento de cada parte. También es posible que

1 Dev-C++ es un entorno de desarrollo integrado para programar en el lenguaje C/C++.Usa MinGW que es una versión de GCC (GNU Compiler Collection) como su compilador.

2 Existen herramientas de captura periódica de pantalla, pero para un entorno web se requiere funcionalidades diferentes como la detección de salir del modo de pantalla completa y de la pérdida del foco en la aplicación.

un trozo realizado por un alumno en un programa más sencillo se reutilice en otros programas más complejos para así poder dividir un problema grande en varios más fáciles de entender.

- Ofrecer un entorno accesible vía Internet permitiendo al alumno despreocuparse de instalar y de la arquitectura de la máquina donde está programando o del sistema operativo, que son algunos de los inconvenientes de la mayoría de los entornos de programación. Que se pueda utilizar de manera independiente de la plataforma (Unix/Linux, MacOs, Windows, etc.). Esto facilita enormemente el trabajo de mantenimiento por parte de los técnicos, y por parte del alumno si quiere trabajar desde casa, puesto que no hace falta instalar el entorno en cada máquina donde se vaya a utilizar, con los problemas de compatibilidad que esto pueda acarrear (por ejemplo, en muchas asignaturas se utiliza el programa Dev-C++ en entorno Windows y algunos alumnos utilizan otros sistemas operativos).

- Que el sistema lleve un seguimiento del trabajo realizado por el alumno durante la elaboración de las prácticas (guardar las etapas intermedias y no únicamente el resultado final).

- El profesor podrá incluir mecanismos de comprobación automática de la solución propuesta por el alumno mediante el uso de "baterías de prueba", que son conjuntos de pares entrada y de salida del programa, o de los valores de determinadas variables que permiten comprobar de manera automática, aunque sea parcialmente, la corrección del código. De esta manera, el alumno puede saber si lo que ha programado es correcto y esto facilita el trabajo del profesor.

- Hacer más atractivo el desarrollo de las prácticas añadiendo, por ejemplo, una representación visual de la salida del programa o del estado de determinadas variables con el fin de motivar y ayudar a comprender su funcionamiento. También puede ser sencillo incluir bibliotecas gráficas que hagan más atractivo el desarrollo de programas, que en muchos casos se limitan a manipular cadenas de texto en una consola.

- Como tarea adicional, será posible complementar las actividades de programación, mediante preguntas de test o pequeños enunciados de comprensión que pueden entregarse mediante la misma herramienta.

- Ofrecer una forma de programar en la que se fusionen los enunciados con los ejercicios prácticos y preguntas teóricas en un mismo documento visible por el usuario de la aplicación. Es decir, la herramienta puede servir también para mostrar apuntes de teoría donde los lectores podrán probar los ejemplos.

- Ofrecer una interacción en tiempo real del alumno con el programa mediante la terminal del entorno web.

1.3 ESTRUCTURA DE LA MEMORIA

En esta sección se presentará de forma breve la organización por capítulos del documento, exceptuando este capítulo de introducción:

- *Entornos y plataformas de desarrollo*
  Capítulo 2: Se explicará la situación de las prácticas de programación desde hace unos años hasta la actualidad, así como las herramientas que se han venido usando a lo largo del tiempo y las herramientas disponibles a día de hoy teniendo en cuenta las últimas tendencias en cuanto al aprendizaje.

- *Análisis de la solución*
  Capítulo 3: Se realizará una descripción de los requisitos y necesidades del sistema.

- *Diseño de la solución*
  Capítulo 4: Se realizará la descripción del sistema en lo que a diseño se refiere, diferenciado entre diseño de la aplicación, modelo de datos, interfaz de usuario y modelo de ejecución.

- *Tecnologías utilizadas*
  Capítulo 5: Se explicarán las principales tecnologías utilizadas para el desarrollo de la aplicación y del proyecto

- *Seguridad, escalabilidad y eficiencia*
  Capítulo 6: Se explicarán los aspectos principales de la aplicación relacionados con la seguridad, escalabilidad y eficiencia.

- *Conclusiones y trabajo futuro*
  Capítulo 7: Se enumerarán las principales conclusiones del trabajo presentado y, además, se explicarán las mejoras y funcionalidades futuras que se pueden añadir a la aplicación.

- *Anexos*
  Capítulo 8: En forma de anexos se presentarán un manual detallado para creación de prácticas y un breve manual de usuario de la aplicación.

- *Referencias*
  Se enumerará la lista de referencias a artículos y bibliografía en general utilizada para la realización del presente proyecto.

2 ENTORNOS Y PLATAFORMAS DE DESARROLLO

2.1 SITUACIÓN ACTUAL

Actualmente, en la mayoría de cursos y asignaturas de iniciación a la programación los alumnos utilizan un entorno medianamente profesional, pero esto no siempre ha sido así. Antiguamente no existían entornos integrados de desarrollo y el programador tenía que conformarse con un simple editor de texto y un compilador que podía utilizar de forma independiente. Con la evolución de los lenguajes de programación y la necesidad para generar software han surgido los entornos de desarrollo. Estas herramientas tienen sus pros y sus contras en cuanto a usabilidad en el campo de la programación.

El uso de estos entornos en la realización de las prácticas tiene beneficios, ya que acerca más la forma de trabajar del alumno a un entorno profesional. Aparte, un entorno de desarrollo integrado suele agrupar una serie de herramientas como: un editor de texto, un compilador o un intérprete, un depurador, control de versiones y otras características que ayudan a un mejor y más rápido desarrollo del software. Todo esto convierte al entorno en una herramienta muy potente para los que saben usarla.

Por otro lado los entornos de desarrollo integrados presentan inconvenientes de cara a un programador novel. Estos inconvenientes están relacionados directamente con la cantidad de utilidades que ofrecen, dado que muchas veces para entornos profesionales se necesitan cursos especializados para aprender a utilizar la herramienta. Habitualmente se asume que el alumno ya conoce algún lenguaje de programación y lo que persigue es acelerar la tarea de programar utilizando los entornos integrados. Lo que para un programador profesional puede ser una ventaja, para un programador novel se puede convertir en un inconveniente ya que el entorno actúa de barrera entre la tarea de aprender y el alumno, teniendo éste que hacer un doble esfuerzo: aprender a programar y aprender a usar el entorno de programación.

Cada vez más hay una tenencia a favorecer la tarea de programar al alumno novel con la utilización de entornos que resultan fáciles de configurar y manejar.

El profesor tiene un papel importante en la realización de las prácticas por parte de los alumnos. Tradicionalmente, el profesor se ha limitado a elaborar un boletín con los enunciados de las prácticas. En algunos casos, el alumno tenía que partir de cero y en otros el profesor le proporciona una plantilla a rellenar.

Es importante mencionar que, en muchas ocasiones, el profesor tiene escasa capacidad para decidir el entorno, que puede venir impuesto por limitaciones en la elección del sistema operativo. A continuación se muestran las ventajas que tendría un entorno cliente-servidor vía web:

- Poder monitorizar la actividad del alumno cuando realiza ejercicios en casa, ya que el sistema registra un historial de acceso a los recursos.
- Tener un modo examen.
- Poder usar el entorno incluso para educación a distancia.

En la actualidad existe una infinidad de aplicaciones que sirven de apoyo a la enseñanza. Entre ellas, podemos mencionar a PoliformaT (proyecto Sakai) y a Moodle, entre otros. Es natural plantearse la posibilidad de realizar este proyecto en el seno de estas plataformas tratándolo como un *plugin*, pero esto conllevaría una serie de problemas para realizar dicha integración. Algunas de las dificultades que podíamos encontrar son:

- Dificultad por tener que aprender cómo funciona la tecnología y poder probarla.
- Permiso para poder utilizar la plataforma, ya que al ser una plataforma ya creada no se pueden introducir errores funcionando en producción.
- Dependencia de la plataforma por parte de usuarios de otra.

2.2 CURSOS Y ENTORNOS WEB PARA APRENDER A PROGRAMAR

Durante los últimos años, los cursos cursos a distancia han experimentado un gran auge debido a las plataformas online para la realización de estos cursos. Estos cursos destinados a la comunidad en general son conocidos como Cursos Online Masivos (Massive Open Online Course (MOOC)). Existen varias plataformas (y cada vez más) que imparten este tipo de cursos, además muchos de ellos están relacionados directamente con el campo de la computación y, en concreto, de la programación.

Los cursos que se imparten de forma *online*, tanto oficiales como no oficiales, son innumerables; por tanto, en este apartado vamos a centrarnos en aquellas plataformas online diseñadas para aprender a programar o que incluyen contenidos altamente ligados con la programación. Se hace hincapié, como es de esperar, en aquellos que disponen de entornos para escribir y ejecutar código de forma online desde el propio navegador.

Dentro de los cursos y plataformas diseñadas para programar se pueden dividir en 3 tipos:

- Las que el código se desarrolla en local (en la misma máquina del usuario), se ejecuta en local y se verifica la solución enviando la respuesta mediante un formulario web. En este caso, el usuario se descarga el código, o un conjunto de archivos (si procede), edita el código y ejecuta el programa en su máquina con los datos que se le han proporcionado. Después, mediante un formulario vía web envía la solución tras ejecutar los casos. Usualmente se envía el material realizado por el alumno a distancia y los profesores se encargan de revisar los trabajos. Este tipo de plataformas no disponen de ningún tipo de ejecución ni edición online. El propio entorno web sirve sólo para verificar la solución y para mostrar el contenido del curso (vídeos, transparencias, manuales, etc...).

- Las que el código se ejecuta vía web en el lado cliente. El usuario escribe el código en el mismo navegador y los resultados se muestran en la misma interfaz web. Este sistema es bastante común en cursos de programación web: HyperText Markup Language (HTML), Cascading Style Sheets (CSS), Javascript y demás lenguajes web, donde el código se puede ejecutar fácilmente en el lado cliente. Existen otros sistemas más complejos que permiten traducir código de un determinado lenguaje (C, C++) a Javascript y ejecutarse en el mismo navegador. Es el caso de la combinación de un compilador que genere código de la máquina virtual LLVM (por ejemplo, el compilador clang) junto con el compilador de LLVM a Javascript denominado Emscripten. Esto permite ejecutar programas en el lado cliente escritos en C. También los conocidos applets de Java, donde se puede ejecutar código Java en un navegador.

- Las que el código se ejecuta en el servidor y los resultados se muestran en el lado cliente. Este servicio es el que más flexibilidad permite, puesto que el servidor es el que tiene todos los componentes necesarios para ejecutar el código del usuario y éste no debe preocuparse de tener instaladas las librerías necesarias.

También existen varias formas de tratar el código realizado por el usuario. En algunos casos se pide rellenar una función, y el sistema comprueba que se ha completado correctamente mediante los casos de prueba ocultos. Otras plataformas disponen de la opción de ejecutar un terminal, normalmente en lenguajes interpretados (Python, Javascript), y realizar ejecuciones de forma interactiva del código desarrollado. En algunos concursos de programación, se dispone de la posibilidad de ejecutar el programa realizado con una entrada diseñada por el mismo usuario. Aún así, la ejecución se realiza en el servidor y de forma no interactiva.

Aunque la herramienta presentada se centra en proporcionar una plataforma de apoyo para facilitar el aprendizaje a la programación en determinados cursos académicos, es interesante destacar diferentes plataformas online y MOOC's dedicados a la enseñanza mediante recursos web (transparencias, vídeos, etc).

Las plataformas MOOC's más destacadas son Udacity [9] y Coursera [8]. Ambas presentan diversos cursos de enseñanza de diferentes temáticas, la mayoría de los cuales están orientados a temas de informática y, en especial, de programación.

A mediados de 2011, *Stanford University*, presentó una serie de cursos online, totalmente gratuitos, sobre materias enseñadas en asignaturas del grado de computación. Aunque no eran los primeros cursos que se ofertaban en este formato, fueron ampliamente aceptados por la comunidad online. Los cursos fueron impartidos por profesores de prestigio, como es el caso del curso de Inteligencia Artificial, presentados por Peter Norvig (reconocido investigador en IA) y por Sebastian Thrun; y el de Aprendizaje automático, dirigido por Andrew Ng. Más tarde, la oferta de cursos se amplió, apareciendo cursos de Algoritmos (Tim Rougharden), Procesamiento del Lenguaje Natural (Daniel Jurafsky y Chris Manning), Modelado (Daphne Koller) y algunos más. Aunque en un principio no existía una plataforma tecnológica unificada, normalmente los cursos constan de vídeos, de preguntas (incluidas o no en los vídeos) y de un apartado de cuestiones tipo test o problemas, así como de exámenes que se evalúan de forma totalmente automática. La parte práctica, que es la que nos interesa, se desarrolla normalmente usando toolkits descargables de los programas donde el usuario debe rellenar los trozos de código que faltan, y después enviar los resultados mediante el mismo ejecutable descargado o completar un formulario en la página web.

Debido a la gran popularidad de estos cursos en la comunidad online, de forma simultánea aparecieron dos plataformas creadas exclusivamente para enseñar diversos cursos de forma online: Udacity y Coursera.

Existen varias diferencias entre ambas plataformas, tanto a nivel de contenidos como a nivel de las herramientas. Aún así, cabe destacar que Coursera incluye cursos presentados por diversos profesores de diferentes universidades alrededor del mundo, mientras que Udacity incluye cursos que, aunque muchas veces son impartidos por profesores de prestigiosas universidades, no están vinculados directamente a ninguna universidad.

En los cursos relacionados con la informática ambas plataformas proporcionan diferentes mecanismos de evaluación de los ejercicios donde hay que programar:

- Coursera sigue incluyendo descargables con el código, donde el usuario debe tener las herramientas necesarias en su ordenador.

- Udacity incluye un editor web (en principio para el lenguaje Python) donde el usuario desarrolla su código y se ejecuta sobre una serie de casos ocultos y se le muestra al usuario el resultado de la ejecución: si está correcto o no. Normalmente, el código a desarrollar se realiza después de una lección (o vídeo) para afianzar contenidos.

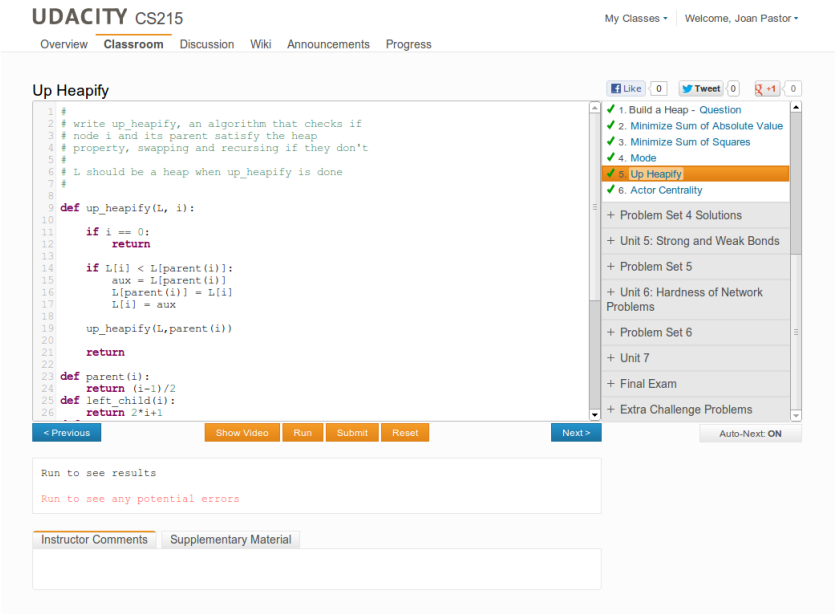


Figura 1: Captura de la interfaz de udacity.

Últimamente, han aparecido un sinfín de plataformas de educación online, incluso de habla española como es el caso de Miriadax o la reciente creada por la Universitat Politècnica de Valencia UPVx [12]. Sin olvidar también Khan Academy [1], que fue fundada antes que las mostradas anteriormente (2006) y que incluye cursos de diversas especialidades, ncluyendo algunos cursos de Informática y de programación. La plataforma proporciona una interfaz muy intuitiva que permite mostrar los resultados de una forma visual que ayuda a entender los contenidos del curso.

Aparte de estas academias online, existen muchas otras plataformas web, que presentan una gran variedad de cursos, la mayoría de ellos relacionados con la informática y el desarrollo en general. A continuación se enumeran algunas de ellas:

- Codeacademy [5] es una plataforma online creada exclusivamente para dar clases de programación de diferentes lenguajes. Entre ellos encontramos cursos para Python, Javascript, HTML, CSS, etc. Esta plataforma fue creada en Septiembre de 2011 y los cursos son totalmente gratuitos. La interfaz de los cursos es bastante intuitiva y nos permite en los cursos basados en Web (HTML, CSS, Javascript) ver los resultados de los ejercicios en el

mismo navegador. Uno de los pocos inconvenientes que tiene la plataforma es que a veces es difícil de identificar los errores, sobretodo en HTML y Javascript.

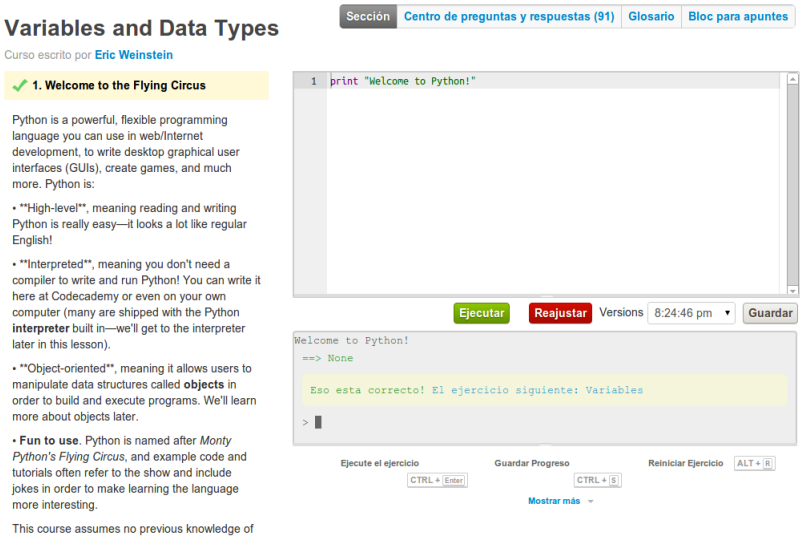


Figura 2: Captura de la interfaz de codeacademy.

- Udemy [10], engloba cursos de diferentes temáticas: gestión de empresas, arte, salud, idiomas, música, tecnología. Como característica destacable, los instructores pueden crear los cursos usando la plataforma y pueden subir contenido adicional para los cursos como transparencias, vídeo u otros. En la plataforma existen muchos cursos gratuitos, aún así existen cursos de pago cuyo precio varía entre $5 y $250.

- Codelearn [6], dispone de diferentes cursos de programación gratuitos. Esta plataforma dispone de una potente interfaz web que permite crear y ejecutar código en la misma plataforma. Learncode está pensado para reducir el hueco que existe entre los conocimientos aprendidos en la universidad/escuela y las demandas de programación en el mundo real. Como detalle importante de la plataforma, existe un apartado llamado "Code Garage" donde se pueden crear proyectos más complejos paso a paso como puede ser un Sudoku, Tic Tac Toe, el buscaminas entre otros.

- Codeschool [7] es una plataforma muy similar a las anteriores, donde se dispone de una gran cantidad de cursos, la mayoría orientada a tecnologías de desarrollo web (Javascript, Jquery, CSS3, etc) aunque también, aparecen tutoriales más generales como es el caso de Git Hub, IOS o Unix.

- Google también tiene su propia plataforma de cursos online, aunque dichos cursos están más orientados a plataformas móviles (Android) qué en vez de enseñar a programar desde cero.

Por otro lado, se incluyen los concursos de programación y/o algorítmica donde se pretende resolver problemas de diversa dificultad mediante la programación. Aunque estos concursos no están pensados para enseñar a programar desde cero, existen diferentes webs donde se presentan problemas ordenados por dificultad, por lo tanto se puede acceder a este tipo de sitios para afianzar los conceptos de programación ya sabidos y buscarles una aplicación a los problemas propuestos. Sobre las páginas web y sitios destacan: Universidad de Valladolid Online Judge, Codeforces, TopCoder, Checkio, USA Computing Olimpiad, Codechef,...

De los sitios mostrados, Check.io [4] destaca por su interfaz, y su forma de mostrar los problemas de programación como un juego donde debemos ir resolviendo problemas de dificultad progresiva para avanzar en la historia. También, al igual que la herramienta presentada en este proyecto, proporciona un editor totalmente online (en este caso para Python), donde el usuario es capaz de ejecutar su código en el servidor pero visualizar los resultados en una interfaz web.

Figura 3: Checkio, plataforma donde en forma de juego se plantean retos de programación.

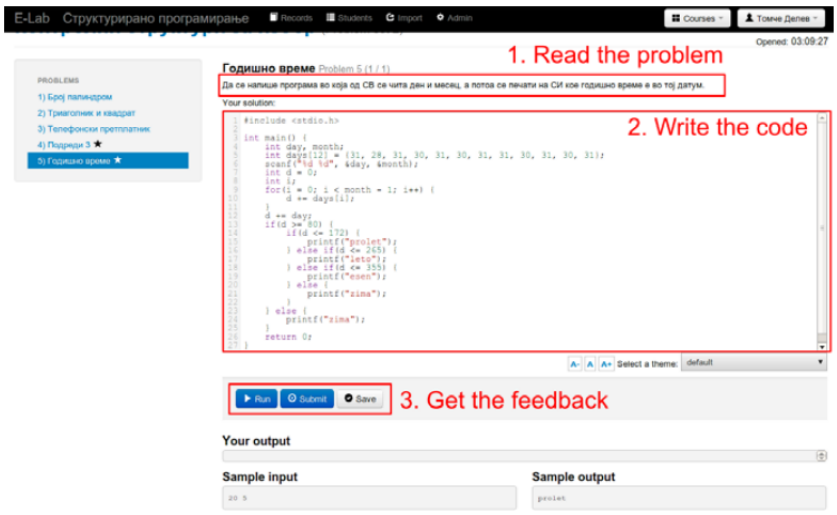


Figura 4: E-Lab system es otro entorno web para la validación automática de problemas de programación Delev and Gjorgjevikj [19].

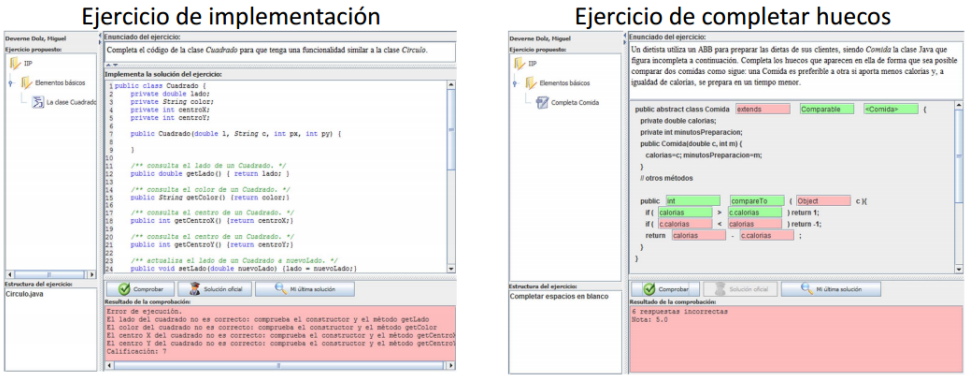


Figura 5: CAP: corrector automático de programas para la evaluación continua y el aprendizaje autónomo descrito en Sapena et al. [38].

3

ANÁLISIS DE LA SOLUCIÓN

Para entender el funcionamiento completo de la herramienta es necesario realizar una descripción completa de los requisitos software que definan el comportamiento de la misma, así como los casos de uso. Un requisito software describe el comportamiento de un sistema. Los requisitos se pueden dividir en dos tipos, el primero son los requisitos funcionales que define el comportamiento interno del software, cuyas interacciones se explican mediante los casos de uso. El segundo tipo, son los requisitos no funcionales o complementarios. A continuación se mostrarán los requisitos software de la aplicación.

Una parte muy importante de la descripción del software es que no haya ambigüedades en cuanto a la terminología utiliza, por ello se van a explicar los principales términos utilizados para la descripción del sistema que se está desarrollando:

- Usuario - Es cualquier persona que utilice la aplicación.
- Profesor - Es la persona encargada de definir y crear el contenido de las prácticas para los alumnos. Se le considera un usuario del sistema.
- Alumno - Es la persona encargada de resolver los ejercicios planteados por el profesor, utilizando el sistema desarrollado. Se le considera un usuario con restricciones
- Administrador - Es el usuario con más privilegios del sistema. Este puede realizar cualquier cambio que desee en el sistema.
- Grupo - Es un conjunto o equipo de personas que el profesor organiza en una escuela, y es constituido por alumnos. Un alumno puede pertenecer a más de un grupo, por ejemplo puede estar en un grupo de prácticas y en un grupo de teoría, cuyos constituyentes no sean los mismos.
- Práctica - Es un conjunto de contenidos en forma de texto constituido por una serie bloques.
- Bloque - Es una agrupación de contenido visual que se puede detallar en tres tipos:
  - Bloque de código - El conjunto formado por elementos (texto, botones, etc.) que concretan la parte de la práctica perteneciente a un programa.
  - Bloque de texto - Se refiere al correspondiente a las respuestas del alumno.

    - Bloque *markdown* - Se corresponde con un texto que utiliza una sintaxis especial definida por un lenguaje de marcado ligero llamado "markdown" y que es famoso por su utilización para describir las páginas de Wikipedia. El objetivo de este tipo de bloque es proveer a la aplicación de un método de introducción de texto con formato, sin necesidad de recurrir a otro tipo de marcados como eXtensible Markup Language (XML) o HTML.

- Programa - Conjunto unitario de instrucciones que permite a un ordenador realizar funciones diversas.
- Compilador - Programa que convierte el lenguaje informático empleado por el usuario en lenguaje propio del computador.

El elemento principal de la aplicación es la práctica. A continuación a modo de algoritmo se muestra el ciclo de vida de una práctica:

1. Elaboración de la práctica por parte del profesor.
2. Introducción de la práctica en el sistema de bases de datos.
3. Comprobación del correcto funcionamiento de la práctica. Si el comportamiento no es el deseado, volver al paso 1.
4. Publicación de la práctica.
5. Resolución de la práctica por parte del alumno.
6. Valoración del ejercicio por parte del profesor.

3.1 ESPECIFICACIÓN DE REQUISITOS SOFTWARE

3.1.1 *Requisitos funcionales*

Como se ha comentado, un requisito funcional define el comportamiento interno del software. A continuación se enumeran los requisitos funcionales del sistema y posteriormente se detallarán mediante los casos de uso.

- Sistema principal:
    - Iniciar sesión: permite al usuario acceder a la aplicación.
    - Cerrar sesión: permite al usuario cerrar sesión en el sistema.
    - Listar grupos: muestra la lista de grupos a los que está inscrito el usuario.
    - Listar prácticas: muestra la lista de prácticas asignadas a un grupo en el que el usuario esta inscrito.

  - Abrir práctica: permite al usuario abrir una práctica de su lista de prácticas.
  - Cerrar práctica: permite al usuario cerrar una práctica.
  - Guardar práctica: permite al usuario guardar una práctica.
  - Guardar prácticas: permite al usuario guardar todas las prácticas activas en la sesión.
  - Imprimir práctica: permite al usuario imprimir una práctica.
  - Iniciar pantalla completa: permite al usuario ejecutar el programa en pantalla completa.

- Sistema administración:
  - Listar usuarios: permite al usuario listar todo los usuarios del sistema.
  - Listar grupos: permite al usuario listar todo los grupos del sistema.
  - Listar grupos del usuario: muestra todos los grupos al que esta inscrito un usuario.
  - Añadir usuario al grupo: permite añadir un nuevo usuario a un grupo.
  - Listar prácticas: permite listar todas las prácticas del sistema.
  - Listar prácticas del grupo: permite al usuario listar todas las prácticas de un grupo.
  - Añadir práctica a grupo: permite al usuario añadir una nueva práctica a un grupo.
  - Añadir grupo: permite añadir nuevo grupo al sistema.
  - Añadir práctica: permite añadir una nueva práctica al sistema.
  - Añadir usuario a grupo: permite añadir un nuevo usuario a un grupo.

3.1.2 *Requisitos no funcionales*

Los requisitos no funcionales son aquellas características del sistema que no proporcionan nuevas funcionalidades, sino que son restricciones que presenta el sistema para adaptarse a las necesidades del producto.

Los requisitos no funcionales que presenta la aplicación son los que se enumeran a continuación:

- El sistema debe permitir ejecutar programas de forma segura sin comprometer la información que se encuentra en el servidor.

- El sistema debe ser multiplataforma, pudiendo ejecutarse en cualquier sistema operativo en cualquiera de los dos navegadores para los que se ha optimizado (Google Chrome y Mozilla Firefox), aunque es de esperar que funcione con otros navegadores como Safari.
- El sistema debe ser intuitivo para que el usuario necesite muy poco tiempo para adaptarse a la forma de trabajar del mismo.
- El sistema debe ofrecer la posibilidad de manejar un gran numero de usuarios.

3.2 CASOS DE USO

Tal y como se ha aclarado al principio del capítulo, los casos de uso describen las secuencias de interacciones que se desarrollan en un sistema, como también los participantes que colaboran en los mismos.

Los participantes de los casos de uso se denominan actores. En la Figura 6 se pueden ver identificados los actores que participan en el sistema que se está describiendo.

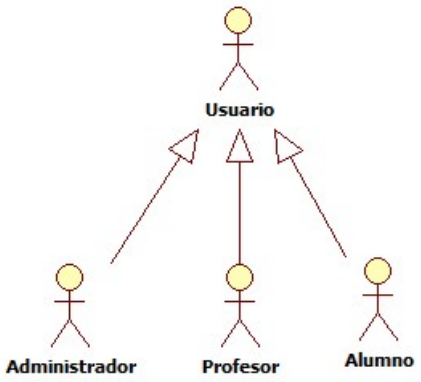


Figura 6: Actores del sistema.

- El usuario tipo *Alumno* podrá visualizar las prácticas que tiene accesibles en los grupos que está inscrito. Además podrá ejecutar las secciones de código escritas por él o por el profesor de manera selectiva pudiendo combinarlas para conseguir completar los ejercicios además de responder preguntas teóricas que se le puedan plantear.
- El usuario tipo *Profesor* será el encargado de escribir el contenido de las prácticas para su posterior introducción en el sistema.
- El usuario tipo *Administrador* será el encargado de gestionar los grupos usuarios y prácticas que se encuentran en el sistema, pudiendo organizarlas de la manera correcta.

3.2.1 *Sistema principal*

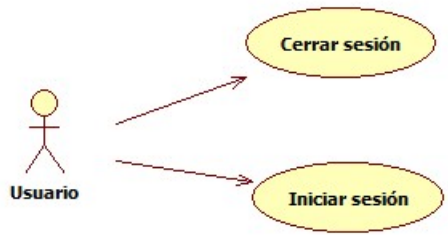


Figura 7: Casos de uso relacionados con la gestión de la sesión

3.2.1.1 *Iniciar sesión*

Caso de uso:	Iniciar sesión
Descripción:	1. El usuario inicia el navegador con la dirección web de la aplicación. 2. Se le muestra la pantalla de inicio de sesión donde se muestra un formulario con dos campos. El primero para introducir el nombre de usuario y el segundo para introducir la contraseña del usuario. Por último el botón "Iniciar Sesión". 3. El usuario introduce su nombre de usuario y su contraseña y pulsa el botón "Iniciar Sesión". 4. El sistema comprueba el par usuario, contraseña. 5. En caso de ser incorrectos se muestra el error "Usuario no existe o contraseña incorrecta". Si el usuario no introdujo ningún dato antes de pulsar el botón de inicio de sesión se mostrará el mensaje "Por favor rellene todos los campos". 6. Si los datos introducidos son correctos se muestra la pantalla principal de la aplicación.
Actores:	Alumno, profesor, administrador
Precondición:	El usuario tenga una cuenta registrada en el sistema para que la acción se pueda llevar a cabo con éxito.

3.2.1.2 *Cerrar sesión*

Caso de uso:	Cerrar sesión
Descripción:	1. El usuario pulsa el botón Salir. 2. El sistema toma el control y libera todos los recursos que el usuario ha generado durante la sesión. 3. Cuando el sistema ha concluido la liberación de los recursos se muestra la pantalla de inicio de sesión, con esto acabando el cierre de sesión en la aplicación.
Actores:	Alumno, profesor, administrador
Precondición:	El usuario haya iniciado sesión anteriormente.

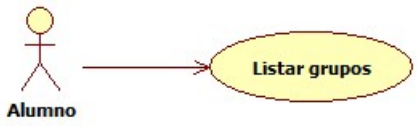


Figura 8: Caso de uso: Listar grupos

3.2.1.3 *Listar grupos*

Caso de uso:	Listar grupos
Descripción:	1. El caso de uso comienza después del inicio de sesión, siendo éste el desencadenante. 2. El sistema recuperará de la base de datos los grupos a los que pertenece el usuario. 3. Se mostrará en el panel izquierdo una lista con los nombres de los grupos a los que pertenece el usuario.
Actores:	Alumno, administrador, profesor.
Precondición:	1. Haber iniciado sesión. 2. Pertenecer a algún grupo.

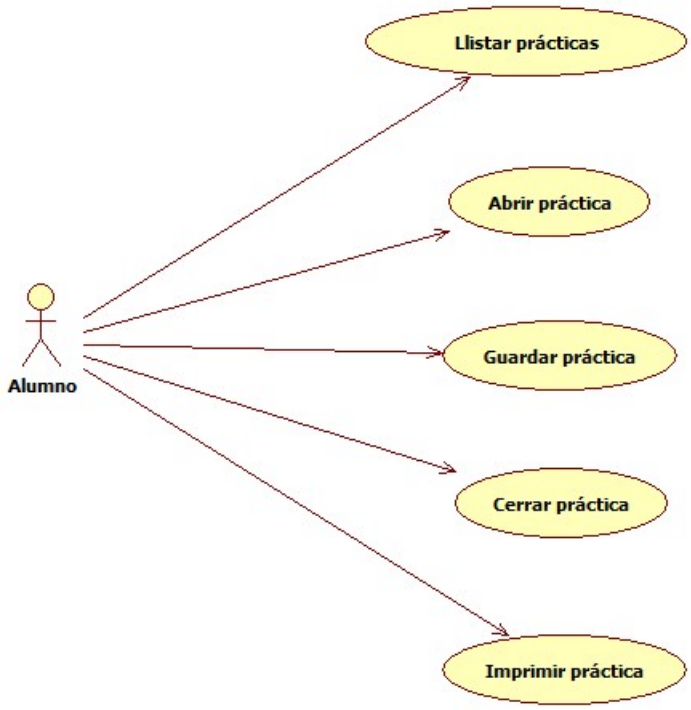


Tabla 4: Casos de uso relacionados con la gestión de las prácticas

3.2.1.4 *Listar prácticas*

Caso de uso:	Listar prácticas
Descripción:	1. El caso de uso se inicia cuando el usuario ha realizado un clic sobre el nombre de un grupo de la lista de grupos. 3. Se listan las prácticas para el grupo en la interfaz de usuario.
Actores:	Alumnos
Precondición:	Haber iniciado sesión.

3.2.1.5 *Abrir práctica*

Caso de uso:	Abrir práctica
Descripción:	1. El usuario sitúa el ratón encima de una práctica de la lista de prácticas. Con un clic selecciona la práctica. 2. El usuario realiza doble click sobre la práctica. 3. Se actualiza la interfaz de usuario, creando una nueva pestaña con el nombre de la práctica en el visor de prácticas y mostrando los contenidos de la práctica.
Actores:	Alumno.
Precondición:	1. Haber iniciado sesión. 2. La práctica no esté abierta.

3.2.1.6 *Cerrar práctica*

Caso de uso:	Cerrar práctica
Descripción:	1. El usuario hace clic sobre el icono para cerrar la práctica de las pestañas abiertas en el visor de prácticas. 2. Se elimina la pestaña con el nombre de la práctica y el contenido de la práctica del visor de prácticas.
Actores:	Alumno.
Precondición:	1. Haber iniciado sesión. 2 Tener una práctica abierta.

3.2.1.7 *Guardar práctica*

Caso de uso:	Guardar práctica
Descripción:	1. El usuario pincha con el ratón encima de la opción "Guardar actual" del menú "Archivos" de la interfaz de usuario. 2. Se muestra un mensaje para avisar al usuario de que está guardando la práctica. 3. El sistema toma el control y envía todo el contenido de la práctica al servidor. 4. En el servidor se recibe el contenido y este se guarda en la base de datos.
Actores:	Alumno
Precondición:	1. Haber iniciado sesión. 2. Tener una práctica abierta.

3.2.1.8 *Guardar todas las prácticas*

Caso de uso:	Guardar todas las prácticas
Descripción:	1. El usuario pincha con el ratón encima de la opción "Guardar todo" del menú "Archivos" de la interfaz de usuario. 2. Se muestra un mensaje para avisar al usuario de que están guardando todas las prácticas. 4. En el servidor se recibe el contenido de todas las prácticas abiertas y éste se guarda en la base de datos.
Actores:	Alumno, administrador, profesor.
Precondición:	1. Haber iniciado sesión. 2. Tener alguna práctica abierta.

3.2.1.9 *Imprimir práctica*

Caso de uso:	Imprimir práctica
Descripción:	1. El usuario pincha sobre el botón "Imprimir Actual" del menú "Opciones". 2. Se mostrará en el navegador la ventana de impresión donde el usuario podrá seleccionar dónde guardar la práctica y el formato (dependiente del navegador que se esté utilizando).
Actores:	Alumno.
Precondición:	1. Haber iniciado sesión. 2. Tener una práctica abierta.

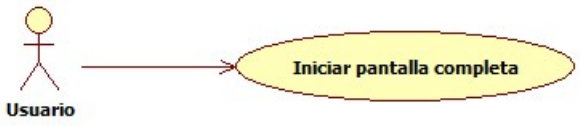


Figura 9: Caso de uso iniciar pantalla completa.

3.2.1.10 *Iniciar pantalla completa*

Caso de uso:	Iniciar pantalla completa
Descripción:	1. El usuario pulsa botón "P. Completa" de menú "Opciones". 2. Se muestra la aplicación sin las opciones del navegador en pantalla completa.

Actores:	Alumno
Precondición:	1. Haber iniciado sesión.

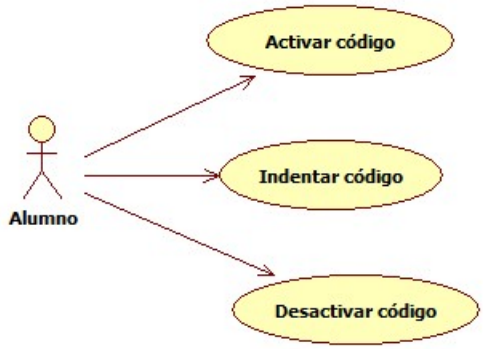


Figura 10: Casos de uso relacionados con la gestión del código escrito por el alumno.

3.2.1.11 *Activar código*

Caso de uso:	Activar código
Descripción:	1. El caso de uso comienza cuando el usuario pulsa sobre el botón "Código desactivado" de la práctica. 2. El sistema toma el control y cambia la opción del usuario para ese bloque de código, marcando la versión del alumno para la etapa de compilación. 3. El botón para a mostrar el texto "Código activado".
Actores:	Alumno
Precondición:	1. Tener la práctica abierta. 2. La práctica contenga algún bloque con código editable que presenta la opción de activar el código. 3. El código debe estar desactivado previamente.
Comentarios:	El efecto de este caso de uso sólo se puede ver cuando el alumno ejecuta el programa que ha desarrollado en una práctica.

3.2.1.12 *Desactivar código*

Caso de uso:	Desactivar código
Descripción:	1. El caso de uso comienza cuando el usuario pulsa sobre el botón "Código activado" de la práctica. 2. El sistema toma el control y cambia la opción del usuario para ese bloque de código, marcando la versión del profesor para la etapa de compilación. 3. El botón pasa a mostrar el texto "Código desactivado".
Actores:	Alumno
Precondición:	Para que se desactive el código previamente tiene que haber estado activado.

3.2.1.13 *Indentar código*

Caso de uso:	Indentar código
Descripción:	1. El usuario pulsa con el ratón el botón "Reindentar" de un bloque de código. 2. Se formatea el texto escrito en el bloque de código para que quede correctamente indentando.
Actores:	Alumno
Precondición:	1. Haber iniciado sesión. 2. Tener una práctica abierta con un bloque de código.

3.2.2 *Sistema administración*

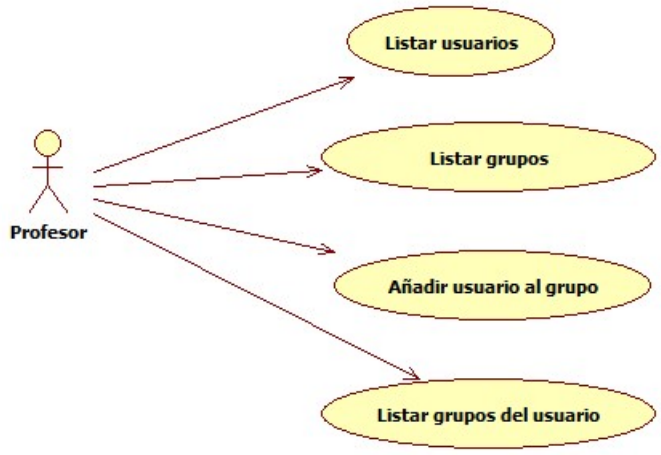


Figura 11: Casos de uso relacionados con la gestión de usuarios en modo administración

3.2.2.1 *Listar usuarios*

Caso de uso:	Listar usuarios
Descripción:	1. El usuario pulsa con el ratón el botón "Actualizar" de la lista de usuarios. 2. Se actualiza la interfaz de usuario mostrando el nombre y el apellido de todos los usuarios del sistema.
Actores:	Administrador
Precondición:	1. Haber iniciado sesión. 2. Ser usuario administrador.

3.2.2.2 *Listar grupos*

Caso de uso:	Listar grupos
Descripción:	1. El usuario pulsa con el ratón el botón "Actualizar" de la lista de grupos. 2. Se actualiza la interfaz de usuario mostrando en la lista de grupos el nombre de todos los grupos del sistema.
Actores:	Administrador
Precondición:	1. Haber iniciado sesión. 2. Ser usuario administrador.

3.2.2.3 *Añadir usuario a grupo*

Caso de uso:	Añadir usuario a grupo
Descripción:	1. El usuario selecciona un nombre de usuario de la lista de usuarios. 2. El usuario selecciona el nombre de un grupo de la lista de grupos. 3. El usuario selecciona con el ratón el botón "Añadir" de la lista de grupos de un usuario. 4. El sistema toma el control enviando los datos al servidor para guardarlos en el sistema. 5. Cuando la operación se haya realizado el sistema actualizará la lista de usuarios y además mostrará un mensaje para avisar de que la operación se ha realizado correctamente.
Actores:	Administrador
Precondición:	1. Haber iniciado sesión. 2. Ser usuario administrador. 3. El usuario no pertenezca al grupo que se ha seleccionado.

3.2.2.4 *Listar prácticas del grupo*

Caso de uso:	Listar prácticas del grupo
Descripción:	1. El usuario selecciona un grupo de la lista de grupos. Marcándose el grupo de otro color. 2. El usuario pulsa con el ratón el botón "Actualizar" de la lista de prácticas. 3. Se actualiza la interfaz de usuario mostrando el nombre de las prácticas que pertenecen al grupo seleccionado.
Actores:	Administrador
Precondición:	1. Haber iniciado sesión. 2. Ser usuario administrador.

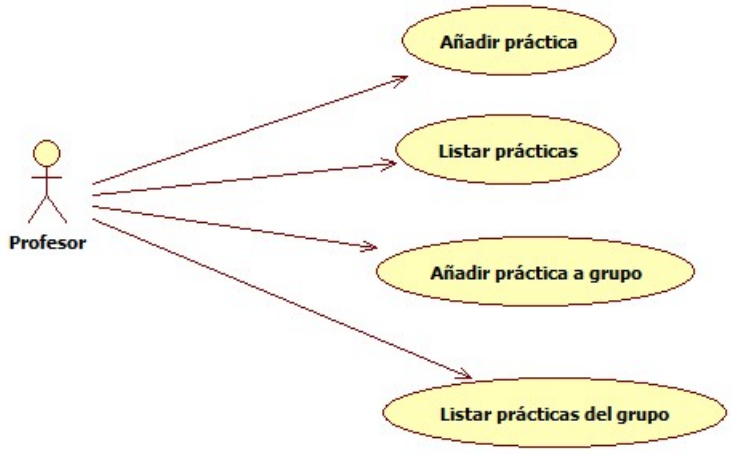


Figura 12: Casos de uso relacionados con la gestión de prácticas en modo administración.

3.2.2.5 *Añadir práctica*

Caso de uso:	Añadir práctica
Descripción:	El caso de uso empieza cuando el profesor ha acabado de crear el contenido de la práctica y ejecuta el *script* para introducir la práctica en el sistema. El *script* se encarga de insertar el contenido dentro del sistema ofreciendo información sobre las operaciones realizadas hasta que el resultado acabe o bien con fallo, en cuyo caso no se habrá insertado nada en el sistema, o bien con éxito insertándose la práctica en el sistema.
Actores:	Profesor
Precondición:	1. Ser profesor.

3.2.2.6 *Listar prácticas*

Caso de uso:	Listar prácticas
Descripción:	El caso de uso empieza cuando el usuario pulsa el botón "Actualizar" del listado de prácticas. Una vez realizada la acción, el sistema se encargará de actualizar la lista de prácticas.
Actores:	Administrador
Precondición:	1. Haber iniciado sesión. 2. Ser usuario administrador.

3.2.2.7 *Añadir práctica a grupo*

Caso de uso:	Añadir práctica a grupo
Descripción:	1. El usuario selecciona el nombre de un grupo de la lista de grupos del menú "Practicas - Grupos". 2. El usuario selecciona el nombre de una práctica de la lista de prácticas de menú "Practicas - Grupos". 3. El usuario selecciona con el ratón el botón "Añadir" de la lista de prácticas del grupo. 4. El sistema toma el control enviando los datos al servidor para guardarlos en el sistema. 5. Cuando la operación se haya realizado el sistema actualizará la lista de prácticas de un grupo y además mostrará un mensaje para avisar de que la operación se ha realizado correctamente.
Actores:	Administrador
Precondición:	1. Haber iniciado sesión. 2. Ser usuario administrador.

3.2.2.8 *Listar grupos del usuario*

Caso de uso:	Listar grupos del usuario
Descripción:	1. El usuario selecciona el nombre de un usuario de la lista de usuarios, marcándose el nombre con otro color. 2. El usuario pulsa con el ratón el botón "Actualizar" de la lista de grupos de usuario. 3. El sistema toma el control y manda un evento al servidor para obtener la lista de grupos del usuario. 4. Cuando se ha recuperado la lista de grupos se envía el contenido al cliente y se actualiza la interfaz de usuario mostrando el nombre de los grupos a los que pertenece el usuario seleccionado.

Actores:	Administrador
Precondición:	1. Haber iniciado sesión. 2. Ser usuario administrador.

(a) Caso de uso añadir usuario. (b) Caso de uso añadir grupo.

Figura 13: Casos de uso relacionados con el usuario de tipo administrador.

3.2.2.9 *Añadir usuario*

Caso de uso:	Añadir usuario
Descripción:	El caso de uso empieza una vez el administrador ejecute el *script* para insertar los usuarios descritos en un fichero con formato JavaScript Object Notation (JSON). El sistema se encargará de insertar los usuarios en el sistema.
Actores:	Administrador
Precondición:	1. Haber iniciado sesión. 2. Ser usuario administrador. 3. Tener un usuario y un grupo seleccionados. 4. No exista un usuario con el mismo identificador en el sistema.

3.2.2.10 *Añadir grupo*

Caso de uso:	Añadir grupo
Descripción:	El caso de uso empieza una vez el administrador ejecute el *script* para insertar los grupos descritos en un fichero con formato JSON. El sistema se encargará de insertar los grupos en el sistema.
Actores:	Administrador
Precondición:	1. Haber iniciado sesión. 2. Ser usuario administrador. 3. Tener un usuario y un grupo seleccionados. 4. No exista un grupo con el mismo identificador en el sistema.

4

DISEÑO DE LA SOLUCIÓN

Una vez especificado el comportamiento de la aplicación, es necesario entrar en detalles de implementación y de las tecnologías que se utilizarán para llevar a cabo el desarrollo de la plataforma. Esta fase de diseño es muy importante, puesto que es la parte creativa en la que el diseñador define cómo se realizará la aplicación en términos de implementación. Hay que tener en cuenta que diseñar una aplicación e implementarla son términos totalmente diferentes, por ello en el diseño de la aplicación se explica qué se debe implementar, pero no cómo se debe desarrollar. Para el diseño de una aplicación, una herramienta muy común es el lenguaje Unified Modeling Language (UML). En este capítulo se va explicar el diseño de la aplicación en términos de los diferentes modelos. En primer lugar se explicará el modelo de aplicación, es decir el tipo de aplicación al que se aspira, siendo ésta una aplicación cliente-servidor. En segundo lugar, se detallará el modelo de datos y conceptual, donde se describirá el tipo de bases de datos utilizado y el modelo de datos que se ha implementado. En tercer lugar, se detallarán los aspectos relacionados con la interfaz de usuario. En la siguiente parte se verá el modelo de compilación y de ejecución, ya que hay que tener en cuenta que es una aplicación que permite la ejecución de programas creados por los usuarios. Y, en último lugar, se comentarán los pasos a seguir para definir una práctica.

4.1 MODELO CLIENTE Y SERVIDOR

El presente proyecto está basado en una arquitectura cliente-servidor. La estructura cliente-servidor, según se define en la Wikipedia: "Es un modelo de aplicación distribuida en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Un cliente realiza peticiones a otro programa, el servidor, quien le da respuesta. Esta idea también se puede aplicar a programas que se ejecutan sobre una sola computadora, aunque es más ventajosa en un sistema operativo multiusuario distribuido a través de una red de computadoras.". En este caso, parte de la aplicación se ejecuta en servidores web mientras que la otra parte se ejecuta en el cliente mediante el navegador web (HTML y Javascript). Este sistema distribuido, entre otras cosas, permite a los usuarios finales acceder a la información de forma transparente independientemente de la plataforma. El lado servidor se ha imple-

mentado con la la tecnología *nodejs*. Esta tecnología se basa en una sistema asíncrono dirigido por eventos y, por tanto, no bloqueante.

Al utilizar una arquitectura cliente-servidor se necesita un proceso central que atienda las peticiones de los usuarios y que, posteriormente, construya una respuesta para la petición realizada. El cliente puede ser cualquier navegador web actualizado que soporte conexiones *websocket*. Aunque, en principio, la implementación del lado cliente debe funcionar en la mayoría de navegadores actuales, se ha optimizado la compatibilidad para dos navegadores en concreto (Google Chrome y Mozilla Firefox) en lo que se refiere a vistas y estilos.

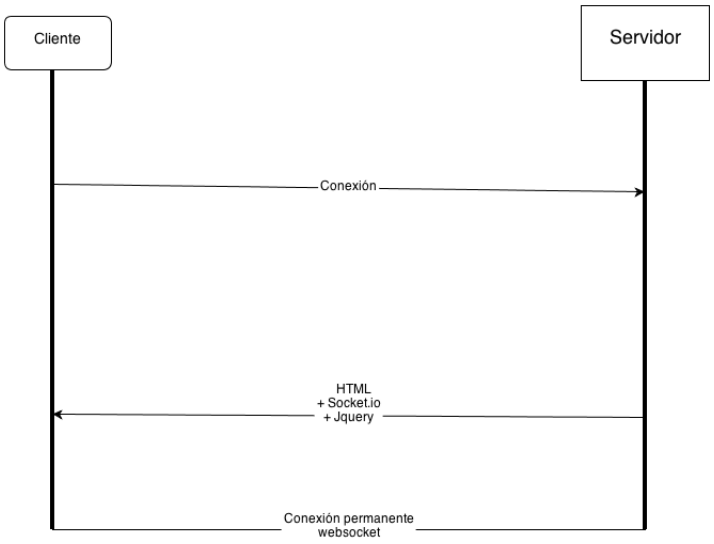


Figura 14: Cliente-Servidor

A continuación se detallan los elementos principales que se encuentran en el lado servidor de la aplicación.

- Un objeto llamado *server*. El proceso implementa un servidor basado en *websocket* para atender peticiones en tiempo real desde los clientes. En la Figura 14 se puede ver cómo se establece la conexión *websocket* para la comunicación en tiempo real.
- El objeto *socketio* está encargado de capturar los eventos que se mandan por *websocket* e informar al *server* de ello. De este modo, el *server* podrá realizar las operaciones necesarias para devolver una respuesta.
- El objeto *workspace:* es donde se delegan todas las tareas que realiza un usuario. Lee operaciones del modulo de generación de código y del módulo de ejecución.
- El objeto *bd* que se encarga de proporcionar acceso para las principales operaciones sobre los elementos de la base de datos.
- El objeto *log* se encarga de escribir, bien por consola o bien en un fichero, todas las tareas y eventos que ocurren en el sistema

para poder analizar el correcto funcionamiento del sistema y poder detectar fácilmente posibles fallos si se ha producido un error en el sistema.

- El objeto *server* se encarga de iniciar el servidor para atender los clientes en un puerto de la máquina donde se está ejecutando. Además, implementa un servidor Hypertext Transfer Protocol Secure (HTTPS) para que los mensajes que se envían por red estén cifrados y nadie pueda leerlos en texto plano. Se encarga de la autenticación de usuarios y de servir los contenidos HTML.

El objeto *socketio* es el encargado de mantener la conexión *websocket* para la comunicación en tiempo real. Para aquellos que no conocen esta tecnología, *websocket* permite la conexión entre un cliente y un servidor mediante un *socket* Transmission Control Protocol (TCP). Esta conexión es bidireccional y está especialmente diseñada para navegadores y servidores web. En la aplicación, un cliente inicia la interfaz gráfica y se crea una conexión *websocket* entre el cliente y el servidor. Mediante el *socket*, el cliente manda peticiones de datos y posteriormente recibe las respuestas desde el servidor cuando se hayan recuperado de la base de datos. Esta comunicación permite interacción en tiempo real entre el cliente y el servidor de manera que la página web no se tiene que refrescar cada vez que se muestre un nuevo contenido, sino que los contenidos se cargan de forma dinámica a medida que se piden al servidor. El objeto *socketio* implementa todas las peticiones que puede recibir del cliente y los posibles tipos de respuesta a las peticiones.

Para que el diseño de la aplicación sea más modular, se ha creado el objeto *workspace*, donde se delegan todas las tareas relacionadas con un usuario. Este objeto se ha creado con la idea de ser un *workspace* del usuario, donde para cada usuario se delegan las operaciones relacionadas con los eventos que recibe *socketio.*

Otro objeto importante es el objeto *db*. Este objeto implementa los accesos a la base de datos para mantener la persistencia de los datos. Las operaciones que ofrece el objeto de la base de datos son las de crear, leer, actualizar y borrar (lo que se conoce como CRUD, las cuatro operaciones básicas un sistema de persistencia de datos). Todas las peticiones de datos son asíncronas, con lo cual una petición a la base de datos no requiere esperar para obtener los datos sino que se ejecuta la operación solicitada y, cuando los datos se hayan recuperado, se podrán enviar al cliente. Una vez se ha realizado la consulta, bien con éxito o bien devolviendo un error, se ejecuta una función para devolver el resultado y actualizar la interfaz de usuario o rellenar los objetos que pedían esos datos. Esta función se denomina *callback*, y es como funcionan todas las operaciones asíncronas.

Por último está el objeto de *log*. Este objeto, más que una utilidad funcional, tiene el objetivo de mantener, bien en consola de Javascript

o bien en un fichero de texto, la información de los eventos y operaciones que van realizándose en el servidor. El objetivo es, por un lado, determinar el correcto funcionamiento de la aplicación y, por otro, determinar más fácilmente los desencadenantes de los eventuales errores. En la aplicación existe una sola instancia de este objeto y es accesible desde cualquier parte.

El objeto generador de código es el que se encarga de extraer de la base de datos las secciones de código para construir los ficheros de código que se podrán compilar y ejecutar posteriormente. Este módulo hace uso de plantillas que, junto a las opciones que se les proporciona, generan un contenido u otro según las opciones que tiene seleccionadas el usuario.

Hasta el momento se ha hablado de objetos que se implementan en el servidor, pero también hay que analizar los objetos que se crean en el cliente, donde conviven dos objetos básicos. El primero es el objeto *conexión* y el segundo el objeto *interfaz* de usuario. El objeto conexión es el encargado de mantener la conexión *websocket* en el lado cliente y, a su vez, realizar las peticiones relacionadas con los eventos desencadenados por el usuario. El otro objeto, la interfaz de usuario, se encarga de mantener actualizada la interfaz gráfica, de manera que el usuario puede interactuar fácilmente con el sistema. En la Figura 15 se puede ver el diagrama de objetos del cliente.

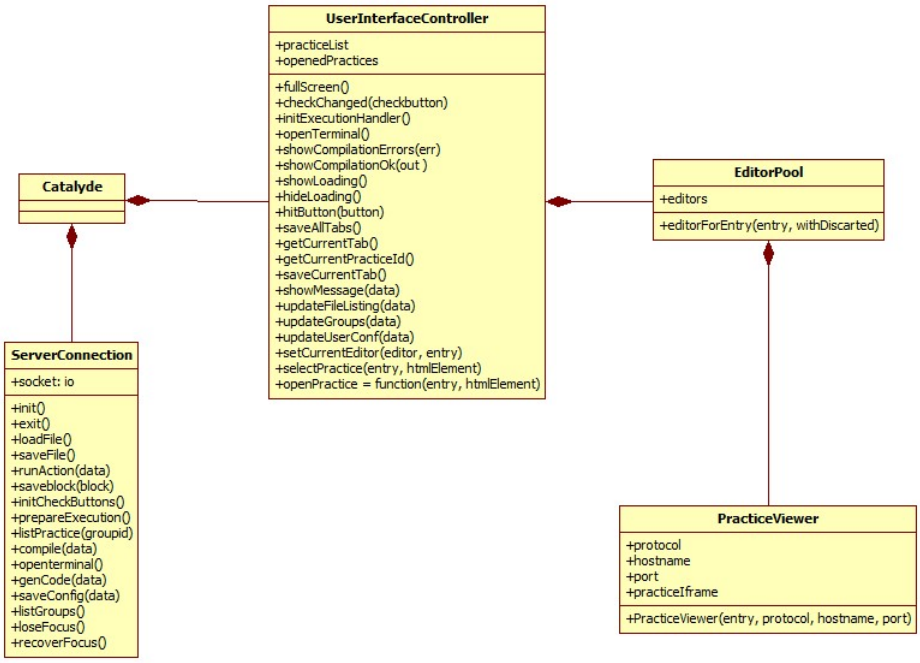


Figura 15: Modelo de objetos del cliente

El objeto encargado de la interfaz de usuario ofrece funcionalidades para mostrar advertencias y mensajes de error, y también es el encargado de mantener las prácticas abiertas mediante el *pool* de prác-

ticas (que es un contenedor de las prácticas). El *pool* de prácticas es donde se encuentran los datos sobre las prácticas. De esta manera, se pueden controlar fácilmente las prácticas que el usuario abre en la aplicación, pudiendo determinar, en cada momento, si una práctica ya está abierta. También puede evitar la reapertura de una misma práctica.

4.2 MODELO DE DATOS Y CONCEPTUAL

La base de datos es la parte de un sistema software que permite la persistencia de los datos. Tener un sistema de bases de datos apropiado para la herramienta que se está desarrollando es esencial para el buen funcionamiento de la aplicación. Para la herramienta que se está describiendo, se ha optado por el uso de una base de datos no relacional. El tipo de base de datos utilizado está basado en el concepto de documento. Los documentos de una base de datos orientada a documentos son similares, en cierto modo, a los registros o filas en bases de datos relacionales, pero son menos rígidas. Se podría ver un documento dentro de la base de datos no relacional como un documento XML donde el contenido es variable y no existe un esquema fijo, sino que el contenido puede cambiar. Las bases de datos no relacionales proporcionan un entorno menos restringido que las bases de datos relacionales, dando más libertad al desarrollador para diseñar el modelo de datos, ya que no presentan un esquema fijo, sino que se puede modificar a medida que la aplicación se va desarrollando. Hay que mencionar que este tipo de base de datos se cataloga como Not Only Structured Query Language (NOSQL), aunque no son directamente comparables con las bases de datos relacionales, es decir que usar un tipo de bases de datos con el otro no es excluyente. Las bases de datos NOSQL ofrecen eficiencia para las aplicaciones en tiempo real donde la cantidad de información almacenada es muy grande y es el caso donde las bases de datos Structured Query Language (SQL) suelen ser más lentas. En contrapartida, las bases de datos SQL son totalmente consistentes, mientras que las bases de datos NOSQL son parcialmente consistentes. Este hecho indica que este tipo de bases de datos no utilizan un lenguaje de consultas basado en el estándar SQL. Los principales motivos de la elección de este tipo de base de datos se basan en:

- Aplicaciones en cambio continuo. Ofrece flexibilidad de cara al desarrollador, ahorrando tiempo de desarrollo y de diseño respecto de un esquema relacional que se tendría que replantear cada vez que haya que añadir componentes nuevos.
- Es un sistema fácilmente escalable mediante la configuración maestro - esclavo. Este tipo de configuración permite realizar las operaciones de inserción y borrado en el servidor maestro,

mientras que las operaciones de consulta se pueden repartir entre los dos servidores, tanto el maestro como el esclavo, distribuyendo la carga. Uno de los problemas que presenta esto es que, si el servidor maestro se apaga de forma inesperada, no se pueden realizar inserciones y borrados en el servidor esclavo. Esto se soluciona mediante resignación automática del maestro, es decir: que al fallar el nodo maestro, el nodo esclavo se pone como maestro. De esa forma, si un servidor falla no se deja de dar servicio a los usuarios. Con esto se pueden soportar un mayor número de usuarios, haciendo más robusta la aplicación.

Como se ha dicho, este tipo de bases de datos no son directamente comparables a las bases de datos SQL (entidad - relación). Con esto se quiere decir que no son una alternativa sino algo que se puede usar conjuntamente con éstas. Algunos inconvenientes que podemos encontrar al usar solamente bases de datos NOSQL son:

- Se enfrentan a un problema de credibilidad importante con muchas empresas por ser algo novedoso. La novedad de NOSQL se traduce a una gran cantidad de desarrolladores y administradores que no conocen la tecnología, lo que hace difícil a las empresas encontrar personas con los conocimientos técnicos apropiados.

- Tienen problemas de compatibilidad. Cada base de datos NOSQL tiene su propia Application Programming Interface (API), las interfaces de consultas son únicas y cada una tiene sus peculiaridades. Esta falta de normas significa que es difícil cambiar de un sistema a otro.

- No cumplen el estándar ACID (*Atomicity, Consistency, Isolation, Durability*). ACID es un conjunto de características o propiedades que garantizan que las transacciones en una base de datos son fiables, entendiéndose por transacción una operación única sobre los datos.

De las bases de datos NOSQL se ha elegido, para la implementación del proyecto, "mongoDB". MongoDB ofrece un intérprete de órdenes para poder realizar el salvado, actualización y recuperación de los datos. De forma complementaria, se utiliza una biblioteca denominada "mongoose" que permite utilizar directamente el intérprete.

Mongoose es, básicamente, una herramienta para el modelado de objetos en entornos asíncronos. Este sistema ofrece una forma de definir objetos, como si se usaran tablas, pero que quedan más cercanos al desarrollador que a un entorno de bases de datos. A continuación se puede ver cómo definir un esquema en la base de datos con Mongoose:

```
var Persona = new Schema({
                nombre: {type: String, required: true},
                apellidos: {type: String, required: true},
                edad: {type: Number, min: 18, required: true },
                dni: {type: String, required: true, unique: true}
});
```

Dado que Mongoose es un módulo desarrollado en Javascript, define objetos de Javascript. A su vez mongoDB es un sistema orientado a documentos y utiliza un lenguaje Binary JavaScript Object Notation (BSON). En este caso, no tenemos una diferenciación entre el modelo de datos y el modelo de objetos, sino que directamente son el mismo. Esto es posible gracias al modulo de Mongoose, que permite definir los objetos de la base de datos como objetos de Javascript y proporciona la serialización de los objetos de la aplicación en la base de datos.

Por tanto en la figura 16 se puede ver el modelo conceptual de objetos que se serializan en la aplicación mediante mongoose.

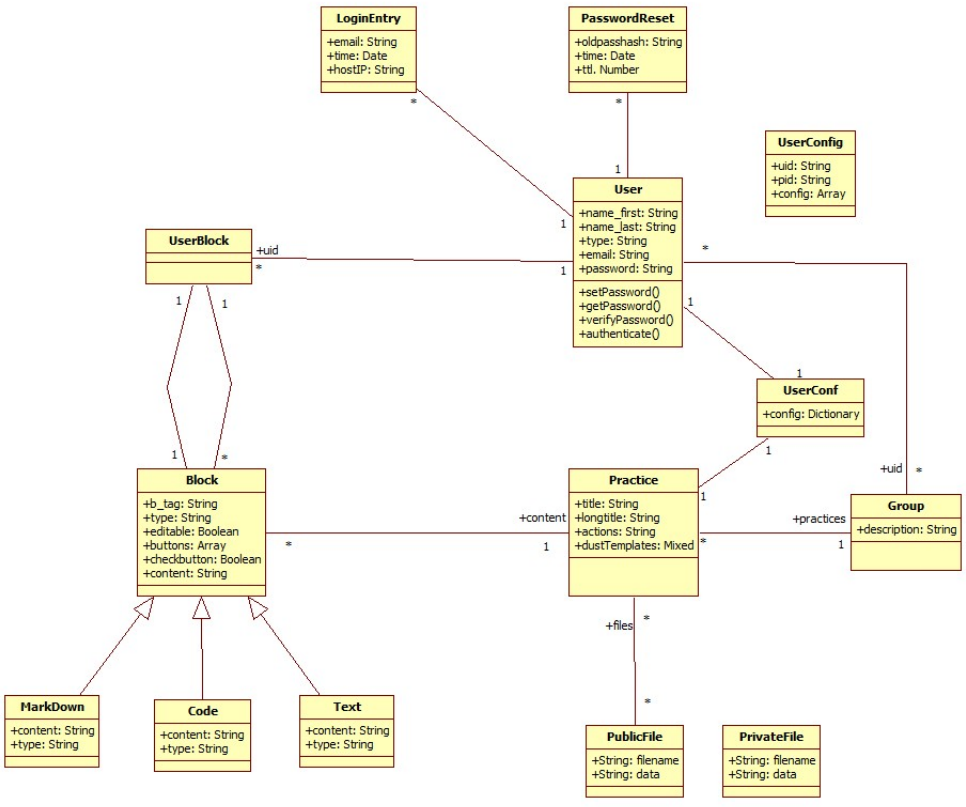


Figura 16: Esquema de objetos/datos

4.3 DISEÑO DE LA INTERFAZ

De cara a los usuarios de una aplicación, sea el tipo que sea, la interfaz de usuario, aquello que el usuario final realmente ve, tiene una gran importancia ya que es la única forma que tiene el usuario de comunicarse con el sistema. En este apartado se explica en qué consiste la interfaz gráfica de usuario de la aplicación.

En primer lugar, hay que recordar que se trata de una aplicación web basada en la aproximación cliente-servidor. Por ello la interfaz de usuario se mostrará en un navegador web. Dado que existen varios navegadores y hay aspectos que cambian de unos a otros, se han utilizado librerías de componentes como jQuery para asegurar la compatibilidad con varios navegadores. Esta elección no supone que la aplicación no se podrá visualizar en el resto de navegadores web existentes en el mercado, sino que algunos detalles de la interfaz puede cambiar dando lugar a un comportamiento no esperado en otros navegadores.

A continuación se mostrará de forma gráfica, a modo de bocetos, el aspecto de la aplicación, explicando la interfaz de libre acceso

4.3.1 *Inicio de Sesión*

La interfaz de libre acceso proporciona al usuario una serie de recursos a los que puede acceder con un usuario y una contraseña. Esto último puede confundir, ya que se ha nombrado a esta parte como de "libre acceso". Con libre acceso se hace referencia a que cualquier usuario que esté en posesión de un nombre identificativo (una dirección de correo) y de una contraseña podrá tener acceso a esta parte de la aplicación, cuyos contenidos dependerán del usuario.

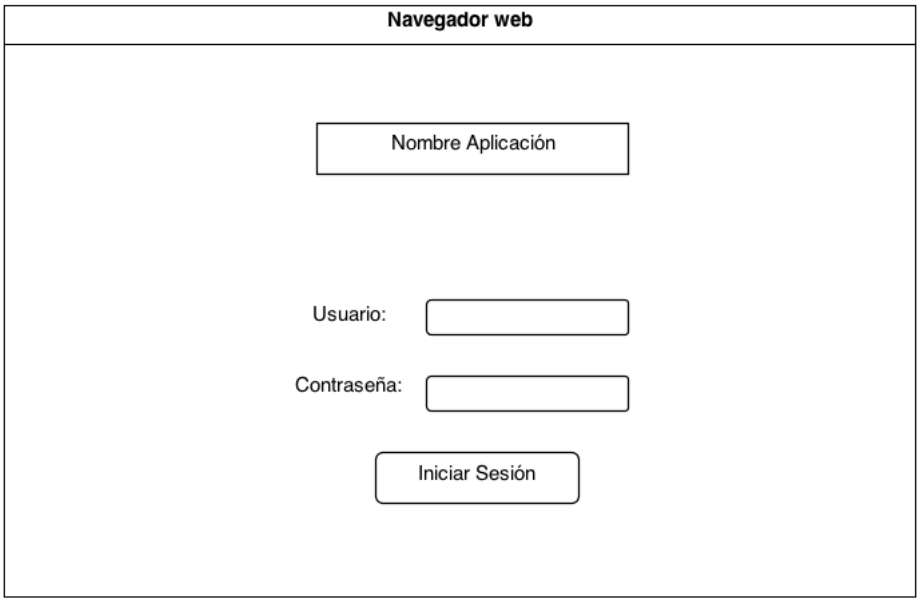


Figura 17: Boceto de inicio de sesión

En primer lugar, como se ha dicho en el párrafo anterior, la aplicación dispone de un menú de inicio de sesión. Si el usuario no está autentificado en el sistema siempre se le redirigirá a esta interfaz. Cuando el usuario introduce sus datos de identificación se pueden dar dos escenarios. El primero es que el usuario haya introducido los datos correctamente, en cuyo caso se realizará el inicio de sesión de forma normal y el usuario tendrá acceso a la interfaz libre. El segundo caso se puede dar cuando el usuario ha introducido los datos de acceso de forma errónea, en cuyo caso se mostrará el error correspondiente.

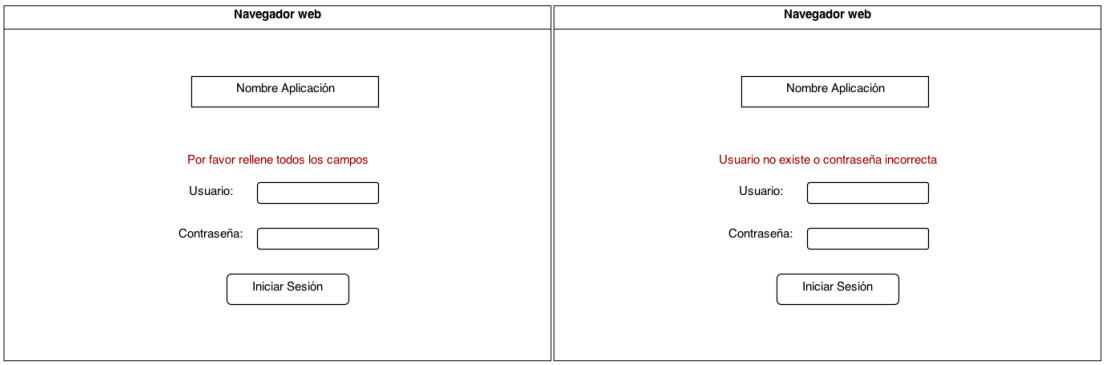


Figura 18: Errores al proporcionar datos incorrectos en el inicio de sesión.

4.3.2 *Vista general de la aplicación*

Si el usuario ha realizado correctamente el inicio de sesión, se iniciará la interfaz de acceso libre. En esta interfaz se pueden apreciar varias zonas. Una barra de menú donde se le proporciona botones

con opciones para interacción con el sistema. Más adelante se explicará con detalle la utilidad de la barra de menú. Otra zona que se puede apreciar es la lista de grupos y prácticas. Estas listas son desplegables, es decir que el usuario puede plegarlas y desplegarlas con un simple clic. Si el clic se realiza sobre un grupo (el primer nivel), éste se despliega, y si se realiza sobre una práctica, ésta se marcará como seleccionada. En caso de realizar un doble clic encima de una práctica, ésta se abrirá en el visor de prácticas. La Figura 19 muestra una vista de la aplicación en la cual no hay ninguna práctica abierta.

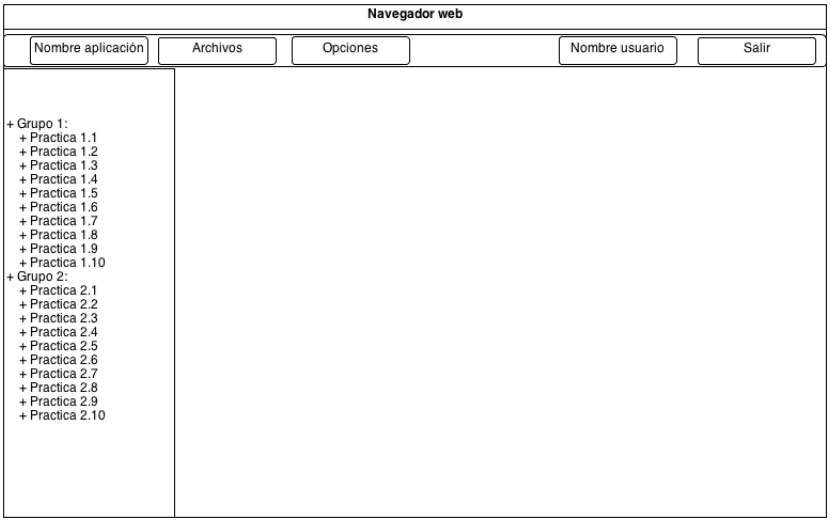


Figura 19: Boceto principal de la aplicación

4.3.3 *Abrir una práctica*

La apertura de una práctica se realizará mediante el doble clic sobre el nombre de la práctica en la lista de la parte derecha de la interfaz. Cuando esto se produce, se envía una petición al sistema y se carga la práctica en el visor de prácticas. La interfaz permite la apertura de varias prácticas al mismo tiempo, de manera que éstas se organizarán en forma de pestañas. Para cambiar la vista de una práctica abierta a otra, bastará realizar un simple clic en la pestaña que tiene el nombre de la práctica. Una vez abierta, una práctica se puede cerrar mediante el icono correspondiente a la pestaña. Una característica que ofrecen las pestañas es que permiten reordenar las prácticas con independencia del orden de apertura. La acción que se debe realizar para poder organizar las pestañas consiste en pinchar y arrastrar las pestaña en la dirección deseada.

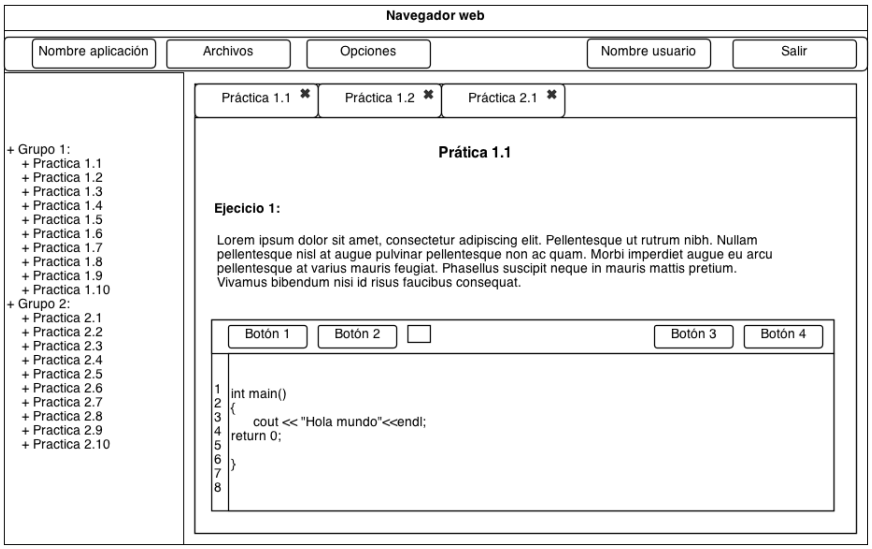


Figura 20: Boceto de la práctica abierta

En la Figura 20 se puede ver un boceto general de la interfaz de la aplicación. Se pueden observar varias prácticas abiertas en las pestañas así como dos grupos en la parte izquierda que tienen desplegadas las prácticas. En la parte superior está la barra de menú con las opciones que ofrece la aplicación. A continuación se explicarán las principales acciones que se pueden realizar en la interfaz de usuario.

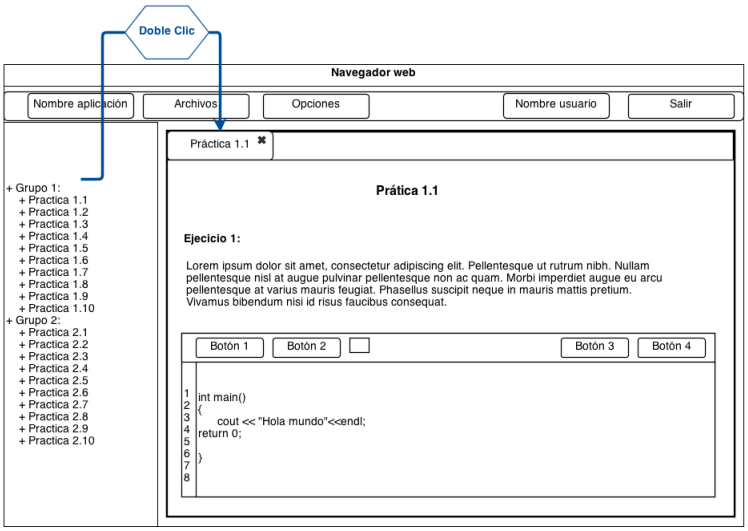


Figura 21: Acción abrir práctica

En la Figura 21 se observa la acción de apertura de una práctica. Para ello, el usuario sitúa el cursor del ratón encima del nombre de la práctica en la lista izquierda y realiza, con un doble clic, la acción de apertura de la práctica. El sistema toma el control y realiza las operaciones necesarias para abrir la práctica en una nueva pestaña en el visor de prácticas.

4.3.4 *Cerrar una práctica*

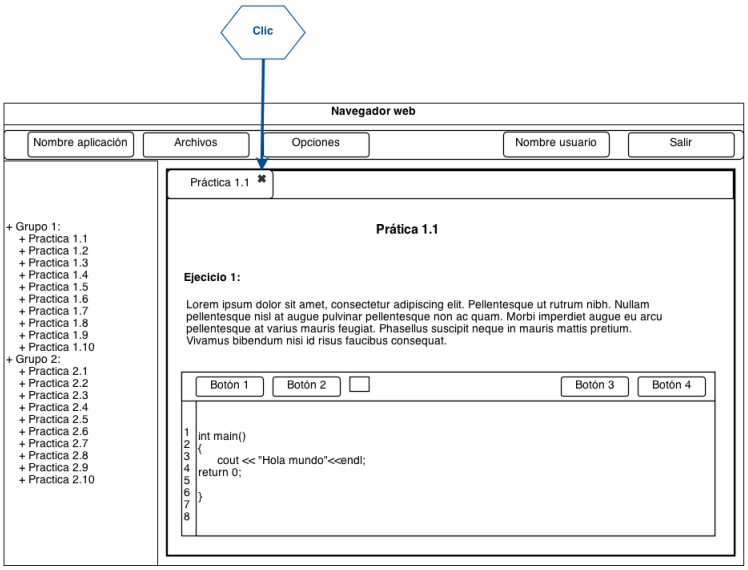


Figura 22: Acción cerrar práctica

En la Figura 22 se muestra la acción que se debe realizar para cerrar una práctica. Para realizar esta acción, tal como se ve en la figura que se está describiendo, el usuario sitúa el cursor del ratón encima del icono de la "x" situado en la parte derecha de la pestaña y, con un simple clic, envía la acción al sistema que tomará el control y realizará las operaciones para que la práctica se cierre correctamente.

4.3.5 *Mover una práctica*

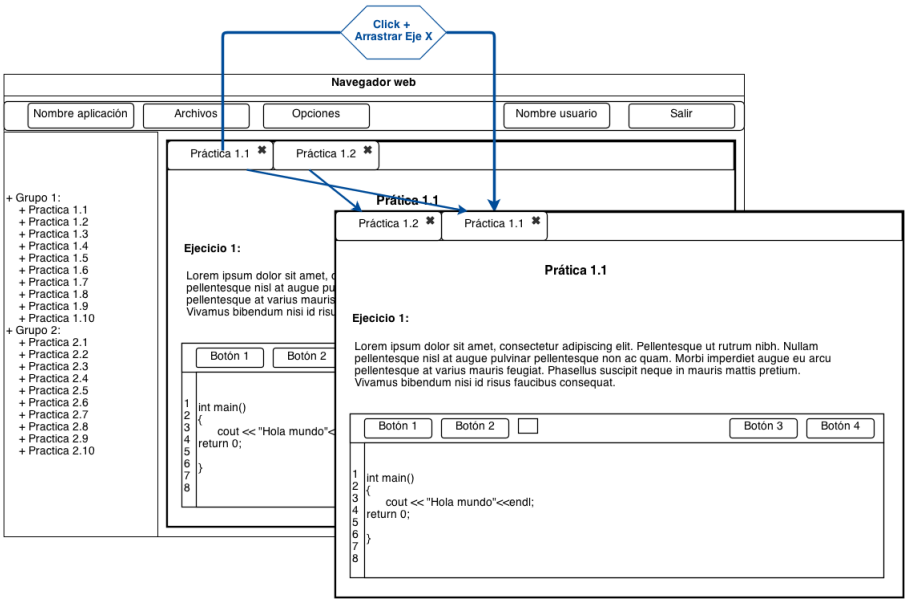


Figura 23: Organizar prácticas abiertas

Otra interacción con el sistema es la organización de las pestañas. El usuario, una vez ha abierto más de una práctica, puede organizar las pestañas en el orden que desee. Para poder realizar esta acción, el usuario debe situar el cursor del ratón encima de la pestaña. A continuación, manteniendo apretado el botón izquierdo del ratón, puede mover el ratón en el eje X de la pantalla. Podrá observar que, a medida que mueva la pestaña, el resto de pestañas se mueven según la posición donde el usuario quiere liberar la pestaña. Cuando el usuario está de acuerdo con la posición de las pestañas, suelta el botón del ratón apretado y la pestaña quedará en esa posición a partir de ese momento.

4.3.6 *Menú de la aplicación*

Los siguientes gráficos ilustran aquellas opciones del menú que contienen algún panel desplegable con más opciones.

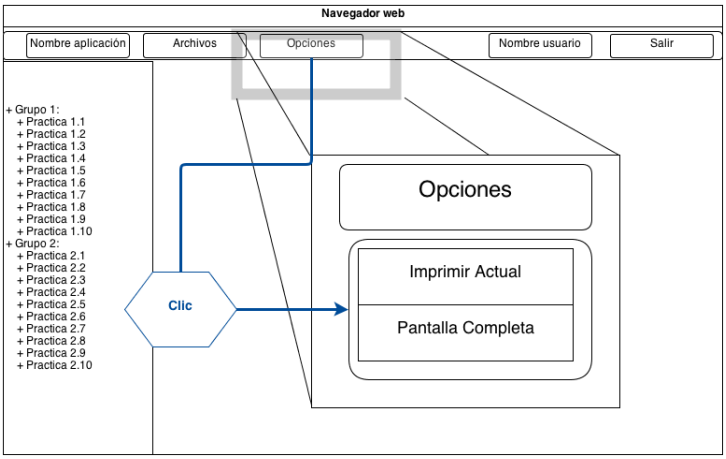


Figura 24: Boceto opciones con detalle

El botón de opciones despliega dos opciones más. Para poder desplegar estas opciones, el usuario sitúa el cursor del ratón de modo que, apretando el botón izquierdo del ratón, aparece el desplegable. A su vez, para seleccionar las opciones, el usuario debería volver a realizar un doble clic con el botón izquierdo del ratón y entonces el sistema ejecutaría la acción. La primera opción ofrece al usuario la posibilidad de imprimir la práctica que tiene abierta en el momento de ejecutar la acción. Si no hubiera ninguna práctica abierta, se mostraría un error indicando que no hay ninguna práctica abierta. La segunda opción, una vez se ejecute la acción de la misma manera que la opción anterior, pondría la aplicación en modo pantalla completa. Es decir, que las opciones del navegador desaparecerían y sólo se vería la interfaz de usuario de la aplicación en la máxima resolución que lo permite la pantalla.

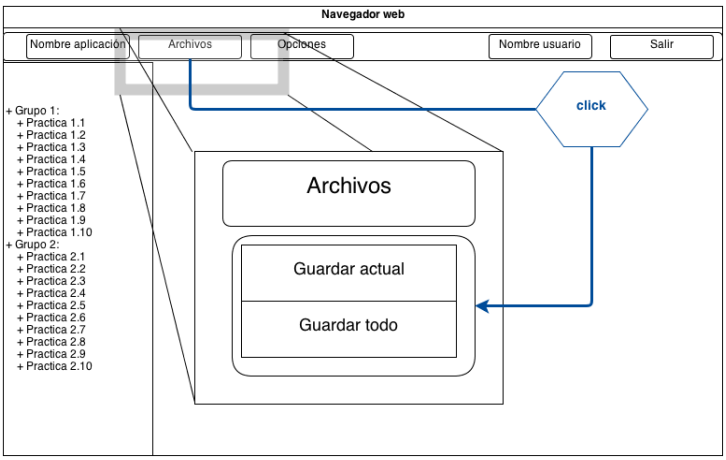


Figura 25: Boceto archivos con detalle

Por último, en la Figura 25, el comportamiento es el mismo que el explicado en la Figura 24. Dado que las acciones se ejecutan de la misma forma, pasaremos a explicar el resultado de ejecutar cada op-

ción. La primera opción "Guardar actual" salva la práctica que se está mostrando en el visor en el momento de su ejecución, mientras que la segunda opción "Guardar todo" guarda en el sistema el contenido de todas las prácticas abiertas.

4.4 MODELO DE EJECUCIÓN Y COMPILACIÓN

La principal meta de la aplicación se centra en proporcionar al usuario la posibilidad de ejecutar programas implementados por él mismo, pero con el matiz de que podrá ejecutar partes del código que aun no ha escrito. Siendo un sistema cliente-servidor se puede plantear la posibilidad de ejecutar el código en el lado cliente o en el lado servidor. A continuación se de explica la opción elegida y justificar esta elección en base a los problemas encontrados para elegir la que mejor se adapta a los requisitos de la aplicación.

- Ejecutar un programa en el lado cliente se traduce en que el programa implementado tendrá que ser soportado por los recursos que disponga el usuario en su máquina, en este caso el cliente será una aplicación web que se ejecutará en el navegador. Esto significa que el lenguaje de programación, si bien no es alguno de los que se pueda ejecutar en el navegador (Javascript, Java), se tendrá que traducir a un lenguaje que el navegador pueda ejecutar. Con este enfoque se consigue que el servidor esté menos solicitado por el cliente, ya que no se produce una sobrecarga por la ejecución de los programas en este, de modo que el servidor solo debe realizar la tarea de atender las peticiones básicas del cliente. Por otro lado la máquina del cliente puede no ser suficientemente potente para poder ejecutar el programa o los recursos sean los justos (o incluso insuficientes). Utilizar esta forma de ejecutar el programa supone que el usuario no realiza una espera de la ejecución ya que es una interacción directa con el programa que se ejecuta en su propia máquina, evitando los retrasos que se producen por la transmisión de la información por la red. Además se pueden evitar los problemas de seguridad en el servidor que pueda generar la ejecución del programa en el mismo, tales como lectura o escritura de ficheros de sistema que contengan las contraseñas de superusuario, y problemas de seguridad por la transmisión de los datos a través de la red.

- Por otro lado, está el enfoque basado en la ejecución del programa en el lado servidor. Dado que a priori se conocen los recursos de los que dispone el servidor se puede determinar la complejidad de los programas que se puedan ejecutar. Al contrario de la ejecución en el cliente no hay dependencia de la máquina de éste. El problema que se podría plantear es que

el servidor, por muy capaz que sea, puede llegar a su punto de máxima carga, afectando a todos los usuarios que estén utilizando la aplicación en ese momento. La respuesta a este problema es la utilización de sistemas escalables. Tanto la base de datos utilizada, como el sistema de eventos, están diseñados específicamente para poder escalarlos de forma sencilla, con lo que el problema de la sobrecarga estaría solucionado.

Planteadas las dos formas de ejecución del programa se explicará la solución adoptada y cómo se llegó a la conclusión de que esta solución es mejor para la aplicación que se esta diseñando.

La primera elección que se hizo fue la de optar por la ejecución de las aplicaciones programadas en el cliente, mediante una herramienta llamada *emscripte*n, que consiste en un conversor del *bytecode* de la máquina virtual de Low Level Virtual Machine (LLVM) a código en lenguaje Javascript. LLVM es una máquina virtual pensada para especificar un lenguaje intermedio para un compilador. El uso de esta tecnología ofrece la posibilidad de evitar problemas de seguridad y abarcar una gran cantidad de lenguajes de programación, más concretamente todos los lenguajes que se puedan compilar con CLang y GNU Compiler Collection (GCC). Dado que ésta parece la solución definitiva, cuando se pasó a la parte de implementación surgió un importante problema que provenía del diseño que tenía *emscripten* para interactuar con el usuario. *Emscripten,* por diseño, dificulta su uso en programas interactivos. Además de esto, la utilización de Javascript en el navegador está limitada a un solo hilo de ejecución, dirigido por eventos, por tanto todos los mecanismos de entrada de datos por parte del usuario producen eventos. Esta forma de interactuar es incompatible con la forma que proporciona *emscripten* para la introducción de datos, que consiste en proporcionar un manejador para que la aplicación solicite de manera activa la introducción de texto, es decir entrada para el usuario. Como este aspecto es muy importante para la implementación de la aplicación se ha descartado la aproximación basada en la ejecución del código en el lado cliente.

Visto que la ejecución en el lado cliente tal como se había planteado no es la solución correcta, se ha optado por realizar la ejecución del programa en el lado servidor. Por ello se añade complejidad para ejecutar el programa ya que se deben tener en cuenta aspectos relacionados con la seguridad, con la ejecución concurrente, pudiendo tener varios usuarios ejecutando al mismo tiempo los programas que ellos mismos escriben. Para ofrecer la interacción deseada con el usuario se ha elegido la utilización en el lado cliente de una terminal (tipo la terminal que pueda ofrecer un sistema operativo) con el matiz de que lo único que podría realizar el usuario en esta terminal es recibir la salida del programa e interactuar con el programa, teniendo en cuenta que el programa se ejecuta en el servidor. Para esta elección

la utilización de la tecnología *websocket* es esencial, ya que mediante ésta, se puede ofrecer al usuario un interacción en tiempo real.
Durante el resto del apartado actual se explicará el modelo de ejecución de la aplicación. El sistema de ejecución de esta propuesta resulta esencial ya que, si no funcionara correctamente, la aplicación no tendría apenas utilidad.
En primer lugar hay que recordar algunos aspectos comentados en los capítulos anteriores. El sistema ejecuta el programa del usuario en el lado servidor y hay una interacción en tiempo real con el usuario.

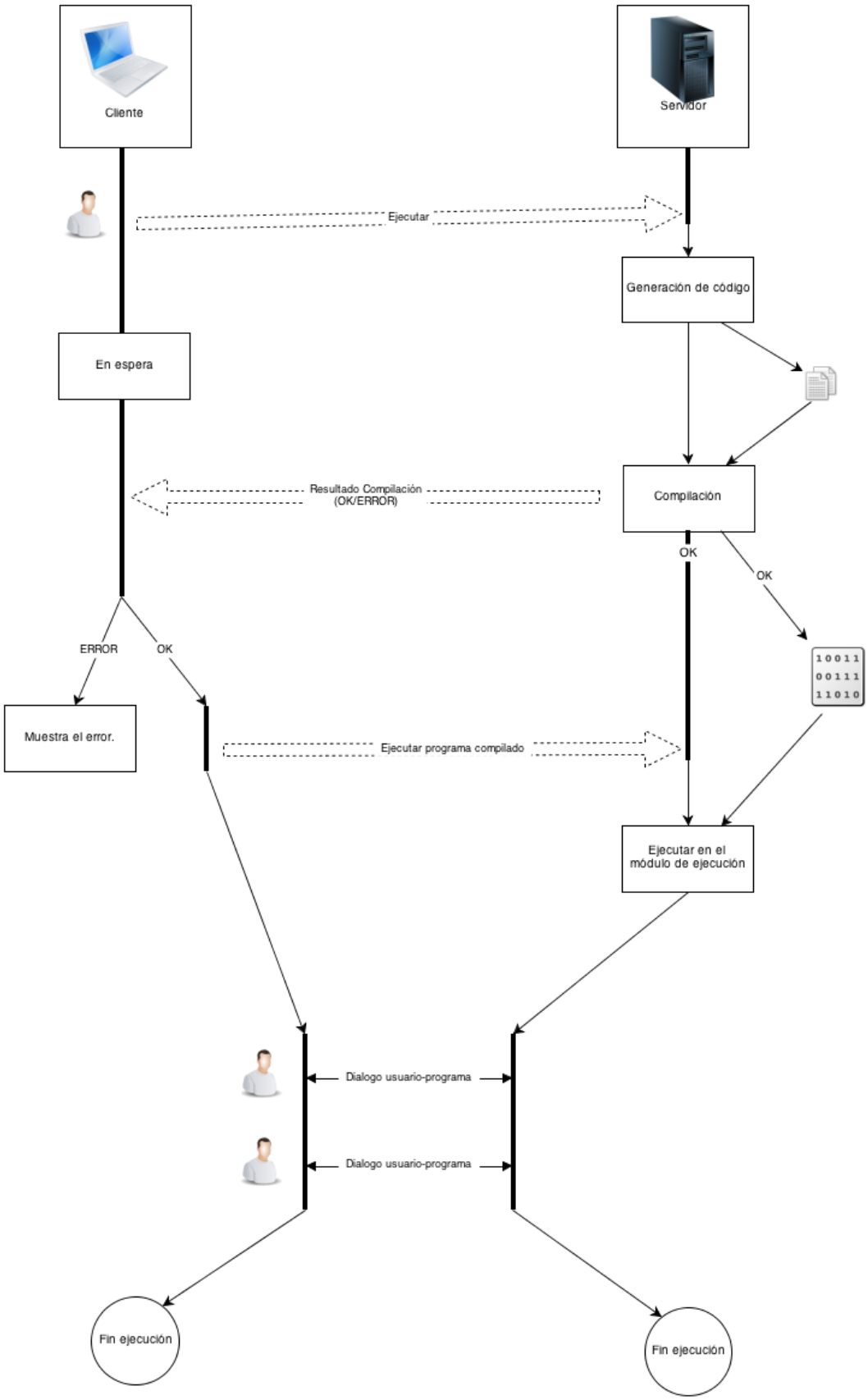


Figura 26: Diagrama del flujo de ejecución de un programa

El diagrama anterior muestra de forma general cómo va el flujo del programa cuando un usuario pulsa el botón de ejecutar de la interfaz de usuario. Hay que mencionar que el proceso, en la práctica, es más complejo que como se especifica en el diagrama. Para disponer posteriormente de un soporte de depuración del programa, se ha optado por utilizar GDB como intermediario para ejecutar el programa. Esto, facilita el control sobre el programa, ya que al ejecutarlo con GDB, en cualquier momento podemos saber el estado del programa, y si se produce algún error inesperado, gracias a GDB se podría notificar. Como se ha dicho, esta elección se ha realizado por la integración que pueda ofrecer el programa en un futuro para la ejecución del código utilizando.

El módulo de ejecución, además de seguir el flujo propuesto en la aplicación, debe seguir una serie de requisitos para poder trabajar de forma eficiente.

- En primer lugar el programa final compilado por la plataforma debe de ejecutarse, con los riesgos de seguridad derivados, y problemas de recursos (memoria, tiempo de ejecución).
- El sistema debe ser capaz de ejecutar programas de diferentes usuarios al mismo tiempo, teniendo una correspondencia entre la ejecución de un programa y la sesión del navegador correspondiente.
- Debe comunicar correctamente la entrada y salida del programa con el usuario (en caso de que se trate de un proceso interactivo) o con el sistema correspondiente (ejecución de casos de prueba).
- El sistema debe proporcionar mecanismos para iniciar y detener la ejecución de programas. Debe ser capaz de detectar si existen procesos que no están asociados a ninguna sesión (por ejemplo, si el usuario cierra el navegador), o interrumpir el proceso.

El siguiente diagrama representa los componentes que forman parte del módulo de ejecución. A continuación explicamos con más detalle su funcionamiento:

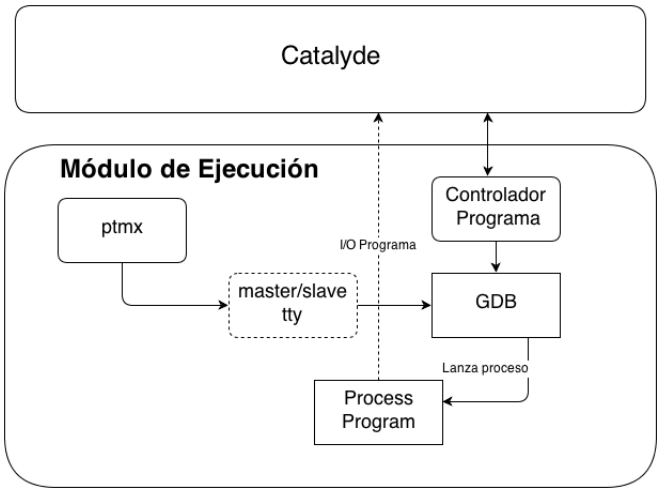


Figura 27: Diagrama del módulo de ejecución

Como ya se ha indicado, los procesos se ejecutan mediante GDB. Esta decisión de diseño ha sido tomada por cuestiones de seguridad y para facilitar la futura inclusión de un modo de depuración. Lanzar mediante GDB el proceso proporciona la opción de controlar la ejecución del programa, siendo capaz de detectar e identificar errores en tiempo de ejecución y de realizar ejecuciones paso a paso en modo depuración para futuras ampliaciones.

En el diagrama del módulo de ejecución, en primer lugar, se observa que el resto de la aplicación (Catalyde) se comunica con el módulo mediante dos canales: el controlador del programa y el mediante la entrada/salida del programa.

CONTROLADOR DEL PROGRAMA El controlador del programa proporciona una series de métodos API para (valga la redundancia) controlar la ejecución del programa:

- Ejecutar programa: este método recibe la ruta de un programa y una configuración, ejecuta una instancia de GDB con el programa cargado y espera a recibir el evento de inicio de ejecución ("run").
- Ejecutar programa en modo *bach*: a diferencia del modo anterior, la ejecución del programa se realiza directamente y no se espera al evento de inicio de ejecución.
- Inicio de ejecución: indica a GDB que inicie la ejecución del programa.
- Matar proceso: evento que finaliza la ejecución del programa manteniendo la ejecución de GDB con el mismo programa, pudiendo retomar la ejecución del programa desde el principio en cualquier momento.

- Terminar proceso: evento que finaliza totalmente la ejecución del programa y del proceso GDB asociado.
- Estado de la ejecución: flag que indica si existe algún proceso en ejecución.
- Señal del programa: devuelve el estado del proceso indicando así si está en ejecución, si ha terminado y el estado de terminación: ['UNKNOWN', 'RUNNING', 'KILLED', 'SUCCESFUL'].
- Emisor de eventos de GDB: Además, el controlador proporciona un emisor de eventos donde se indican los eventos de ejecución del programa: final de ejecución, errores, etc.

COMUNICACIÓN CON EL PROGRAMA Existe un canal de comunicación, que comunica directamente la entrada y salida con el resto de la aplicación. En la práctica se trataría de un socket entre la aplicación Catalyde y el programa. Este socket de lectura y escritura proporciona la opción de añadir texto por la entrada estándar escribiendo sobre el socket y, a su vez, recibe un evento cada vez que el programa escribe algo por la salida estándar o por la salida de error.

El canal de comunicación se realiza mediante Pseudo-Terminales. En algunos sistemas operativos, incluyendo Unix, una pseudo terminal es un par de pseudo-dispositivos maestro/esclavo. El esclavo emula una terminal de texto mientras que el máster se encarga de conectar con la terminal. En otras palabras, se puede ver esto como un canal de comunicación con una terminal. La idea, es crear una pseudo terminal, conectar el pseudo-dispositivo maestro mediante un socket (socket de entrada y salida) con la aplicación Catalyde y, cuando se ejecute el proceso, se conecta el esclavo con la entrada/salida del proceso ejecutado. De esta manera se tiene el canal de comunicación con el proceso en ejecución. Por ejemplo, este canal se conecta al terminal interactivo mostrado en el lado cliente.

Para realizar las pseudo-terminales se utiliza la función de POSIX *open_pt*, obteniendo así los descriptores de fichero y las rutas de los pseudo-dispositivos. Además con nodejs, es posible de utilizar funciones compiladas en C.

4.5 DEFINICIÓN DE UNA PRÁCTICA

En la versión actual, las prácticas se crean sin una interfaz de usuario. En su lugar, el profesor utiliza un editor de texto externo. Para la generación de prácticas se ha implementado un analizador de código que toma todos los datos creados por el profesor y los añade en forma de objetos a la base de datos. Para crear una práctica se debe conocer la sintaxis correcta ya que, en otro caso, el contenido no se

introducirá correctamente en el sistema. En primer lugar se han definido secciones llamadas bloques que pueden ser de código, de texto o enunciado, utilizando el lenguaje de marcado ligero *markdown*. Las secciones se delimitan mediante unas etiquetas.

```
<!-- BEGINBLOCK { "type" : "code", ... } -->
/* aqui va el código de ejemplo */
<!-- ENDBLOCK -->

<!-- BEGINBLOCK { "type" : "text", ... } -->
/* aqui hay contenido en forma de texto*/
<!-- ENDBLOCK -->

<!-- BEGINBLOCK { "type" : "markdown", ... } -->
/* aqui hay contenido en lenguaje markdown*/
<!-- ENDBLOCK -->
```

Figura 28: Ejemplo de los tres tipos de bloques que soporta el sistema.

Como se puede ver en la Figura 28, las secciones se delimitan por etiquetas. El inicio de cada sección contiene la palabra BEGINBLOCK y, entre llaves, atributos en forma de JSON. El final de una sección se marca con la palabra ENDBLOCK. Existe una excepción para esto y es que, si hay texto que no se encuentra entre las etiquetas BEGINBLOCK y ENDBLOCK, se considera que es de tipo markdown, por ello una sección de este tipo no acaba hasta que no empiece un bloque con una etiqueta BEGINBLOCK. Con estos tres tipos de contenido el profesor puede crear la parte estática de una práctica. Ahora bien, los bloques de código son algo especiales ya que van enlazados a otro tipo de plantillas. Estas plantillas definen las reglas de uso del código de un programa. La lógica que contienen las plantillas del código define qué contenido es visible o no por el usuario. En la sección de código 29 se puede ver un ejemplo en el que la parte del alumno se sustituye por el contenido que escribe el alumno y que está guardado en la base de datos, mientras que hay otra parte escrita por el profesor donde está el código al que el alumno no tiene acceso.

```
{?funcion_pedir_entero}
{#load blockid=id.funcion_pedir_entero /}  // Código del alumno
{:else}{~n}
int pedir_entero(char mensaje[])
{
                              //Código de profesor
        int numero;
        fprintf(stdout,"Introduce %s: ",mensaje);
        fscanf(stdin,"%d",&numero);   return numero;
}
{/funcion_pedir_entero}
```

Figura 29: Muestra de un trozo de una plantilla de código.

Aunque se tenga una idea clara de las estructura de una práctica, para facilitar el trabajo del profesor se proporcionan ejemplos de uso de prácticas. Para que se tenga una idea general en la Figura 30 se muestran los elementos principales que intervienen en su creación. En el Anexo 1 se describirá de forma más completa en qué consisten estos elementos.

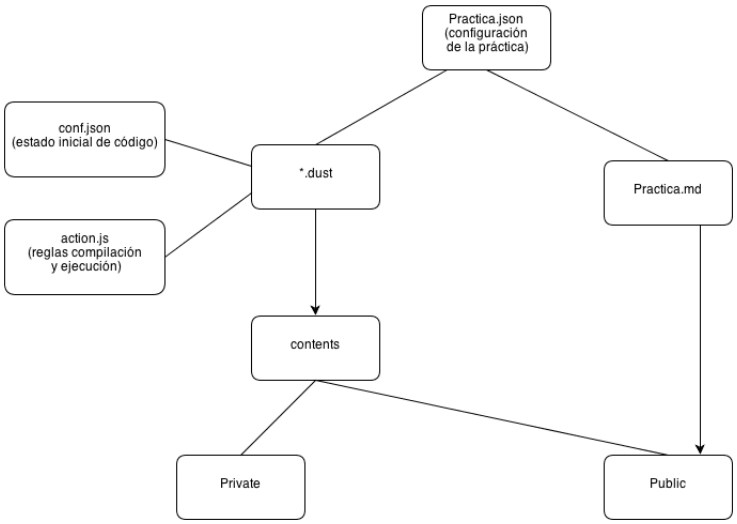


Figura 30: Esquema de los elementos que componen una práctica en relación a la creación de la misma.

5

TECNOLOGÍAS APLICADAS

En este apartado se explicarán las tecnología utilizadas para el desarrollo del proyecto. Existe una semejanza entre la aplicación y un iceberg: en ambos casos la parte no visible es considerablemente más grande. Lo mismo ocurre en esta aplicación donde la parte visible corresponde a la interfaz de usuario, si bien ésta apenas cubre una pequeña parte del total de tecnologías utilizadas y del trabajo realizado (en particular, el usuario no observa directamente el trabajo en el lado servidor), como se ilustra en la siguiente figura. Esta analogía representa el proyecto como un iceberg, donde hay una pequeña parte visible, que es la interfaz final con la que interacciona el usuario y por debajo, existen una gran cantidad de tecnologías usadas que hacen posible la aplicación final.

Se han divido las tecnologías usadas en tres categorías: Tecnologías usadas en el cliente, usadas en el servidor y las herramientas de apoyo para la realización del proyecto.

Figura 31: Tecnologías utilizadas

5.1 TECNOLOGÍAS USADAS EN LA PARTE CLIENTE

Se corresponden con las tecnologías usadas en el navegador web y que se ejecutan en el ordenador del usuario. Normalmente se trata de librerías que trabajan sobre Javascript y por consiguiente sobre HTML.

5.1.1 *Termlib*

Es una librería Javascript que proporciona la posibilidad de mostrar terminales en una interfaz web, siendo una librería libre con una implementación orientada a objetos que permite el manejo de múltiples instancias de terminales. Además permite la personalización de las terminales para adaptarse a las necesidades del entorno. Se ha utilizado para proporcionar al sistema una forma para que el usuario pueda interactuar con el programa que él mismo ha escrito.

5.1.2 *jQquery*

jQuery es una biblioteca de Javascript cuyo principal objetivo es simplificar el manejo de los documentos HTML mediante eventos, animaciones, selección de elementos del DOM, etc. El lema de jQuery es "do less, write more", es decir que intenta acelerar la implementación de entornos web aportando funcionalidades que, sin jQuery, supondrían un mayor esfuerzo. jQuery es utilizado por compañías tan conocidas como Google, Microsoft, o IBM, por lo que no es de extrañar que tenga un gran apoyo por parte de compañías importantes en lo que se refiere a su utilizacion. Se ha empleado en el desarrollo de la lógica de la interfaz de usuario para facilitar el manejo de los elementos HTML, pudiendo de esta manera hacer más dinámico el entorno de usuario de la herramienta.

5.1.3 *jQuery UI*

jQuery UI utiliza la misma filosofía que jQuery, tal como se ha explicado en el apartado anterior. Se ha implementado utilizando como base de la librería jQuery y trata de ofrecer interfaces gráficas para su inserción en el entorno web, facilitando de esta manera el diseño de los entornos web. Proporciona a la aplicación desarrollada elementos gráficos, siendo un ejemplo concreto el visor de prácticas mediante el uso de las pestañas para poder tener abiertas varias prácticas a la vez.

5.1.4 *CodeMirror*

Es un entorno basado en Javascript que posibilita disponer de un editor de código en el navegador. Tiene varios modos para resaltar la sintaxis del código mediante colores para diversos lenguajes de programación e incorpora funcionalidades para la edición tales como deshacer, rehacer, etc. Esto proporciona gran flexibilidad para tener un editor de código en el entorno web. Google utiliza este editor en Google Code para la edición online del código fuente de los repositorios. Su utilización ha sido significativo para la elaboración de los bloques de código que se han utilizado en la prácticas, siendo éstas las secciones donde el alumno puede introducir el programa que él mismo ha escrito.

5.2 TECNOLOGÍAS USADAS EN EL SERVIDOR

En este apartado se especifican las librerías, frameworks y herramientas que se ejecutan en el lado servidor y que se encargan de la mayor parte de la lógica de la aplicación.

5.2.1 *Nodejs*

Node.js es framework (entorno de programación y librería) basado en la máquina virtual V8 de Google (la misma que se utiliza para ejecutar Javascript en el navegador Google Chrome). Esto posibilita realizar programas en Javascript que se ejecutan en el lado servidor. El objetivo de nodejs fue la creación de programas de red altamente escalables para construir aplicaciones dirigidas por eventos con baja latencia y concurrentes. Para ello, además del intérprete de Javascript, Node.js dispone de un pool de threads y de una biblioteca de eventos que se complementan con una enorme cantidad de bibliotecas asíncronas.

A pesar de ser una tecnología muy reciente, ha tenido mucha repercusión y ha sido adoptado incluso por empresas como:

- Microsoft: en Windows Azure se usa Node.js internamente para proveer almacenamiento en la nube para aplicaciones.
- Yahoo: en Yahoo! Manhatan Node.js es el núcleo de la aplicación.
- Palm/OS: todos los servicios para la los dispositivos móviles de WebOS han sido desarrollados en Node.js.
- eBay: en el desarrollo de ql.io, un lenguaje inspirando en SQL y JSON para trabajar con APIs Hypertext Transfer Protocol (HTTP).

5.2.2 *Modulos de Node.js*

Una de las claves del éxito de Node.js es la facilidad que ofrece para poder instalar módulos de diversos tipos mediante un sistema de gestión e instalación denominado Node Package Manager (NPM). Mediante esta herramienta se pueden instalar módulos como: controladores para acceso a base de datos, acceso a librerías del sistema, control de flujo. Todas ellas son software libre y, por tanto, se pueden usar siempre y cuando se cumpla la licencia bajo la cual se esté desarrollado.

5.2.2.1 *Mongoose*

Mongoose es una herramienta para modelado de objetos en entornos asíncronos que sirve de enlace con MongoDB. Gracias a mongoose se pueden definir esquemas de objetos para poder guardarlos en mongo, reflejando restricciones, tipos de objetos y muchas más características. Mongoose se ha utilizado para definir los esquemas de los objetos para poder guardar los datos de forma homogénea en la base de datos.

5.2.2.2 *Jade.js*

Es un motor de plantillas implementado con Javascript para simplificar la sintaxis HTML. Aparte, jade proporciona una serie de estructuras de control como bucles, sentencias if/else, etc. que hace que se pueda generar de forma dinámica documentos HTML. Este módulo se ha utilizado para generar los contenidos del sistema en HTML teniendo plantillas base para ello.

5.2.2.3 *Dust.js-linkedin*

Es una librería escrita en Javascript para generación de plantillas. En un inicio se llama simplemente dust.js, pero posteriormente la empresa LinkedIn se hizo cargo de mantener y mejorar la librería. En el proyecto se ha usado para el ensamblado del fichero de código escrito por el alumno que posteriormente se compilaría y ejecutaría. El módulo es muy parecido al que se ha explicado en el apartado anterior, pero se ha utilizado con un fin diferente. El objetivo del módulo en el sistema ha sido la utilización en el generador de código para creación de los ficheros de código fuente que posteriormente se compilan.

5.2.2.4 *Socket.io*

Es una librería escrita en Javascript para comunicación en red y se ejecuta tanto en el servidor con Node.js como en el cliente. Esta librería crea una capa de abstracción sobre websockets y otros esquemas

de comunicación, dependiendo de la capacidad del navegador. Permite enviar mensajes dirigidos o difundir mensajes a todos los clientes que están "escuchando". Mediante este módulo se ha implementado una parte muy importante del sistema, que es la comunicación en tiempo real mediante websocket entre el cliente y el servidor.

5.2.2.5 *ExpressJS*

ExpressJS es un sistema que ayuda a organizar la parte servidor de una aplicación web basada en nodejs utilizando el patrón Model-View-Controller (MVC). El sistema permite controlar prácticamente todo lo que está relacionado con el acceso al servidor, desde rutas hasta el manejo de las peticiones y vistas. Además permite el uso de lenguajes basados en plantillas como (jade, dust,...) que también se han utilizado para el desarrollo de la aplicación. Express se está usando en aplicaciones como MySpace, LearnBoost, Storify entre otros. En el sistema se ha utilizado como enrutador para la creación de las rutas a las que tienen acceso los usuario cuando se conectan a a interfaz web. Este módulo convierte al sistema en un servidor web, ya que hay que tener en cuenta que Node.js es una librería de eventos y no un servidor web.

5.2.2.6 *Passport*

Passport es un paquete para realizar la autenticación de usuarios y se puede integrar perfectamente en una aplicación creada con ExpressJS. Comprende una serie de estrategias para poder autenticar usuarios de Facebook, Twitter y muchos mas. En el proyecto se ha utilizado passport como sistema de autenticación de usuarios creando un sistema de autenticación local con la ayuda de MongoDB como base de datos.

5.2.2.7 *Async*

Async es un módulo que proporciona una serie de funciones que se utilizan para el control de flujo en entornos asíncronos. Se ha empleado en el proyecto para evitar lo que se llama *código spaghetti* [1], de forma que el código queda más legible.

5.2.2.8 *Bcrypt*

Es una librería que se utiliza para el cifrado de las contraseñas en la base de datos. Sirve para guardar las contraseñas con una función de derivación a partir de las cuales podemos validar la contraseña

1 El código spaghetti es un término peyorativo para los programas de computación que tienen una estructura de control de flujo compleja e incomprensible. Su nombre deriva del hecho que este tipo de código parece asemejarse a un plato de espaguetis, es decir, un montón de hilos intrincados y anudados.

que el usuario introduce durante el inicio de sesión. Se ha utilizado en el sistema para mantener la seguridad de los usuarios guardando las contraseñas de éstos de manera más segura.

5.2.2.9 *Pty*

Es una implementación para Node.js para poder utilizar la librería forkpty. Proporciona una forma de emular un terminal al ejecutar un programa y poder interactuar con la entrada y salida del programa como si estuviera en una terminal.

5.2.3 *MongoDB*

Como se ha mencionado con anterioridad, se trata de una base de datos basada en documentos que no utiliza el estándar SQL. Es fácilmente escalable, de código libre y está programado en C++. Empresas como Foursquare, bitly o SourceForge, entre otras, están utilizando MongoDB. Como se ha explicado, es el sistema de bases de datos utilizado para mantener la persistencia de los objetos en el sistema desarrollado.

5.2.4 *GDB*

GDB es un depurador de programas desarrollado por GNU. Está disponible para la mayoría de los sistemas operativos basados en Unix y se puede utilizar con gran cantidad de lenguajes de programación. Aunque no se haya utilizado con el fin de depurar programas, este depurador se ha utilizado pensando en las ampliaciones futuras y para tener mayor control sobre la terminación de los programas ejecutados.

5.2.5 *Apparmor*

Es un módulo de seguridad del núcleo de linux que sirve para asociar perfiles de ejecución y restricciones sobre los programa que se ejecute en el sistema operativo. Se ha empleado en el proyecto para controlar la ejecución de los programas de los usuarios como un mecanismo para proteger la seguridad del servidor. En particular, este módulo impide que el programa ejecutado por el alumno acceda a rutas del sistema donde se puedan guardar contraseñas de administrador o impedir escribir o borrar ficheros del sistema fuera de la carpeta destinada a la ejecución del código de cada alumno.

5.3 HERRAMIENTAS ADICIONALES

Adicionalmente, para el desarrollo del proyecto y de la presente memoria, se han utilizado varias herramientas y utilidades que enumeramos a continuación.

5.3.1 *SVN*

Subversion es un sistema de control de versiones bajo una licencia de tipo Apache/BSD y se le conoce también como svn por ser el nombre de la herramienta utilizada en la línea de comandos. Una característica importante de Subversion es que todo el repositorio tiene un único número de versión que identifica un estado común de todos los archivos del repositorio en un instante determinado en el que se está trabajando.

5.3.2 *Aptana Studio*

Aptana Studio es un entorno de desarrollo integrado de software libre basado en eclipse y desarrollado por Aptana. Provee soporte para lenguajes como: Php, Python, Ruby, CSS, Ajax, HTML y Adobe AIR. Tiene la posibilidad de incluir complementos para nuevos lenguajes y funcionalidades.

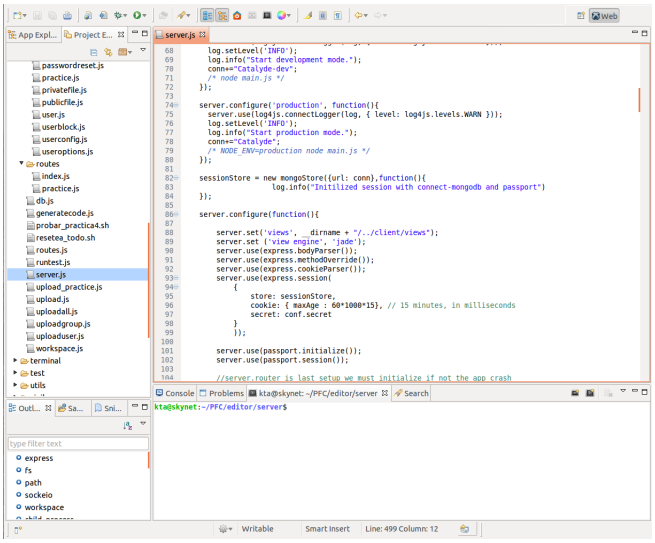


Figura 32: Figura de muestra de Aptana Studio

Este editor se ha utilizado para implementar todo el código de la herramienta.

5.3.3 *StarUML*

Es un proyecto de software libre que se centra en el desarrollo rápido de diagramas UML/Model Driven Architecture (MDA). La herramienta está implementada en java funcionando bajo Windows. Aunque empezó siendo un proyecto comercial, éste se abandonó durante un tiempo y, más tarde, el proyecto se revivió cambiando el lenguaje de programación de delphi a java. La herramienta soporta un conjunto de diagramas que se enumeran a continuación:

- Soporte completo al diseño UML mediante el uso de.
- Diagrama de casos de uso
- Diagrama de clase
- Diagrama de secuencia
- Diagrama de colaboración.
- Diagrama de estados
- Diagrama de actividad.
- Diagrama de componentes
- Diagrama de despliegue.
- Diagrama de composición estructural (UML 2.0)

Los diagramas utilizados para el diseño de la aplicación, tales como: diagramas de objetos, casos de uso, modelo de datos, se han generado utilizando starUML.

5.3.4 *Draw.io*

Draw.io es una aplicación en línea de código libre que permite la creación de diagramas que se pueden exportar a imágenes. Esta herramienta ofrece multitud de opciones para la generación de diagramas además de estar integrado totalmente con Google Drive. dicha integración permite salvar los diagramas como otro documento más en el sistema que Google ofrece para almacenamiento de documentos.

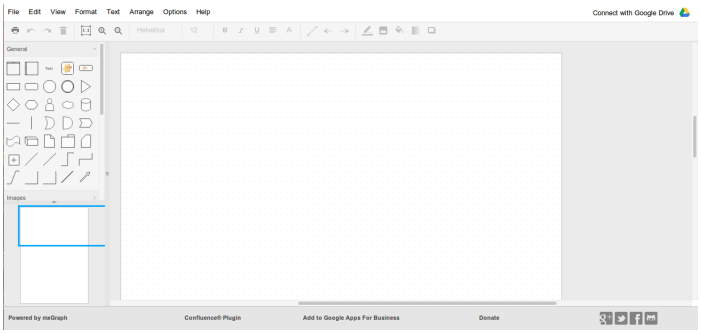


Figura 33: Figura de muestra de Draw.io

Esta herramienta se ha utilizado para el diseño de las interfaces de usuario facilitando enormemente el trabajo.

5.4 INTEGRACIÓN DE TECNOLOGÍAS

A continuación se va a explicar la interacción que existe entre todas las tecnologías descritas en este capítulo. Dado que el sistema está basado en la aproximación de cliente servidor, se verá donde se estarán ejecutando todas y cada una de las tecnologías.

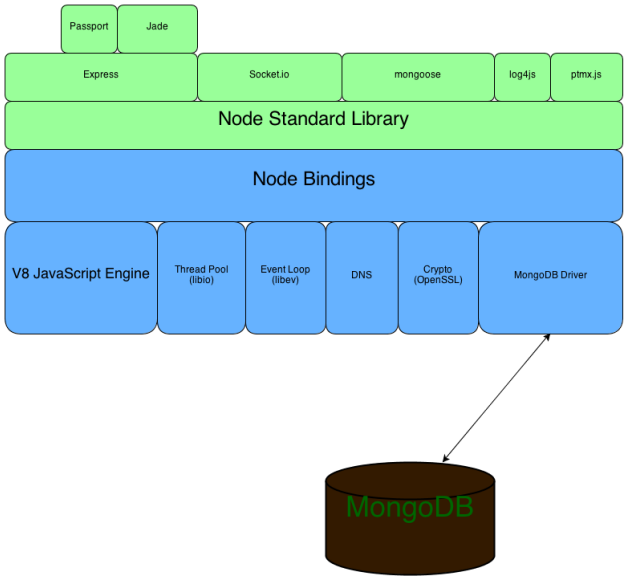


Figura 34: Servidor: Interacción entre NodeJs y todos los módulos utilizados

En el lado servidor está la base del sistema cuyo lenguaje de programación es principalmente C/C++. Con la ayuda del motor V8 de Google se obtiene una potente herramienta para interpretar código Javascript, que es el lenguaje de programación utilizado en Node.js. Además, cualquier otro modulo o librería programada en C/C++ son fácilmente adaptables en Nodejs escribiendo el código para la API en Javascript.

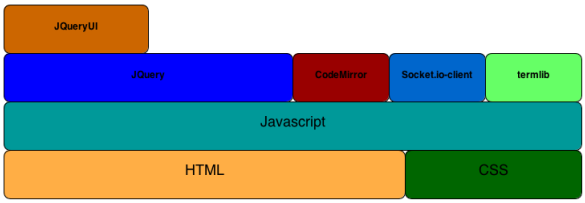


Figura 35: Cliente: Tecnologías utilizadas en el cliente

En el lado cliente se tiene, de base, el lenguaje HTML y el lenguaje para estilos CSS. Mediante el módulo jade, en el servidor, se renderizan los documentos HTML que posteriormente carga el cliente en el navegador. Además, se dispone del lenguaje interpretado Javascript que soportan prácticamente todos los navegadores actuales. Con ello se aprovechan las facilidades que ofrecen las librerías de jQuery y jQueryUI. Por otro lado, al ser una aplicación para ejecución en tiempo real, se necesita la librería CodeMirror para los editores de código y Socket.io para crear una conexión por websocket de forma permanente para el intercambio de mensajes entre cliente y servidor. Por último, mediante termlib se emula la ejecución del programa tal y como se tratara de una terminal en el servidor.

Por último, se mostrará un gráfico que relaciona las diversas tecnologías utilizadas.

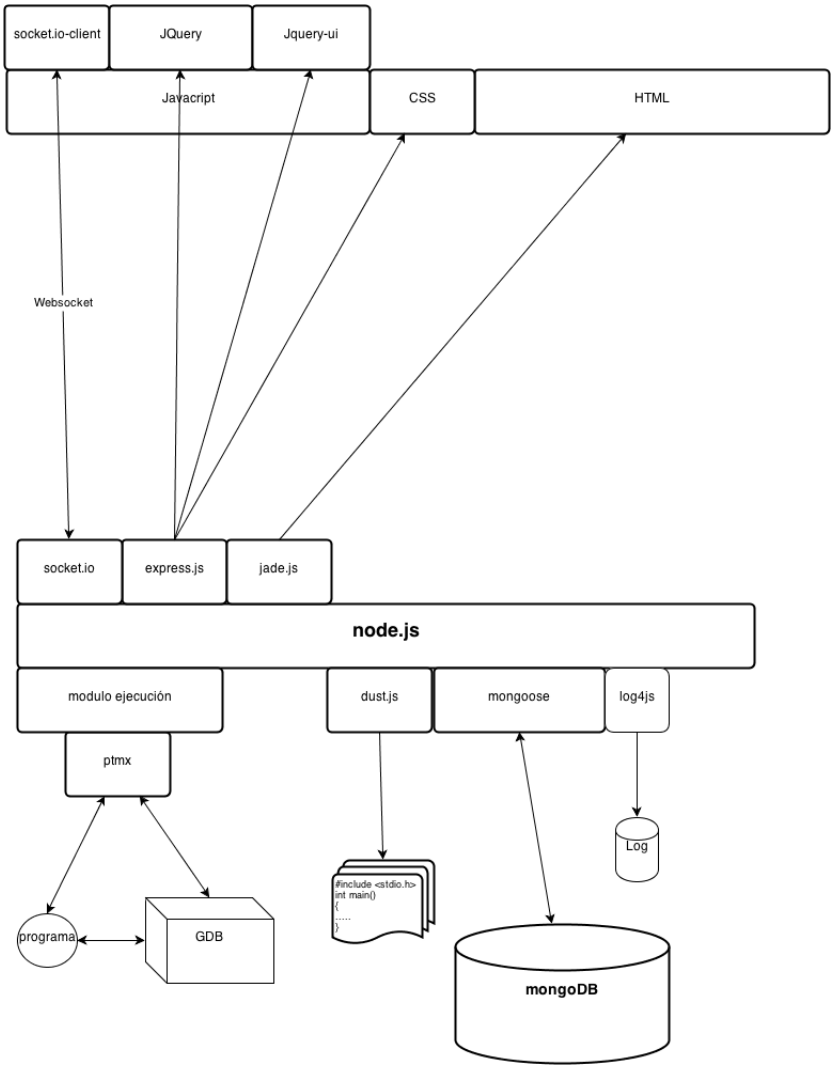


Figura 36: Gráfico que muestra la relación entre las tecnologías.

6 SEGURIDAD, ESCALABILIDAD Y EFICIENCIA

Como se ha explicado en el apartado 4 y razonado en la introducción se ha optado por el diseño de la herramienta Catalyde con una arquitectura cliente servidor. Entre otras cosas, la aplicación permite que el profesor recopile datos del desarrollo de las prácticas por parte de sus alumnos, lo que permite un mejor seguimiento de los mismos.

En este capítulo se van a tratar varios requisitos prácticos a tener en cuenta en este tipo de arquitecturas: por un lado, la robustez del servidor ante diversos factores como pueda ser la carga y, por otro lado, la seguridad del sistema ante fallos, tanto por parte de los usuarios o causados por ataques malintencionados.

Uno de los principales puntos débiles de una herramienta de programación con arquitectura cliente servidor es la ejecución, tanto del compilador como del código compilado, en el lado servidor. Nada parece menos robusto ante un ataque que una máquina que deja a un usuario escribir un programa, compilarlo y ejecutarlo en el propio servidor.

Se debe remarcar que esto puede ser vulnerable incluso ante usuarios no malintencionados. Un ejemplo sería que un usuario quiera borrar un fichero propio y se equivoque en la ruta pudiendo borrar algún fichero del sistema que no sea el deseado. Algo todavía más probable en asignaturas de programación de primeros cursos es que un usuario lance la ejecución de un programa que provoca un bucle infinito. Si bien esto no causa problemas de seguridad, sí perjudica muy gravemente la eficiencia, pudiendo consumir tantos recursos que el sistema se volvería totalmente inservible.

La parte de compilación presenta menos problemas que la propia ejecución. Si bien no es técnicamente inconcebible incluir el compilador en el lado cliente, supone una mayor complejidad y tiene el inconveniente de que se necesitaría pasar al lado cliente el código fuente ocultado por el profesor, lo cual hace más fácil que el usuario pueda tener acceso al código del profesor. Así pues, no se va a considerar, en lo que sigue, esta posibilidad.

6.1 ESCALABILIDAD

En lo que respecta a la escalabilidad, debemos mencionar que para el desarrollo del servidor hemos optado por Node.js. Este entorno ha sido diseñado para evitar tener que crear un proceso distinto para servir a cada usuario. En lugar de ello, utiliza un único bucle que atiende todas las peticiones. Esto se consigue mediante el uso de una

interfaz totalmente asíncrona dirigida por eventos. Existe un único hilo de ejecución que atiende todas las peticiones una detrás de otra, cada petición nueva de un usuario considerándose un evento. Cuando se recibe una petición, Node.js comienza a procesarla y cuando se encuentra con una operación de entrada-salida no espera a que ésta finalice, sino que registra un función de *callback* y continúa procesando el siguiente evento. Cuando la operación de entrada-salida acaba, se produce otro evento, y node lo procesa ejecutando la función de *callback* cuanto antes. La clave está en el uso de una API completamente asíncrona y en el uso de un *pool* de hilos de ejecución que permiten sacar el máximo provecho de todos los núcleos disponibles en la máquina. De este modo node no necesita nunca crear hilos adicionales lo que supone que la carga en el servidor es muy baja. El inconveniente que tiene esto es que a la hora de programar si se añade en cualquier parte del código algún tipo de espera activa (es decir una llamada a *sleep* o un bucle infinito) el proceso deja de atender cualquier evento quedándose atascado en dicha llamada. A continuación se muestra un ejemplo de código en un lenguaje secuencial como C++ y otro en Node.js, de manera que se puedan extraer algunas conclusiones sobre el funcionamiento de Node.js:

Algoritmo 6.1 Código ejemplo lectura de fichero en C++.

```
int main() {
    char cadena[128];
    ifstream fichero("prueba.txt");
        cout << "Contenido fichero:"<<endl;
    while(!fichero.eof()) {
         fichero >> cadena;
         cout << cadena << endl;
    }
    fichero.close();
    return 0;
}
```

Mientras que en node para realizar la misma operación escribiríamos el siguiente programa:

Algoritmo 6.2 Código de ejemplo de lectura de fichero en node.

```
var fs = require('fs');

fs.readFile('prueba.txt', function (err, data) {
        if (err) throw err;
        console.log(data);
});

console.log("Contenido fichero:");
```

Los dos programas anteriormente mostrados realizan la misma operación, es decir, leer el contenido de un fichero y escribirlo por pantalla, escribiendo antes "Contenido del fichero:". El programa escrito en C++ es un programa totalmente síncrono y sabemos que siempre se comportará igual, pero en el programa escrito en Node.js, la lectura del fichero es asíncrona, por tanto no sabemos cuándo va a escribir el contenido de fichero. Sin embargo, la siguiente instrucción se ejecuta y escribe "Contenido del fichero:" sin tener que esperar a que acabe la lectura del fichero. Cuando se ha leído el contenido del fichero se ejecuta la función de callback escribiendo por pantalla el contenido de éste. Con este ejemplo podemos remarcar que "Contenido del fichero:" se escribe antes pero no porque tenga que hacerlo sino porque la operación de lectura de fichero es una operación de entrada-salida, que es mucho más lenta que la simple escritura de una cadena por pantalla.

Por otro lado, si bien se conocen diseños para poder escalar mediante varios servidores Node.js de manera bastante sofisticada, en esta aplicación resulta suficiente mantener un servidor, ya que:

- Resulta sencillo dividir la carga por grupos de alumnos. Un solo servidor puede tratar el tráfico habitual asociado a peticiones que apenas requieren cálculo en el lado servidor incluso para grupos de alumnos suficientemente grandes.
- Se puede delegar en otras máquinas, y de manera muy sencilla, las partes computacionalmente más costosas: la compilación y ejecución del código de los alumnos, pudiendo también disponer del servidor de mongoDB en otra máquina si fuese necesario.

6.2 SEGURIDAD

El problema de la seguridad se debe abordar por todos los frentes:

Para evitar el problema de incluir funcionalidades del sistema, siempre que no sean estrictamente necesarias para la realización de las prácticas, se pueden sustituir las librerías necesitadas por los alumnos por otras más limitadas que tengan las funcionalidades justas para el objetivo de la práctica.

Otra opción complementaria consiste en ejecutar el programa en un entorno controlado, lo que se conoce en inglés como "*sandbox*". Existen multitud de mecanismos para realizar este tipo de entornos en el sistema operativo unix/linux, que es el utilizado en este proyecto. Entre otras cosas, se pueden destacar las siguientes:

- Ejecutar el código del alumno en un usuario con pocos privilegios. Aun así, el usuario podría hacer prácticamente todo lo que podría hacer ese mismo usuario con un *shell* en dicha máquina.

- Ejecutar el código dentro de una "jaula" *chroot* que impide que el usuario pueda acceder fuera de una determinada carpeta. En particular, podemos limitar qué programas del sistema pueden ser accesibles para él.
- Utilizar otros mecanismos de seguridad incorporados en el kernel del propio sistema operativo Linux para controlar de manera personalizada el acceso a determinados recursos por parte de los procesos. Existen varias alternativas y en el proyecto se ha elegido "AppArmor" que actualmente se encuentra en la distribución de linux "Ubuntu", que es la utilizada en este proyecto. AppArmor es un módulo que permite asociar a cada programa un perfil de seguridad definiendo las restricciones sobre dicho programa. El módulo se creó como alternativa a SELinux, un sistema que proporciona políticas de seguridad en sistemas Linux, ya que éste se consideraba un sistema difícil de configurar y mantener. A diferencia de SELinux, AppAmor se puede aplicar sobre cualquier sistema de fichero con lo que en ese sentido no se ve limitado.

Aunque AppArmor permite crear perfiles para limitar el tiempo de CPU de un programa, las implementaciones actuales no permiten utilizar esta opción, así que se ha tenido que buscar una alternativa.

Se necesita limitar el tiempo total acumulado de CPU ante la eventual aparición de programas con bucles infinitos. Es bien sabido que el problema de la parada es indecidible, pero en la práctica resulta muy razonable limitar a priori el tiempo de CPU puesto que el profesor puede estimar el tiempo de CPU que, de manera razonable, utilizarán las soluciones de los ejercicios que propone.

Sería un error limitar el tiempo "real" o tiempo de "reloj", puesto que algunos usuarios pueden tomarse mucho más tiempo que otros en, por ejemplo, introducir un dato que se leerá por el teclado. Mientras el programa espera que el usuario introduzca ese dato por el teclado, el proceso está parado sin consumir CPU (consume otros recursos, tanto memoria de la máquina como descriptores de ficheros y otros recursos en el propio servidor). De ahí que el tiempo real pueda ser bastante alto sin que ello implique un problema de un bucle infinito. Es decir, que existen dos límites de tiempo, uno del tiempo de CPU y otro del tiempo real.

Por otro lado podría darse el caso de que un alumno opte por cerrar de golpe el navegador con su sesión abierta y con un programa en ejecución (incluso de manera involuntaria: podría haber un corte en la red). En este caso, existe un mecanismo para detectar que la sesión de dicho usuario ha quedado abandonada y el proceso en ejecución de dicho usuario debería de ser destruido (esto serviría como límite del tiempo real). Para resolver todos estos problemas se ha diseñado un pequeño *script* que se limita a ejecutar periódicamente (aproximadamente cada 20 segundos) unas llamadas a sistema capaces de obtener

la lista de todos los procesos lanzados por los usuarios. Si alguno de estos procesos ha quedado huérfano o si su tiempo de CPU supera un umbral definido, el proceso se elimina liberando los recursos que éste está reteniendo.

Se ha analizado el funcionamiento de dicho *script* y se ha observado que mantener este proceso en marcha no consume recursos del sistema de forma significativa. Por ello es viable utilizar el *script* para controlar los procesos que se están ejecutando.

7 CONCLUSIONES Y TRABAJO FUTURO

7.1 CONCLUSIONES

En este proyecto final de carrera se ha desarrollado una herramienta de aprendizaje de introducción a la programación enfocada a estudiantes de primeros cursos. De este modo se ha llevado a cabo una aplicación cliente-servidor conocida como Catalyde.

Esta herramienta, entre otras cosas, presenta una interfaz sencilla y amigable. Así, los alumnos se pueden adaptar fácilmente a la misma para centrarse más en resolver los ejercicios propuestos que en aprender a usar el entorno de programación.

Cuando un alumno empieza a programar mediante este sistema dispone de ejercicios donde se integran los enunciados junto a las partes prácticas, es decir, junto a las secciones donde es posible escribir el código. Esto le ayuda a centrarse en cada momento en un problema o desafío de programación, afianzando así los conocimientos adquiridos en las clases de teoría.

Por otra parte, la herramienta permite al alumno ejecutar los programas escritos por él mismo, interactuando con ellos a través de una consola, permitiendo además, la posibilidad de activar el código escrito por él mismo o por el profesor, según la necesidad en cada momento, y comprobar su correcto funcionamiento. Poder alternar entre el código del alumno y un código por defecto permite al alumno ejecutar el programa escribiendo sólo algunas partes del mismo, ya que el resto del código estará escrito por el profesor, facilitando así la corrección de ciertas partes del código. Estas ideas han sido presentadas en la V Jornada de Innovación Docente JIDINF'12 organizadas por la Escuela Técnica Superior de Ingeniería Informática (ETSINF) Stanciu et al. [40].

Por último, el sistema se ha implementado para facilitar su uso desde diferentes sistemas operativos y distintos navegadores web, de forma que el alumno pueda utilizarla cómodamente sin necesidad de instalar ninguna aplicación en su ordenador. Esto da una gran flexibilidad para el desarrollo de las prácticas, pudiendo añadirse exámenes o ejercicios complementarios que el alumno podrá realizar desde casa.

Aparte de la realización de las prácticas por parte del alumno, la herramienta también se ha realizado haciendo hincapié en la forma de desarrollar las prácticas por parte del profesor. Durante el diseño de la aplicación, se ha seguido la máxima de *hacer fácil lo sencillo y posible lo difícil*. Es decir, un profesor puede realizar prácticas más convencio-

nales (ejercicios que constan de un enunciado y un trozo de código que se ejecuta directamente) utilizando plantillas estándar y modificando el contenido. Pero la aplicación es suficientemente flexible como para que el profesor interesado pueda modificar estos *scripts* y realizar ejercicios más complejos. Por ejemplo, dibujar elementos por pantalla, reutilizar funciones implementadas en un enunciado en diferentes programas, generar casos de prueba, etc...

A nivel de desarrollo, la implementación realizada en este proyecto ha supuesto todo un reto por la gran cantidad de tecnologías que se han tenido que integrar. Se trata de tecnologías que han sido desarrolladas recientemente (NodeJs, MongoDB) y que están en pleno auge debido a la gran aceptación que están teniendo en la comunidad de desarrolladores. Esto que *a priori* supone una ventaja, añade doble dificultad al desarrollo de la aplicación:

- El desconocimiento no sólo de las herramientas, sino también de los paradigmas de programación. Esto supone un desafío y acerca al proyectando a tecnologías que están siendo demandadas en el mundo laboral actualmente.
- La poca información que se encuentra disponible y que además está sujeta a cambios, lo cual ha llevado a participar activamente en foros de desarrollo (como StackOverflow, o github) e incluso a ponerse en contacto con los principales desarrolladores de las tecnologías mencionadas.

Aunque se ha tratado de un proyecto muy ambicioso que, por sus requisitos, precisaría de un esfuerzo considerable por parte de un equipo de desarrollo, estamos muy satisfechos de que ya sea posible utilizar la herramienta con alumnos reales.

7.2 TRABAJO FUTURO

El actual proyecto se ha realizado pensando en ofrecer al usuario una infraestructura para poder ejecutar programas escritos por él mismo con la funcionalidad de poder activar y desactivar partes de una versión completa (aunque oculta) del programa, cosa que facilita enormemente el entendimiento de lo que se le está pidiendo, pues tiene la opción de ejecutarlo desde el principio. Muchas de las mejoras y ampliaciones han quedado fuera del alcance de este proyecto debido fundamentalmente a la enorme envergadura del mismo. Por tanto, no es de extrañar que la herramienta, conforme ha sido implementada y desarrollada, presente un sinfín de mejoras y de posibles ampliaciones. A continuación, enumeramos algunas de las funcionalidades que pueden mejorar o completar la aplicación desarrollada:

- **Formulario de creación de las prácticas.** Adaptar el lado cliente para que el profesor pueda generar el contenido de la práctica

con mayor comodidad mediante un editor que permita ver de forma instantánea el resultado de lo que está haciendo, sin tener que esperar a introducir el contenido en el sistema, como ocurre actualmente.

- **Trabajo colaborativo.** En las clases prácticas se permite el trabajo en grupo. Normalmente los alumnos forman grupos de al menos dos personas. La mejora propuesta debería de permitir que el ejercicio que resuelvan juntos pueda estar disponible para todos los miembros del grupo simplemente indicando los componentes del mismo.

- **Estadísticas y modo de evaluación.** Para que el profesor tenga más control sobre el trabajo que están realizando los alumnos, añadir la posibilidad de que éste pueda observar el contenido que están añadiendo los alumnos a las prácticas. Además tener estadísticas del tiempo que han estado escribiendo código, días que han accedido al sistema, intentos de ejecución del programa.

- **Acceso programado.** Para controlar el acceso a las prácticas añadir control de acceso para que éstas sólo estén disponibles para el grupo al que pertenece un alumno durante un periodo de tiempo determinado por el profesor.

- **Ejecución en modo depuración.** Los alumnos pueden ejecutar el programa pero no disponen de ninguna herramienta para depurar el mismo. Añadir soporte para poder utilizar algún depurador requiere que el alumno pueda añadir puntos de interrupción y que pueda ver el contenido de las variables cuando se produce una parada. Se ha investigado el uso de GDB por parte de diversos IDEs y, como trabajo futuro, se propone el uso de la interfaz GDBMI de GDB para añadir la posibilidad de realización de trazas y de depuración de los programas en futuras versiones de la herramienta. GDBMI es la abreviatura de "GDB machine interface". Es un modo de GDB donde los mensajes que proporciona el depurador cumplen una sintaxis que resulta más fácil de analizar por parte de un programa, es decir, está pensado para que lo procese un analizador sintáctico más que para que lo lea un ser humano. Dado que en el servidor ya se utiliza GDB para el control del programa, se puede adaptar este depurador para esta ampliación.

- **Modo examen.** La realización de exámenes prácticos se ha venido elaborando habitualmente en papel. Entre los diversos motivos y posibles ventajas para realizar los exámenes de este modo, con independencia de que actualmente se disponga de mayor facilidad para realizar el examen con un ordenador, podemos destacar que:

- En un ordenador pueden perderse los datos si hay una avería o un fallo de corriente, mientras que en papel es mucho más difícil que se extravíe.
- Es más difícil falsificar la letra de un alumno que verificar la suplantación de autoría de un fichero generado en un sistema informático.
- El alumno dispone de más facilidades para copiar si el ordenador tiene acceso a red, no solamente copiar la solución sino también acceder a apuntes de teoría, a programas similares o incluso a comunicarse con otras personas durante la realización del examen

A pesar de los inconveniente que presenta esta ampliación, pensamos que es una funcionalidad que mejoraría la utilidad de la misma dando la posibilidad de centralizar los ejercicios de un examen en el sistema. Además, tomando las precauciones suficientes, se podrían evitar los problemas planteados. Pensamos que un modo examen es fácil de implementar en un entorno web combinado la posibilidad de solicitar el modo de pantalla completa con la detección de la pérdida del foco de la aplicación. También es posible incluir mecanismos semi-automáticos de detección de posibles copias. En este caso, se dispone de mucha más información al poder disponer no solamente del resultado final sino también de las etapas intermedias del desarrollo del programa.

- **Pérdida del foco.** Añadir un modo de funcionamiento en el que el sistema detecta que no hay actividad y el alumno no está realizando la práctica sino que, se está entreteniendo con otras cosas o, en caso de un examen, podría estar accediendo al correo electrónico o a algún servicio de chat para contactar con alguien que le pudiera facilitar las repuestas a algunos ejercicios del examen.

- **Chat y control a distancia.** La posibilidad de que el alumno pueda preguntar dudas al profesor mediante un sistema de chat puede ser importante en la aplicación, por ello una de las mejoras que se proponen es añadir un sistema de chat mediante el cual el alumno pueda realizar consultas al profesor. Esto podría venir acompañado de un mecanismo por el que el profesor puede inspeccionar a distancia la práctica realizada por el alumno e incluso tomar momentáneamente el control de modo que el alumno pudiera ver lo que realiza el profesor para corregir una duda puntual. Esto facilitaría todavía más el uso de la aplicación en entornos de educación a distancia.

8 ANEXOS

8.1 ANEXO 1: CREACIÓN DE LA PRÁCTICA

Se presenta un manual donde se explica el trabajo que se debe realizar para la definición una práctica por parte del profesor. Todo este trabajo se realiza fuera de la aplicación con los recursos que dispone el usuario. Realmente lo único que se necesita es un editor de texto, por lo tanto se puede usar el editor de texto preferido por el usuario.

Una práctica está definida por un conjunto de recursos que por separado no tienen ningún sentido, pero si se juntan, dan lugar a una práctica que permite ejecutar código, mostrar contenido textual y gráfico en forma de botones.

Para entrar en más detalles una práctica está constituida por los siguientes documentos:

- Fichero ".md": es el documento con el boletín de la práctica en formato Markdown y con marcas indicando el inicio y el fin de cada bloque, así como sus propiedades, donde se escriben los enunciados de la práctica y se definen las secciones donde el usuario puede introducir código o simplemente texto. Para una práctica solo hay un documento de este tipo.

- Fichero ".dust": son plantillas que se utilizarán en las etapas de compilación del código. Esto sirve para unir los fragmentos de código del alumno o del profesor y generar un programa que se pueda compilar. En la misma práctica pueden haber muchos documentos de este tipo, por lo menos uno por cada programa (aunque para el mismo programa puede haber varios, dependiendo de la complejidad del programa).

- Fichero "pract.json": es el documento donde se definen aquellos ficheros que intervienen en la práctica (".md" y los ".dust"). También se define el nombre de la práctica desde el punto de vista de como lo reconoce el sistema.

- Fichero "conf.json": es el documento donde se define la configuración inicial de los bloques de código marcándolos como activos o inactivos.

- Fichero "action.js": un módulo Javascript que contiene un diccionario de acciones, es decir que asocia un repertorio de cadenas alfanuméricas que describen una acción a una función que se ejecuta para llevar a cabo dicha acción. En éste se puede definir aspectos como el compilador a usar, la orden que se

ejecuta cunado se compila, cuando se ejecuta y los manejadores en casos de fallo.

- Carpeta "content": es una carpeta pensada para recursos como imágenes o ficheros que no se modificarán (documentos destinados a ejercicios de lectura de fichero). Esta carpeta esta subdividida en dos subcarpetas "PublicFile" y "PrivateFile". Pudiendo dejar los recursos que se utilicen para acceso publico (es decir el usuario podría verlo) o para acceso privado (el usuario no sabe de la existencia de tal recurso).

Una vez se tiene claro el objetivo de cada fichero que forma parte de una práctica se detallarán los aspectos principales del contenido de los ficheros.

El archivo .md

Está compuesto por una serie de entidades llamadas bloques. Un bloque es una sección que empieza con una etiqueta BEGINBLOCK y acaba con una etiqueta ENDBLOCK. Las etiquetas de inicio y de fin de bloque se delimitan por "<!–" al comienzo y por "–>" al final. Además la etiqueta de inicio de bloque presenta una serie de características definidas entre la etiqueta BEGINBLOCK y el delimitador de finalización de la misma ("–>") y se delimitan por dos llaves "{ características }". Las características se introducen en lenguaje JSON. Las posibles características que presentan los bloques son:

1. b_tag: define el id para ese bloque. Solo es útil en caso de ser un bloque de código, ya que es la forma de enlazarlo con las platillas de código.
2. type: define el tipo de bloque que puede ser:
   a) code - código
   b) text - texto
   c) markdown - código de marcado Markdown
3. checkbutton: Indica si se muestra un botón de activación/desactivación de código. Tiene valor booleano "true" o "false" y solo es aplicable a bloques de código.
4. editable : Indica si el bloque es editable o no editable y tiene valor booleano "true" o "false".
5. buttons: Es una lista de botones que se mostrarán con el bloque. Cada botón tendrá los siguientes atributos:
   a) name : indica el nombre que tendrá el botón (texto que se mostrará en la interfaz web).

b) action : acción que se ejecutará en el servidor cuando se pulse el botón. Esta acción deberá definirse con el mismo nombre en el fichero "actions.js".

El archivo .dust

Se declarará la lógica del código para la compilación. Mediante un ejemplo se puede entender mejor este tipo de ficheros.

Algoritmo 8.1 Ejemplo de código en el fichero .dust

```
#include <stdio.h>

{?funcion_perimetro_rectangulo}
{#load blockid=id.funcion_perimetro_rectangulo /}
{:else}{~n}
float perimetro_rectangulo(float base, float altura)
{
    return 2*(base+altura);
}
{/funcion_perimetro_rectangulo}

int main() {
        float base, altura, perimetro;
        fprintf(stdout,"Introduce la base: ");
        fscanf(stdin,"%f",&base);
        perimetro = perimetro_rectangulo(base,altura);
        fprintf(stdout,"El perimetro es: %f\n",perimetro);
return 0;
}
```

En la sección 8.1 de código se puede ver una regla para componer el fichero que será compilado. Por un lado el identificador de la regla es "funcion_perimetro_rectangulo". Este identificador es el que se ha definido el el fichero .md en el bloque de código que se ha introducido. Si el usuario tiene activado el código en la salida aparecerá el contenido que se carga de la base de datos el bloque con el id "funcion_perimetro_rectangulo", para ese usuario en concreto y sino se pondrá el contenido que sigue a la sentencia "else". Una regla comienza de la siguiente forma: "{?idenficador}" y lo que sigue a continuación es el contenido que se cargará en caso de que el alumno tenga activada la opción de usar su código. La aparición de la sentencia "{:else}" significa que si el usuario tenia desactivada la opción de usar su código se usará lo que haya a continuación hasta el final de regla, que se compone por llave y una barra seguida del identificador del bloque "{/identificador}", en este caso concreto "{/funcion_perimetro_rectangulo}".

Algoritmo 8.2 Ejemplo de un fichero pract.json

```
{
        "title" : "Práctica 1",
        "longtitle" : "Ejercicios de funciones",
        "practice": "pract.md",
        "dustTemplates":["area_rectangulo.dust","perimetro.dust"]
}
```

El archivo pract.json

En este archivo se encuentra la configuración de la práctica determinándose, que ficheros se utilizarán para componer la práctica y el título de la prácticas.

1. title: Cadena que indica el título de la práctica que se usará en los menús de la aplicación.
2. longtitle: Cadena que indica el título de la práctica, pero más largo. Destinado para las zonas que no sean menús de la aplicación.
3. practice: Indica el fichero .md que define el cuerpo de la práctica.
4. dustTemplates: Lista de plantillas dust que se van a utilizar en la práctica del directorio dust, deberán ser coherentes con el fichero de la propiedad "practice", ya que el comportamiento no será el deseado.

El archivo conf.json

Tal y como se ha descrito al principio de este manual, este fichero sirve para marcar el estado inicial de un bloque de código. La sección donde se reflejará esto se llamará "status". Como bien lo pone en el nombre del archivo configuración, el lenguaje utilizado para esto es JSON. Para marcar un bloque como activado o desactivado se debe poner como clave el identificador del bloque de código (el que marcamos como b_tag en el fichero .md) y como clave el "true" o "false". En la sección de código 8.3 se puede ver un ejemplo de un fichero conf.json.

Algoritmo 8.3 Ejemplo de fichero conf.json

```
{
        "status": {
                "funcion_area_rectangulo":         false,
                "pedir_altura":                    false,
                "funcion_perimetro_rectangulo":    false,
                "cuerpo_funcion_volumen_cilindro": false,
                "cuerpo_funcion_segundos_dia":     false,
                "funcion_pedir_entero":            false,
                "programa_intervalo":              false
        }
}
```

El archivo actions.js

En este fichero se definen las acciones que se ejecutarán al pulsar sobre un botón en la interfaz web, definiéndose las funciones que se enlazan con los botones descritos en las propiedades del archivo .md. Mediante un ejemplo se podría ver mejor como se estructura este fichero.

Algoritmo 8.4 Ejemplo de fichero actions.js

```
module.exports = {
    "run_area":  function(args,obj) {
                   obj.compileAndRun(args,{
                               files: [],
                               templates: [["area_rectangulo.
                                   dust","area.c"]],
                               compiler: "clang",
                               compileArgs: ["-o","area","area.c
                                   "],
                               compileErrorHandler: "default",
                               runCommand:"area",
                               runErrorHandler: "default",
                               interactive: true,
                               handler: "terminal"
                   });
                 }
}
```

El algoritmo 8.4 muestra como se definiría la regla de ejecución al pulsar sobre el botón que se enlaza con esta regla. A continuación se detallan los parámetros que aparecen en la configuración anterior:

1. files: Lista de ficheros que se usarán para ejecutar (son los ficheros que se dejarían en "content").

2. templates: lista de plantillas dust que se van a utilizar o bien de archivos con código.
3. compiler: El compilador a utilizar.
4. compileArgs: Lista de argumentos para la compilación.
5. compileErrorHandler: El manejador en caso de fallo de compilación. Existe uno por defecto pero se deja la posibilidad de manejar los errores de otra manera.
6. runCommand: Indica la linea de commandos que se ejecutará al realizarse la acción.
7. runErrorHandler: Manejador en caso de que falle la ejecución. Existe uno por defecto pero se deja la posibilidad de manejar los errores de forma personalizada.
8. interactive: valor booleano que indica si es ejecución interactiva o no.
9. handler: Indica el manejador para la ejecución por defecto es "terminal" y de momento solo se soporta una terminal.

8.2 ANEXO 2: EJEMPLO DE USO

Una parte relevante a la hora de presentar un aplicación es ver su funcionamiento, aparte de especificaciones, análisis, diseño, y demás etapas para llevar a cabo su desarrollo. El objetivo de este apartado es mostrar de forma sencilla cómo utilizar la aplicación desarrollada, dando una idea general de lo que el usuario puede hacer cuando esta conectado a ella, mediante ejemplos que se han utilizado para las pruebas durante el desarrollo de la misma.

Lo primero que observa el usuario al conectarse a la aplicación es la pantalla de inicio, tal y como se ve en la Figura 37.

Figura 37: Página de inicio de sesión de la aplicación Catalyde.

Se pide un usuario y una contraseña que controla el acceso a la aplicación, pudiendo obtener algunos mensajes de errores en caso de introducir los datos incorrectamente tal y como se observa en la Figura 38. En la parte izquierda se muestra el error cuando algún campo está vacío y en la parte derecha el error cuando la contraseña se introduce mal o el usuario no se encuentra en el sistema.

Figura 38: Mensajes de error que se pueden obtener al iniciar sesión.

Una vez se haya realizado la identificación con éxito se accede a la pantalla principal. Se pueden identificar diferentes elemento como la barra de menú en la parte superior, en el lado izquierdo el panel de prácticas, y el resto del espacio que se encuentra vacío se situará el visor de prácticas.

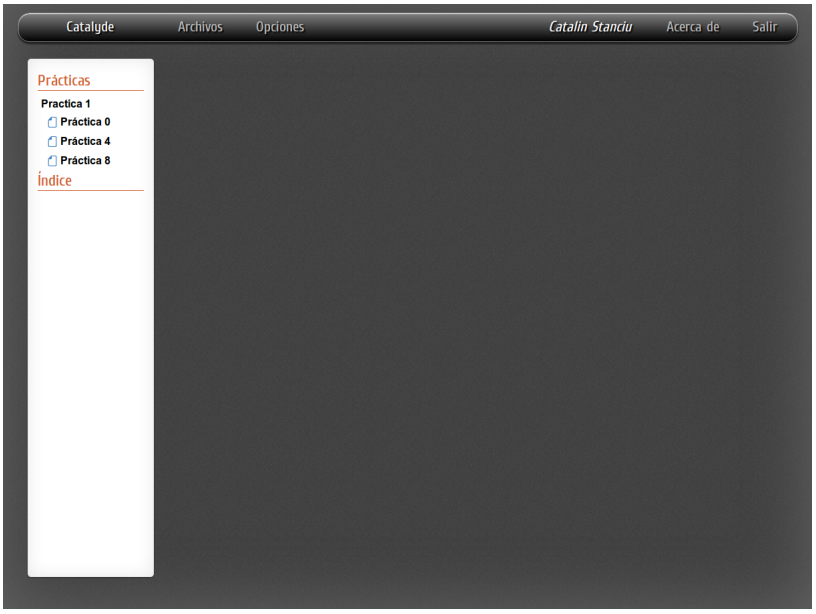


Figura 39: Vista general de la aplicación.

En el barra de menú hay dos opciones desplegables que se muestran en la Figura 40.

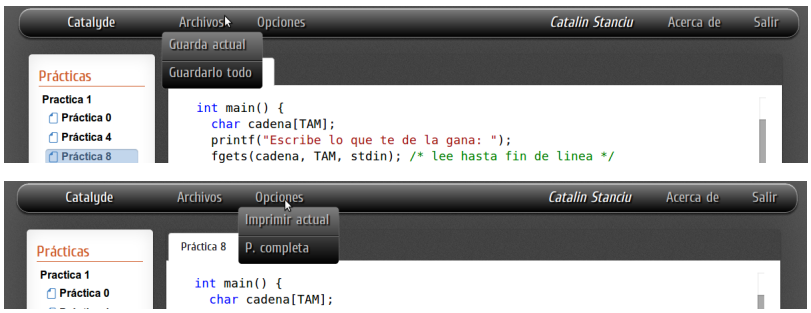


Figura 40: Opciones de la barra de menú.

Si se abre una práctica se abre un visor y se muestra el contenido de la misma. En este ejemplo se pueden identificar zonas de enunciado, zonas de código no editable perteneciente al enunciado y zonas de código editables. Además hay varios botones para las zonas de edición de código.

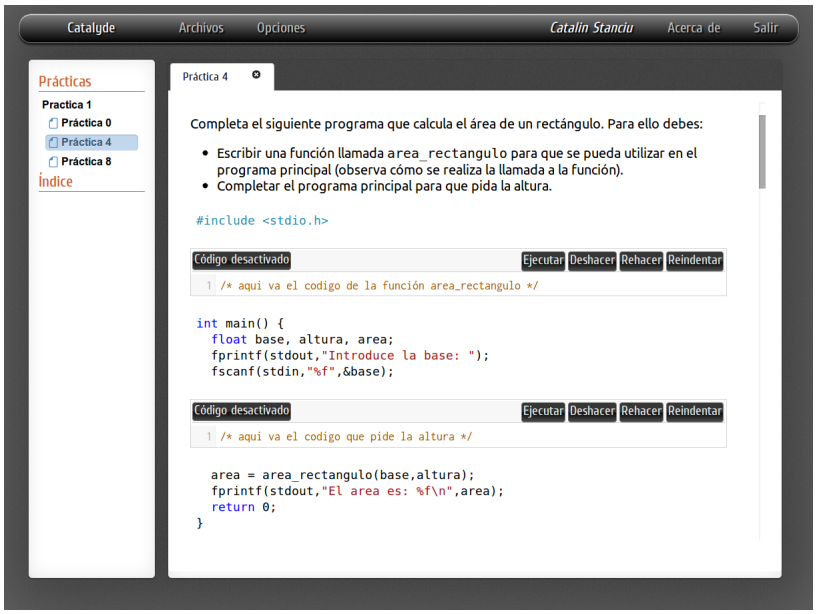


Figura 41: Ejemplo de práctica abierta.

Si se ejecuta el programa anterior con la versión del profesor, se puede ver como se inicia una terminal donde se puede interactuar con el programa. Para identificar si se usa el código del alumno o del profesor se utiliza el botón "Código desactivado", que indica que el código utilizado es el del profesor.

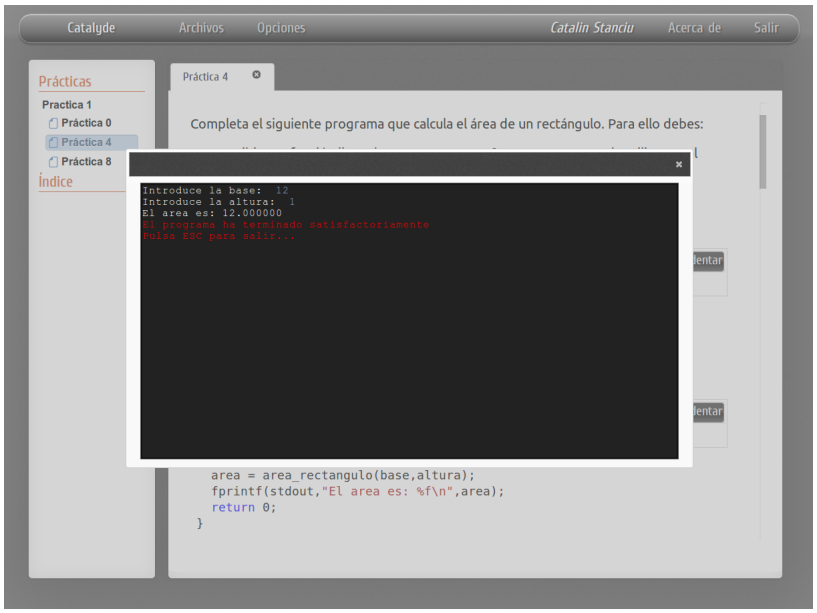


Figura 42: Programa en ejecución.

Puede haber un error en tiempo de compilación por lo que se le mostrará al usuario.

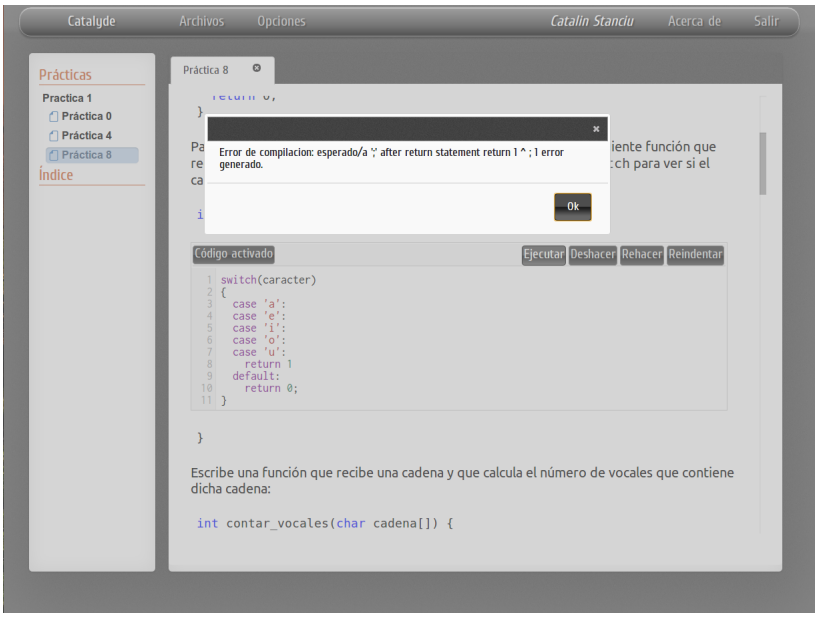


Figura 43: Ejemplo de error de compilación.

Como se puede ver en las capturas, se utiliza una interfaz sencilla e intuitiva que facilita el trabajo del usuario de la aplicación.

8.3 POSTER ENVIADO A JDINF'12

UNIVERSIDAD POLITECNICA DE VALENCIA

Catalyde: herramienta para enseñar programación

DSIC

Catalin Stanciu, Joan Pastor, Salvador España
cstanciu@dsic.upv.es, jpastor@dsic.upv.es, sespana@dsic.upv.es
Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia

Motivación

- Aprender a programar requiere del desarrollo de habilidades por medio de la práctica. Del mismo modo que no aprendemos a ir en bicicleta con una bici de carrera, también tiene sentido aprender a programar en unas condiciones que faciliten el proceso de aprendizaje.

- Históricamente se ha planteado el desarrollo de lenguajes con fines pedagógicos (ej. Logo), así como entornos más orientados al aprendizaje que al uso profesional (ej. para java, utilizar Bluej en lugar de Eclipse).
- Se presenta una herramienta para facilitar el aprendizaje a la programación. Está orientada principalmente a las asignaturas de primeros cursos, en particular, para asignaturas que utilizan el lenguaje C (aunque puede adaptarse a cualquier otro lenguaje de programación como java). Se ha elegido C para poder implantar la herramienta en las asignatura de informática de los grados de ingeniería técnica industrial en la UPV (la ETSINF utiliza Java en los primeros cursos).

Situación actual

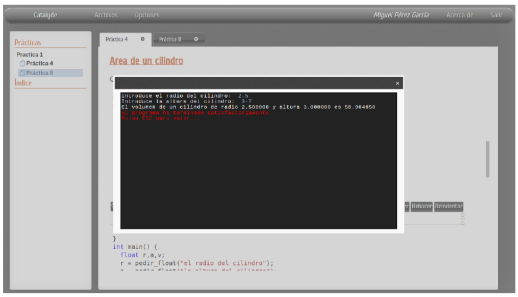
- **Por parte del profesor:** Tradicionalmente, la elaboración de las prácticas consiste en el desarrollo de un boletín, la implementación de una solución y, en algunos casos, la selección de una parte de la misma para que el alumno la complete. Finalmente el material se deja en poliformaT.
- **Por parte del alumno:** bajarse el boletín y los fragmentos de código a completar, manipular las distintas versiones que se van realizando en las carpetas de su ordenador, utilizando un entorno de desarrollo (por ejemplo, en el grado de ingeniería técnica industrial se utiliza devcpp) y finalmente entregar las soluciones, si así se solicita, mediante correo electrónico o vía poliformaT.
  - Normalmente el alumno no tiene acceso a una versión funcional del programa para poder probarlo y así entender más fácilmente qué es lo que se pretende resolver. La experiencia nos indica que poder ejecutar la solución ayuda enormemente a entender lo que se le pide.
  - Aunque proporcionar una solución compilada puede mejorar la comprensión del ejercicio, el alumno no podrá utilizarla para validar las distintas partes realizadas por él y normalmente no podrá saber si una parte es correcta hasta haber completado todo el ejercicio.

Sistema propuesto

- Se trata de una herramienta vía web en la que el alumno se autentifica y puede acceder a las prácticas elaboradas por el profesor.
- La descripción de las prácticas contiene zonas de edición de código donde el alumno puede escribir su solución. Pudiendo además, activarla y desactivarla (se utiliza la solución correcta oculta).
- El alumno puede probar en todo momento la solución oculta para entender lo que se pide o sustituir partes de ella por su propia solución.
- Existe la opción de comprobar la validez de las distintas partes realizadas por el alumno de manera independiente del programa final.
- Las diferentes partes de la práctica se pueden realizar en ejemplos sencillos y combinarse en un programa más complejo que el alumno puede probar. De esta forma, el alumno puede analizar, estudiar y probar dicho código.

¿Qué aporta esta herramienta para aprender a programar?

- Poder trabajar con los programas de manera incremental. El profesor puede dividir el código en apartados y el alumno podrá activar o desactivar su propio código de modo que pueda ejecutar el programa independientemente de que se haya completado todo.
- Centrarse en los aspectos concretos que el alumno debe aprender. Hasta ahora, el profesor proporciona un esqueleto del programa, pero no hay garantía de que el alumno respete la estructura.
- Poder validar determinados fragmentos de código mediante el uso de casos de prueba. Esta validación es parcial en la medida en que pasar unos casos de prueba no garantiza la corrección del algoritmo pero sí a la inversa: no pasar estos casos indica que hay un error. Estas técnicas se utilizan también en los oráculos de los concursos de programación.
- Mejor seguimiento y evaluación del alumno: El sistema guarda todos los pasos intermedios realizados por el alumno. Facilita la corrección de las prácticas, así como la detección de copias.
- Ejecutar el programa e interactuar con él mediante un terminal emulado vía web permite utilizar la aplicación en cursos online y facilita la realización de prácticas desde casa. También elimina la necesidad de instalar el entorno por parte de los alumnos.



Ampliaciones futuras

- Modo examen:
  - Utilizar el mismo entorno que el usado normalmente en las prácticas.
  - No hace falta tener que subir un conjunto de ficheros como ocurre con los entornos offline.
- Trabajo colaborativo: permitir que varios alumnos trabajen en grupo o que si varios alumnos se sienten en un mismo ordenador puedan compartir el trabajo realizado en dicha sesión, mientras actualmente están obligados a entrar con la cuenta de uno de ellos
- Modo evaluación y estadísticas: para que el profesor pueda navegar más cómodamente inspeccionando las prácticas realizadas por los alumnos, visualizar estadísticas, etc.
- Modo para que el profesor pueda editar e introducir los nuevos contenidos con más facilidad.

Conclusiones

- La aplicación permite focalizar a los alumnos en las diversas partes que forman un programa, centrándose así en la parte que el profesor quiera enfatizar en cada momento. Los alumnos, pueden verificar si una parte del código es correcta sin tener que completar todo el programa.
- El profesor escribe su programa y debe marcar lo que deben completar los alumnos. Los alumnos no necesitan acceder al código no relevante para aprender los conceptos requeridos.
- La aplicación ha sido diseñada para poder ser lo suficientemente flexible, para que el profesor pueda añadir funciones que verifiquen el funcionamiento del código, así como ejecutar diferentes instancias del programa con diferentes configuraciones. Por tanto, es el profesor el que decide mediante el diseño de las prácticas aquellos aspectos en los que el alumno deberá centrarse.
- Es criticable la falta de realismo respecto a un entorno de programación convencional:
  - Resulta necesario una transición progresiva a entornos más convencionales. Para ello se pedirá progresivamente ejercicios que requieran escribir un programa completo.
  - Resulta importante atraer a los alumnos con experiencias sencillas y exitosas y centrarse en la parte algorítmica. Una vez aprendido esto, saltar a un entorno real resultará relativamente sencillo.

[1] Khan Academy. https://www.khanacademy.org/, 2006. [Online; accedido 20-12-2012]. (Citado en la pagina 11.)

[2] The MongoDB Manual. http://www.mongodb.org/, 2011. [Online; accedido 16-10-2012].

[3] Express web application framework for node. http://expressjs.com, 2012. [Online; accedido 20-11-2012].

[4] Check iO. http://www.checkio.com/, 2012. [Online; accedido 20-12-2012]. (Citado en la pagina 13.)

[5] Codeacademy. http://www.codecademy.com/, 2012. [Online; accedido 20-12-2012]. (Citado en la pagina 11.)

[6] Codelearn. http://www.codelearn.org/, 2012. [Online; accedido 20-12-2012]. (Citado en la pagina 12.)

[7] Codeschool. http://www.codeschool.com/, 2012. [Online; accedido 20-12-2012]. (Citado en la pagina 12.)

[8] Coursera. http://www.coursera.org, 2012. [Online; accedido 20-12-2012]. (Citado en la pagina 10.)

[9] Udacity. http://www.udacity.com, 2012. [Online; accedido 20-12-2012]. (Citado en la pagina 10.)

[10] Udemy. http://www.udemy.com/, 2012. [Online; accedido 20-12-2012]. (Citado en la pagina 12.)

[11] Miriadax. http://miriadax.net/, 2013. [Online; accedido 08-2-2013].

[12] UPV[X]. http://www.upvx.es/, 2013. [Online; accedido 08-2-2013]. (Citado en la pagina 11.)

[13] Cristiana Areias and António Mendes. A tool to help students to develop programming skills. In *Proceedings of the 2007 international conference on Computer systems and technologies*, CompSysTech '07, pages 89:1–89:7, New York, NY, USA, 2007. ACM. ISBN 978-954-9641-50-9. doi: 10.1145/1330598.1330692. URL http://doi.acm.org/10.1145/1330598.1330692.

[14] Peter Bancroft and Paul Roe. Program annotations: feedback for students learning to program. In *Proceedings of the 8th Australasian Conference on Computing Education - Volume 52*, ACE

'06, pages 19–23, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc. ISBN 1-920682-34-1. URL http://dl.acm.org/citation.cfm?id=1151869.1151874.

[15] Zhixiong Chen and Delia Marx. Experiences with eclipse ide in programming courses. *J. Comput. Sci. Coll.*, 21(2):104–112, December 2005. ISSN 1937-4771.

[16] Kristina Chodorow and Michael Dirolf. *MongoDB: The Definitive Guide*. O'Reilly Media, Inc., 2010.

[17] Leliane Nunes de Barros, Ana Paula dos Santos Mota, Karina Valdivia Delgado, and Patricia Megumi Matsumoto. A tool for programming learning with pedagogical patterns. In *Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange*, eclipse '05, pages 125–129, New York, NY, USA, 2005. ACM. ISBN 1-59593-342-5.

[18] J.C.W. Debuse and M. Lawley. The learning and productivity benefits to student programmers from real-world development environments. 2011.

[19] Tomche Delev and Dejan Gjorgjevikj. E-lab: Web based system for automatic assessment of programming problems. 2012. (Citado en la pagina xiii y 14.)

[20] Llana Díaz, Enrique Martín Martín, and Cristóbal Pareja Flores. Correctores automáticos de programas: Implantación realista en la docencia universitaria. In *Actas del II Seminario de Investigación en Tecnologías de la Información: SITIAE 2008*, pages 31–50. Dykinson, 2009.

[21] Christopher Douce, David Livingstone, and James Orwell. Automatic test-based assessment of programming: A review. *J. Educ. Resour. Comput.*, 5(3), September 2005. ISSN 1531-4278. doi: 10.1145/1163405.1163409. URL http://doi.acm.org/10.1145/1163405.1163409. (Citado en la pagina 2.)

[22] Alireza Ebrahimi. Web teaching and learning programming environment based on plan method and constructs. In Deepak Kumar and Joe Turner, editors, *Education for the 21st Century Impact of ICT and Digital Resources*, volume 210 of *IFIP International Federation for Information Processing*, pages 309–313. Springer Boston, 2006. ISBN 978-0-387-34627-4.

[23] Eric Foxley, A Tsintsifas, CA Higgins, and Pavlos Symeonidis. Ceilidh, a system for the automatic evaluation of students programming work. *Proceedings of CBLISS*, 99, 1999.

[24] Stephen N. Freund and Eric S. Roberts. Thetis: an ansi c programming environment designed for introductory use. *SIGCSE Bull.*, 28(1):300–304, March 1996. ISSN 0097-8418. doi: 10.1145/236462.236560. URL http://doi.acm.org/10.1145/236462.236560.

[25] David B. Garlan and Philip L. Miller. Gnome: An introductory programming environment based on a family of structure editors. In *Proceedings of the first ACM SIGSOFT/SIGPLAN software engineering symposium on Practical software development environments*, SDE 1, pages 65–72, New York, NY, USA, 1984. ACM. ISBN 0-89791-131-8. doi: 10.1145/800020.808250. URL http://doi.acm.org/10.1145/800020.808250.

[26] Devon Govett. Learning Server-Side JavaScript with Node.js. http://net.tutsplus.com/tutorials/javascript-ajax/learning-serverside-javascript-with-node-js, 2010. [Online; accedido 10-05-2012].

[27] Kathryn E. Gray and Matthew Flatt. Professorj: a gradual introduction to java through language levels. In *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, OOPSLA '03, pages 170–177, New York, NY, USA, 2003. ACM. ISBN 1-58113-751-6. doi: 10.1145/949344.949394. URL http://doi.acm.org/10.1145/949344.949394.

[28] Marijn Haverbeke. CodeMirror - User manual and reference guide. http://codemirror.net/doc/manual.html, 2012. [Online; accedido 20-08-2012].

[29] Uriel Hdz. Introducción a Node.js. http://www.rmunoz.net/introduccion-a-node-js.html, 2011. [Online; accedido 20-01-2013].

[30] Uriel Hdz. Entendiendo Callbacks y Node.js. http://ecoding.me/2012/06/entendiendo-callbacks-y-node-js, 2012. [Online; accedido 20-01-2013].

[31] T. Dean Hendrix, James H. Cross, II, and Larry A. Barowski. An extensible framework for providing dynamic data structure visualizations in a lightweight ide. *SIGCSE Bull.*, 36(1):387–391, March 2004. ISSN 0097-8418. doi: 10.1145/1028174.971433. URL http://doi.acm.org/10.1145/1028174.971433.

[32] Manuel Kiessling. *The Node Beginner Book*. 2012. [Online; accedido 10-09-2012].

[33] Ingo Lutkebohle. Introducción a la programación asíncrona con Node.js. http://www.genbetadev.com/frameworks/

introduccion-a-la-programacion-asincrona-con-nodejs, 2011. [Online; accedido 22-07-2012].

[34] A. Moreno, N. Myller, and E. Sutinen. Jeco, a collaborative learning tool for programming. In *Visual Languages and Human Centric Computing, 2004 IEEE Symposium on*, pages 261–263, sept. 2004. doi: 10.1109/VLHCC.2004.33.

[35] Richard Rasala. Toolkits in first year computer science: a pedagogical imperative. *SIGCSE Bull.*, 32(1):185–191, March 2000. ISSN 0097-8418. doi: 10.1145/331795.331852. URL http://doi.acm.org/10.1145/331795.331852.

[36] Charles Reis and Robert Cartwright. Taming a professional ide for the classroom. *SIGCSE Bull.*, 36(1):156–160, March 2004. ISSN 0097-8418. doi: 10.1145/1028174.971357. URL http://doi.acm.org/10.1145/1028174.971357.

[37] Francisco P Romero, Jesus Serrano-Guerrero, and Hernan Pérez de Inestrosa. Cuestor: Una nueva aproximación integral a la evaluación automática de prácticas de programación. *Jornadas de Enseñanza Universitaria de la Informática (16es: 2010: Santiago de Compostela)*, 2010.

[38] Oscar Sapena, Mabel Galiano, Marisa Llorens, and Nati Prieto. Cap: Corrector automático de programas para la evaluación continua y el aprendizaje autónomo. In *V Jornada de Innovación Docente*, Valencia, Spain, December 2012. URL http://jidinf12.webs.upv.es/poster. (Citado en la pagina xiii y 14.)

[39] Ian Sommerville. *Software Engineering*. Addison-Wesley, Boston, MA, USA, 4th edition, 1992.

[40] Catalin Costin Stanciu, Joan Pastor-Pellicer, and Salvador España Boquera. Catalyde: herramienta para enseñar programación. In *V Jornada de Innovación Docente*, Valencia, Spain, December 2012. URL http://jidinf12.webs.upv.es/poster. (Citado en la pagina 73.)

[41] Joan Surrell, Immaculada Boada Oliveras, Josep Soler Masó, Ferran Prados Carrasco, and Jordi Poch Garcia. Corrección automática de ejercicios de estructuras de datos a través de una plataforma de e-learning. *Jornadas de Enseñanza Universitaria de la Informática (17es: 2011: Sevilla)*, 2011.

[42] Nghi Truong, Peter Bancroft, and Paul Roe. Learning to program through the web. *SIGCSE Bull.*, 37(3):9–13, June 2005. ISSN 0097-8418. doi: 10.1145/1151954.1067452. URL http://doi.acm.org/10.1145/1151954.1067452.

[43] Uriel Hdz. Especificación de requisitos de software. http://es.wikipedia.org/wiki/Especificaci%C3%B3n_de_requisitos_de_software, 2011. [Online; accedido 20-01-2013].

[44] Elena Verdu, Luisa M. Regueras, Maria J. Verdu, Jose P. Leal, Juan P. de Castro, and Ricardo Queiros. A distributed system for learning programming on-line. *Computers and Education*, 58 (1):1 – 10, 2012. ISSN 0360-1315. doi: 10.1016/j.compedu.2011.08.015. URL http://www.sciencedirect.com/science/article/pii/S036013151100193X.

[45] Dieter Vogts, André Calitz, and Jean Greyling. Comparison of the effects of professional and pedagogical program development environments on novice programmers. In *Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries: riding the wave of technology*, SAICSIT '08, pages 286–095, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-286-3.