UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Interfaz web para un simulador del MIPS R2000

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Nieves Codoñer Gil

Tutores: Antonio Martí Campoy
Salvador Vicente Petit Martí

Curso 2020-2021

Agradecimientos

Me gustaría expresar mi más sincero agradecimiento a mis padres y hermanos por el apoyo incondicional recibido durante mi etapa académica. A mi pareja por animarme y confiar en mi cuando a veces ni yo misma lo hacía.

A mis tutores Antonio Martí Campoy y Salvador Vicente Petit Martí por el constante apoyo y conocimientos que me han brindado durante estos meses, sin vuestra dedicación y paciencia este TFG no habría sido posible.

A mis amigos por permanecer a mi lado y tener siempre palabras de ánimo en los momentos más difíciles.

A la universidad por ofrecerme estos años de aprendizaje y experiencias que sin duda han marcado una etapa muy importante en mi vida.

Resum

El propòsit d'aquest Treball de Fi de Grau és dissenyar i implementar una interfície web per a un simulador de l'arquitectura R2000, acompanyada d'una interfície de programació d'aplicacions (API) per a estendre la seua funcionalitat, totes dues amb un propòsit final d'ús docent.

Amb referència al desenvolupament de l'aplicació, es parteix del simulador spim de consola escrit en el llenguatge de programació C++ i que per mitjà de la tecnologia Emscripten es compilarà al llenguatge de programació Javascript, per a més tard integrar-lo en l'aplicació web a través de l'ús d'un worker. Aquest worker es comunica per mitjà d'una interfície gràfica desenvolupada amb React i Material UI, que farà peticions a la API creada amb Node.js i Express.js i guardarà la informació en la base de dades de MongoDB.

Com a resultat l'aplicació web permet a l'usuari registrar-se, crear els seus propis programes a través d'un editor de text, poder emmagatzemar-los, gestionar-los, assemblar-los i executar-los per mitjà de l'ús d'un simulador.

Paraules clau: spim, simulador, aplicació web, worker, Emscripten, React, Express, MongoDB

Resumen

El propósito de este Trabajo de Fin de Grado es diseñar e implementar una interfaz web para un simulador de la arquitectura R2000, acompañada de una interfaz de programación de aplicaciones (API) para extender su funcionalidad, ambas con un propósito final de uso docente.

Con referencia al desarrollo de la aplicación, se parte del simulador spim de consola escrito en el lenguaje de programación C++ y que por medio de la tecnología Emscripten se compilará al lenguaje de programación JavaScript, para más tarde integrarlo en la aplicación web a través del uso de un worker. Dicho worker se comunica por medio de una interfaz gráfica desarrollada con React y Material UI, que hará peticiones a la API creada con Node.js y Express.js y guardará la información en la base de datos de MongoDB.

Como resultado la aplicación web permite al usuario registrarse, crear sus propios programas a través de un editor de texto, poder almacenarlos, gestionarlos, ensamblarlos y ejecutarlos por medio del uso de un simulador.

Palabras clave: spim, simulador, aplicación web, worker, Emscripten, React, Express, MongoDB

Abstract

The purpose of this Final Degree Project is to design and implement a web interface for an R2000 architecture simulator, accompanied by an application programming interface (API) to extend its function, both with a final purpose for teaching use.

With reference to the development of the application, it starts with the console spim simulator written in the C ++ programming language and that through Emscripten technology will be compiled into the JavaScript programming language, to later integrate it into the web application through of the use of a worker. Said worker communicates through a graphical interface developed with React and Material UI, which will make

requests to the API created with Node.js and Express.js and will save the information in the MongoDB database.

As a result, the web application allows the user to register, create their own programs through a text editor, and be able to store, manage, assemble and execute them through the use of a simulator.

Key words: spim, simulator, web app, worker, Emscripten, React, Express, MongoDB

Índice general

Índice de figuras

Índice de tablas

Glosario

- **Servidor:** Equipo cuya función es procesar solicitudes de clientes y devolver una respuesta en concordancia.
- **Tomcat:** Contenedor de *servlets* usado comúnmente para compilar y ejecutar aplicaciones que se han desarrollado en el lenguaje de programación Java.
- **Servlet:** Programa escrito en el lenguaje de programación Java que construye o sirve páginas web.
- **Software:** Se trata de un conjunto de componentes lógicos que ejecutados en un computador ofrecen servicios a los usuarios.
- **UML:** Lenguaje unificado de modelado (*Unified Modeling Language*).
- **VPN:** Red privada virtual (*Virtual Private Network*).
- **JSON:** Notación de objeto de JavaScript (*JavaScript Object Notation*)para facilitar el intercambio de datos.
- **Backend:** Parte del desarrollo web donde se incluye la base de datos y el servidor. Esta se encarga de procesar la entrada desde el *frontend* y en donde los usuarios no tienen contacto.
- **Frontend:** Parte del desarrollo web donde se incluye la parte del software que interactúa con los usuarios. Esta se encarga de enviar solicitudes al backend y recibir datos de los usuarios.
- **Framework:** Entorno de trabajo que sirve de base para la organización y desarrollo del software, facilitando de esta manera la programación.
- **Wireframe:** Referente a los prototipos que se crean en el diseño de una página web, siendo estos una guía visual que representa el esqueleto que se espera que tenga una web.
- **Git:** Software de control de versiones, para el mantenimiento de las diferentes versiones de una aplicación.
- **SPA:** Aplicación de página única (*Single-page application*).
- **Embeber:** Insertar código de un lenguaje dentro de otro.
- **Compilar:** Transformar un lenguaje de alto nivel a lenguaje máquina.
- **Transcompilador:** Compilador de código fuente que convierte código fuente de un programa a otro lenguaje.

- **Transpilación:** Generar código de un lenguaje de programación a otro lenguaje de programación pero que tiene un nivel similar de abstracción.
- **C:** Lenguaje de programación imperativo y fuertemente tipado de propósito general.
- **C++:** Extensión del lenguaje C que permite el uso de objetos.
- **JS:** Abreviatura utilizada para nombrar el lenguaje de programación JavaScript.
- **HTML:** Lenguaje de Marcas de Hipertexto (*HyperText Markup Language*).
- **HTTP:** Protocolo de transferencia de hipertexto (Hypertext Transfer Protocol) que permite la transferencia de archivos en la web.
- **CSS:** Hojas de estilo en cascada (*Cascading Sytle Sheets*).
- **WebAssembly:** Formato de código binario portable. Es utilizado para la ejecución en el navegador de secuencias de comandos en el lado del cliente.
- **Encriptar:** Se dice del proceso de ocultar datos mediante una clave.
- **Dominio:** Nombre único que identifica y se le asigna a un conjunto de computadores pertenecientes a la misma intranet.
- **Cookie:** Bloques pequeños de datos creados por un servidor web y que pasan al navegador web del usuario cuando visita estos sitios.
- **Makefile:** Archivo que se utiliza para compilar código, se localiza en la carpeta raíz del programa que se desea compilar y es interpretado por el comando “make".
- **URI:** Identificador de Recursos Uniforme (Uniform Resource Identifier).
- **Script:** Conjunto de ordenes o instrucciones.
- **Worker:** Secuencia de comandos que se ejecuta en un navegador en segundo plano y que no interfiere con la interfaz de usuario de la página web.
- **Cluster:** Colección de bases de datos que es administrada por una sola instancia de un servidor de base de datos.

CAPÍTULO 1
Introducción

El uso de simuladores en el proceso de enseñanza es una práctica habitual, esto es debido a que su empleo proporciona ventajas tanto al usuario como a la propia universidad que lo suministra. Estas ventajas se han visto avaladas a través de los años que estos han estado en activo. De esta manera, un simulador puede conseguir emular sistemas complejos o incluso costosos en el ámbito de su implementación física, suponiendo una posible disminución de carga en los recursos disponibles por parte de la entidad universitaria.

Por lo tanto, y contando con una interfaz simple, clara y coherente podemos suponer que el uso de un simulador facilitará y mejorará la calidad y el proceso de aprendizaje de aquellos alumnos que se enfrenten por primera vez a una nueva tecnología o arquitectura, en este caso con la arquitectura MIPS R2000.

Hay que tener presente que la universidad cuenta con un simulador web que fue desarrollado en el año 2008 como proyecto final de carrera: WebSpim [1], el inconveniente de este trabajo es que utiliza un tipo de servidor Tomcat que está obsoleto y por tanto el departamento DISCA, donde está alojado, ya no le puede proporcionar soporte. Además, se desea trasladar la carga computacional al lado del cliente, reduciendo los recursos necesarios en el servidor, e integrarlo en un sistema de gestión de usuarios y de archivos.

El objetivo de este trabajo consiste en adaptar el simulador spim de consola [2] a través de la compilación de este a JavaScript mediante la tecnología Emscripten, para poder embeber el simulador en la interfaz web. Esta contará con las anteriores características planteadas además de funcionales tales como un editor de código y (o) un sistema de autenticación y de almacenamiento de archivos para los usuarios registrados. Todo esto ayudará a que su aprendizaje de la arquitectura MIPS R2000 sea dinámico y sencillo.

1.1 Motivación

1.1.1. Motivación del trabajo

Actualmente, la asignatura Fundamentos de Computadores cuenta con un simulador web llamado WebSpim. Este fue creado en 2008 como parte de un trabajo de fin de carrera y, aunque en un principio ofrecía un sistema completo y un mantenimiento asequible, conforme fueron pasando los años y debido al avance en el desarrollo web, fue poco a poco quedándose obsoleto. Por otro lado, el servidor Tomcat que lo soportaba acabó desfasado, suponiendo una carga costosa para la universidad y ofreciendo a los alumnos una interfaz antigua y un acceso tedioso al simulador. Es obvio que esta situación no es favorable para ninguna de las partes,y por ese motivo se ofertó este proyecto, el cual

buscaría un remplazo para el WebSpim y de este modo solventar todos los problemas que está causando, ofreciendo al alumno una herramienta alternativa donde poder seguir su proceso de aprendizaje y ahorrar recursos a la universidad.

1.1.2. Motivación personal

La razón por la cual elegí realizar este trabajo de fin de grado fue mi motivación por diversos factores que este incluía.

Uno de ellos fue el tema que subyacía a este, ya que siempre me ha llamado mucho la atención conseguir mejorar el proceso de aprendizaje de las personas y proveerlos de esta forma de herramientas que les ayuden en el ámbito de la docencia.

Por otro lado, mis últimos años de universidad han sido estudiando la rama de ingeniería de computadores, en la que aprendí sobre procesadores, adquiriendo de esta manera conocimientos sobre el funcionamiento de la arquitectura MIPS R2000 y de este modo ser capaz de abordar esa parte del proyecto de forma efectiva.

Asimismo, aprender tecnologías web era algo que nunca había tenido la oportunidad de afrontar y me suponía además de un reto, una posible salida profesional a través de tecnologías en auge que introduciría en el desarrollo del proyecto.

Para concluir y como resumen de este punto, la motivación del proyecto se basa en una elección tanto personal como profesional.

1.2 Objetivos

El objetivo principal de este proyecto es conseguir adaptar el simulador spim y crear una interfaz web que sea utilizada para complementar el aprendizaje del procesador en las clases de teoría y prácticas de las asignaturas: Fundamentos de computadores y Estructura de computadores del GII. Para lograr este propósito se tendrán que llevar a cabo los siguientes puntos:

- Migrar el simulador spim a una tecnología que permita su integración sencilla en un sitio web.
- Diseñar una página web con una distribución similar a la utilizada en las interfaces gráficas del simulador spim, para facilitar al usuario el cambio de una versión a la desarrollada en este trabajo.
- Implementar una interfaz responsiva.
- Implementar un editor de texto que permita al usuario poder escribir sus programas en lenguaje ensamblador y ensamblar y ejecutar el contenido de este en el simulador.
- Implementar un sistema de autenticación y registro de los alumnos.
- Implementar un sistema de ficheros, donde cada alumno pueda subir sus propios programas en un espacio dedicado a ello.

1.2.1. Impacto esperado

La implementación de estos objetivos supondrá un impacto para los usuarios, dado que contarán con una interfaz web actualizada y un sistema de identificación para

acceder a sus programas que han guardado con anterioridad. Asimismo, ayudará al departamento DISCA a prescindir del servidor de Tomcat que utiliza Webspim, evitando el coste que le supondría a la universidad su integración y mantenimiento.

1.3 Metodología

Para el desarrollo del proyecto se ha optado por un modelo en cascada. Este modelo de desarrollo de software se concibe como un conjunto de etapas: análisis, diseño, implementación, pruebas y despliegue. Estas se ejecutan una tras otra siguiendo de esta manera un flujo de ejecución de arriba hacia abajo, como si se tratase de una cascada. Al final de cada etapa se lleva a cabo una revisión final, que se encarga de indicar si el proyecto puede progresar a la siguiente fase. En la figura 1.1 podemos observar las diferentes etapas con las que cuenta la metodología en cascada.

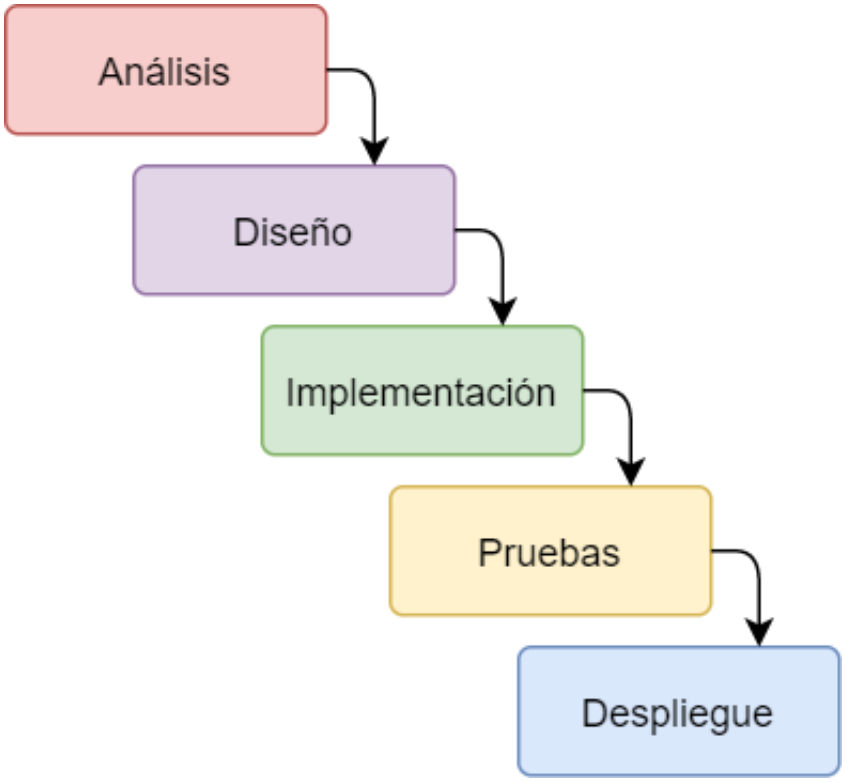

Figura 1.1: Etapas del desarrollo en cascada

Aunque esta metodología cuenta con desventajas como pueden ser el aumento de coste por cualquier error de diseño o que no se pueda pasar de etapa hasta que se haya acabado con la etapa anterior, se ha elegido llevar a la práctica esta metodología ya que el proyecto cuenta con unos requisitos claros y bien definidos, con lo que esta metodología es adecuada para el mismo. Además es un modelo conocido, fácil de implementar y de entender.

1.4 Plan de trabajo

En el presente apartado se va a presentar el plan de trabajo previsto que se va a seguir en la realización de este Trabajo de fin de grado.

En primer lugar, se ha procedido a evaluar las horas que serán dedicadas al día. De esta forma, se ha contando con 8 horas diarias, divididas en un total de 45 días que harán un total 9 semanas. Todas estas horas originarán una suma de 360 horas dedicadas al proyecto, que se dividirán entre las diferentes tareas con las que cuenta el trabajo. La realización del trabajo de fin de grado estaba planteado para unas 320 horas aproximadamente, pero se han añadido 30 horas más para el aprendizaje de ciertas tecnologías web debido a que se proviene de la rama de ingeniería de computadores.

Para que la explicación sea clara se ha hecho uso de un diagrama de Gantt[1]. En la figura 1.2 se puede observar el tiempo previsto para las fases de desarrollo del proyecto.

		Semanas								
Tareas	Duración	1	2	3	4	5	6	7	8	9
Análisis	5 días									
Diseño	5 días									
Implementación	22 días									
Pruebas	8 días									
Despliegue	5 días									

Figura 1.2: Diagrama de Gantt

1.5 Estructura de la memoria

Esta memoria está dividida en diferentes capítulos, cada uno de ellos estará subdividido en apartados y abordará distintas fases del proyecto que se ha desarrollado. De este modo, la memoria está estructurada de la siguiente manera:

- **Capítulo 1:** Se introduce el tema del trabajo desarrollado, se exponen las motivaciones por las que se ha elegido, los objetivos, el impacto esperado que supondrá para los usuarios, la metodología que se ha seguido para realizar el proyecto y se crea un plan de trabajo el cual seguiremos a lo largo del desarrollo.
- **Capítulo 2:** Se resume la situación actual de las aplicaciones con propósito similar que existen en el mercado.
- **Capítulo 3:** Se analiza detalladamente el problema, especificando para ello los posibles requisitos del proyecto por medio del estándar IEEE 830/1998.
- **Capítulo 4:** Se describen los casos de uso y se expone el modelado de los actores de la aplicación.

[1]El diagrama de Gantt es una herramienta gráfica que tiene como objetivo exponer el tiempo de dedicación que se prevé para un conjunto de actividades a lo largo de un tiempo que se ha determinado previamente.

- **Capítulo 5:** Se detalla el diseño previo de la estructura del proyecto: diseño arquitectónico, diseño de base de datos y diseño de las interfaces.
- **Capítulo 6:** Se detallan las tecnologías y herramientas utilizadas en el desarrollo y se explica el por qué se han elegido.
- **Capítulo 7:** Se detalla el proceso de desarrollo de la propuesta, es decir, las decisiones tomadas y los problemas y dificultades que han surgido.
- **Capítulo 8:** Se llevan a cabo las pruebas necesarias para comprobar que el sistema realiza lo que se ha especificado en la etapa de requisitos.
- **Capítulo 9:** Se lleva a cabo la implantación de la solución final y se detalla su proceso.
- **Capítulo 10:** Se muestra el presupuesto que se ha estimado para el desarrollo y explotación del proyecto.
- **Capítulo 11:** Se presentan las conclusiones obtenidas tras el desarrollo software del proyecto.
- **Capítulo 12 :** Se comentan propuestas de mejoras del trabajo y posibles trabajos futuros que por falta de tiempo no se hayan podido implementar.
- **Capítulo 13 :** Se exponen una serie de asignaturas relacionadas con el trabajo y los conocimientos que estas nos han ofrecido para su desarrollo.
- **Capítulo 14 :** Se listan una serie de referencias bibliográficas consultadas en la realización del trabajo así como del aprendizaje de herramientas que se hacen uso en el proyecto.

Cabe destacar que para facilitar la comodidad del lector, se ha optado por ubicar la sección de términos y acrónimos en un capitulo de introducción. Este capitulo cuenta con una serie de términos de carácter técnico y su respectiva definición, ayudando al lector a familiarizarse con términos que se irán encontrando a lo largo del documento.

CAPÍTULO 2
Estado del arte

En el presente capitulo se van a documentar otras aplicaciones que se utilizan para simular procesadores MIPS R2000. Todas ellas están basadas o son parte del trabajo de James R. Larus, creador del simulador autónomo SPIM [3]. Este, ejecuta programas en ensamblador escritos para el procesador MIPS32 y cuenta con documentación, código fuente y varias versiones gráficas y de consola, algunas de las cuales expondremos a continuación.

2.1 PCSpim/XSpim

Se trata de un simulador que cuenta con una versión gráfica que ejecuta programas en lenguaje ensamblador, concretamente computadores basados en procesadores MIPS R2000/R3000. Esta aplicación fue creada por James R. Larus y es soportada tanto por el sistema operativo Windows (PCSpim) como el sistema Linux (XSpim). La arquitectura utilizada en este proyecto está perfectamente descrita en [4].

Como podemos ver en las figuras 2.1 y 2.2, cuentan con un apartado de registros, otro de segmento de texto, de datos y una consola donde se da información sobre el spim.

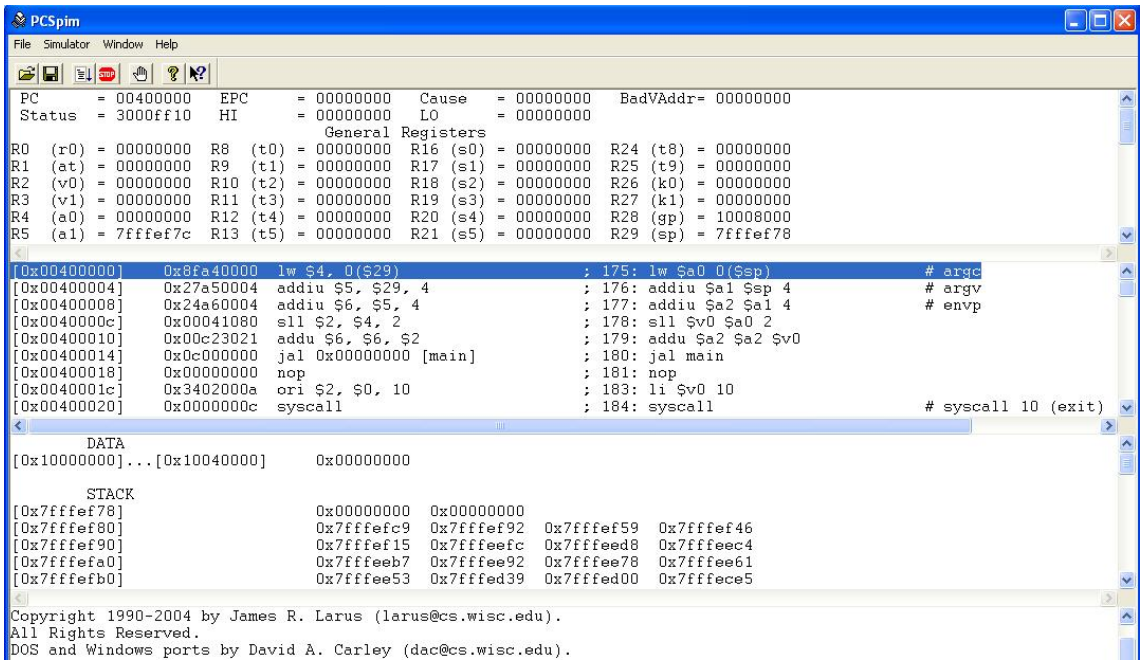

Figura 2.1: PCSpim, versión Windows

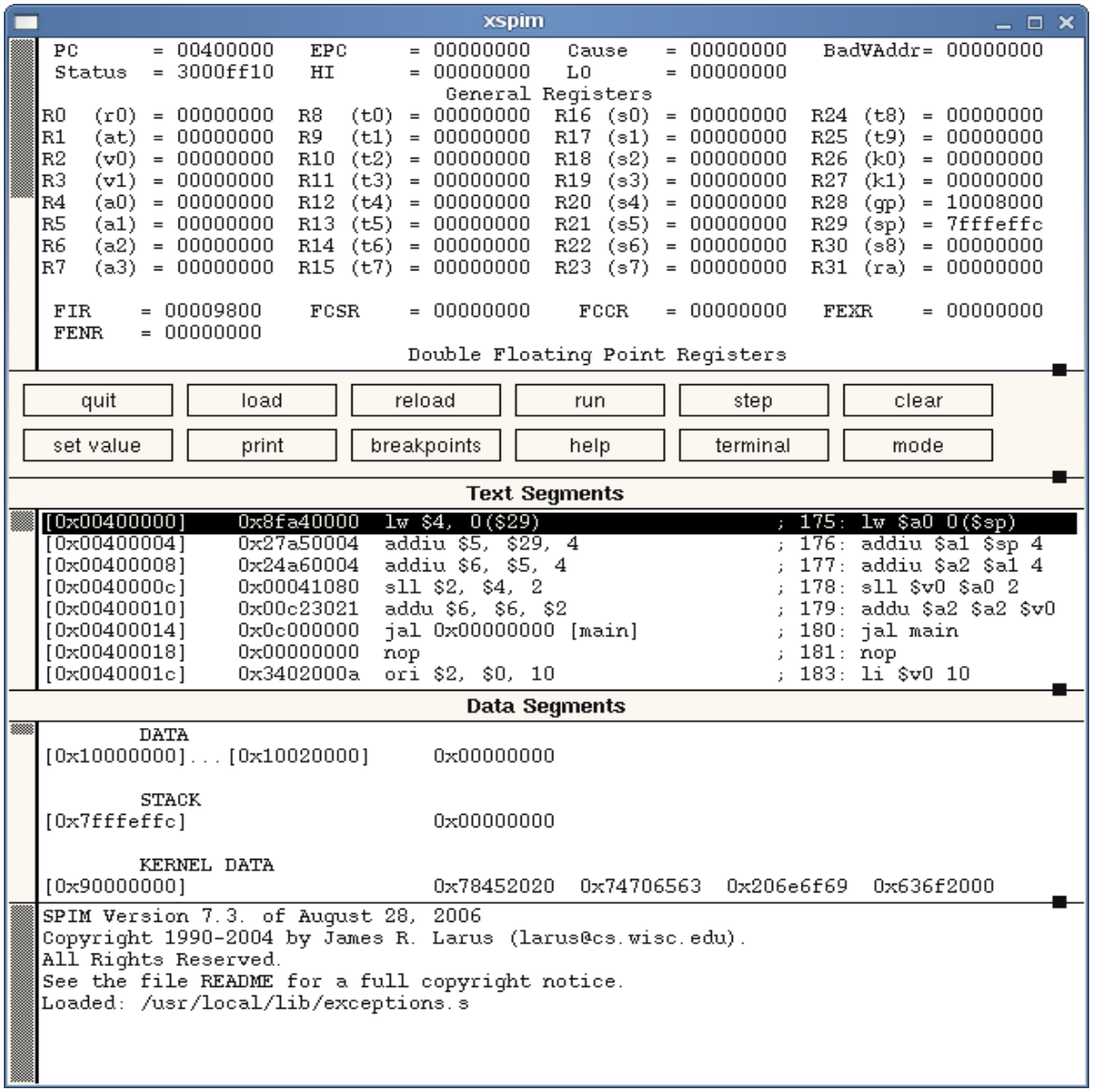

Figura 2.2: XSpim, versión Linux

Como hemos podido observar en las imágenes anteriores, tanto la interfaz de Windows como la de Linux son muy similares, siendo estas en las que se basará nuestro proyecto. Así pues, se diseñará una interfaz parecida a las anteriormente citadas para que los alumnos o usuarios que hayan usado estas puedan adaptarse fácilmente a la nueva interfaz, y suavizar la curva de aprendizaje. Además existe también una interfaz de consola (figura 2.3) que será la versión que utilizaremos para el desarrollo de nuestro proyecto.

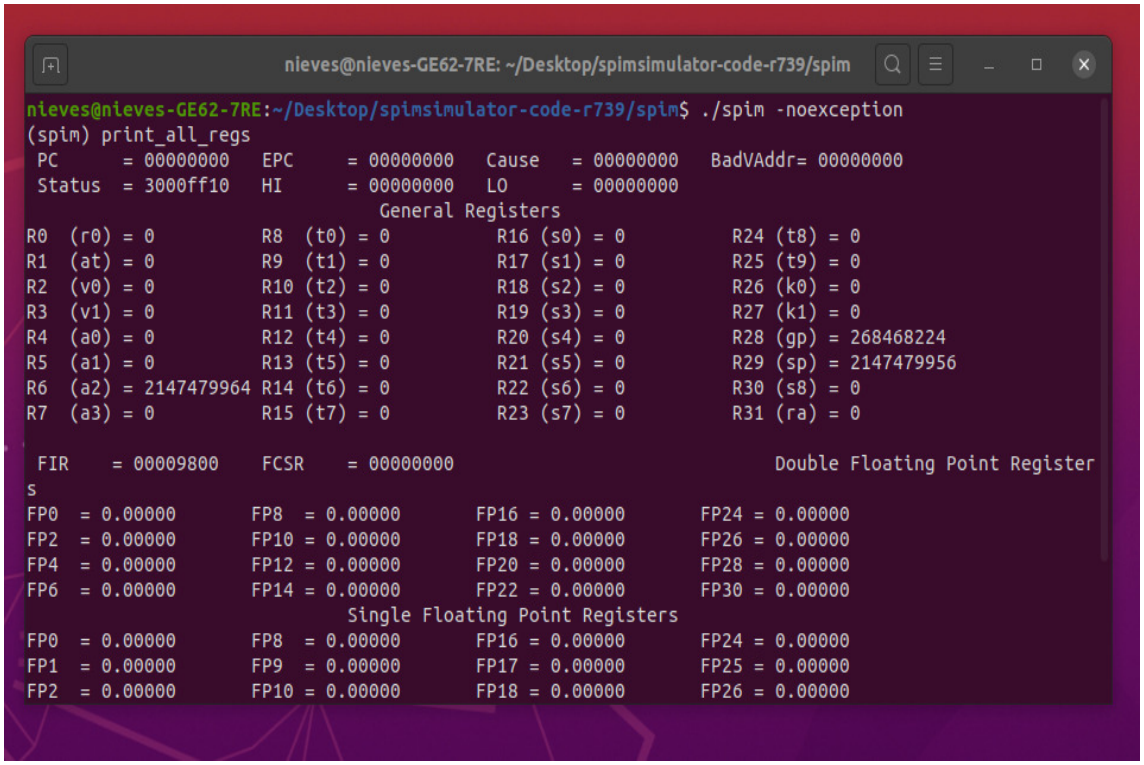

Figura 2.3: Spim con interfaz de consola

2.2 QtSpim

Se trata de la versión más actualizada de Spim. Es una aplicación desarrollada con QT [5], entorno de desarrollo multiplataforma con interfaz gráfica de usuario, por lo que es soportada por los sistemas operativos Windows, Mac y Linux, contando con el mismo código fuente y la misma interfaz en las tres plataformas.

Como se puede observar en la figura 2.4, cuenta con una interfaz clara y coherente donde los segmentos de texto, datos y los diferentes registros están distribuidos por medio de pestañas. Además, cuenta con un panel de mensajes donde se da información de QtSpim [6] al usuario.

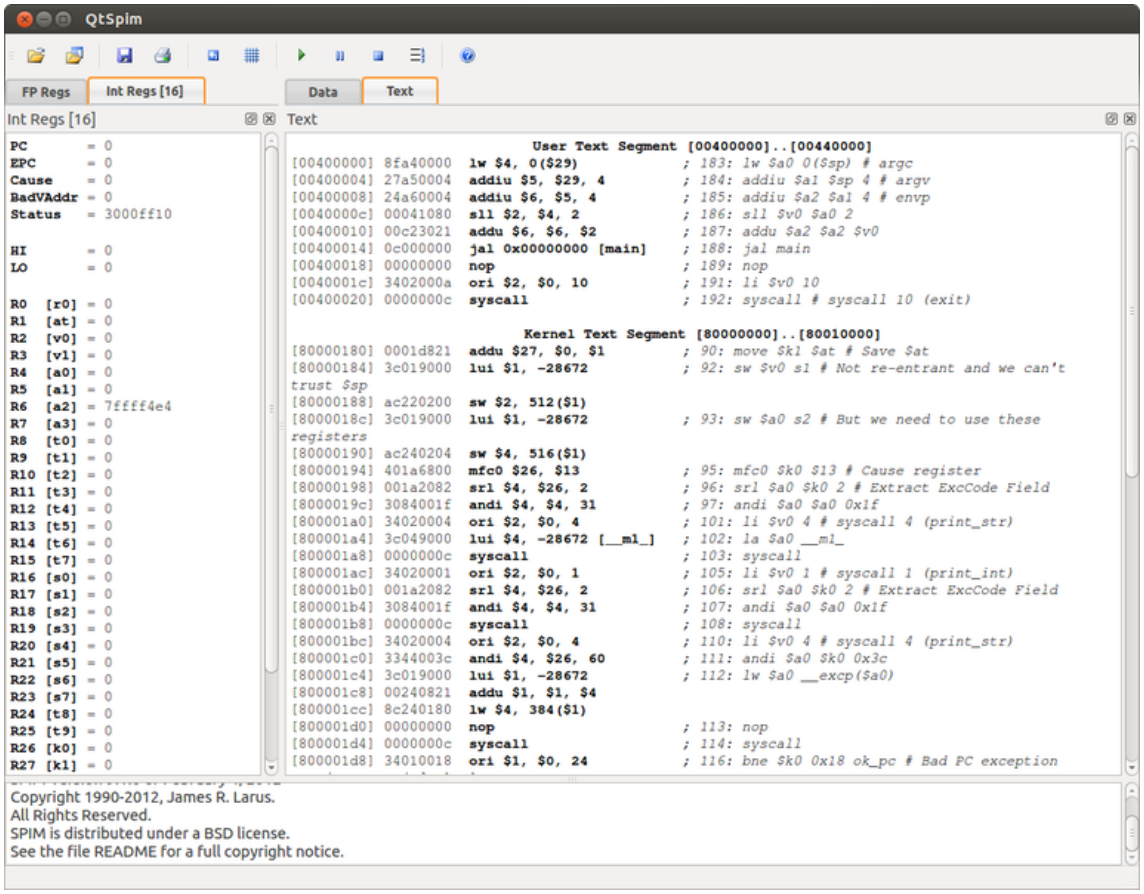

Figura 2.4: QtSpim

Por otro lado, cuenta con todas las funcionalidades básicas del spim original 2.1. Estas están organizadas en el menú superior del programa, a través de una serie de iconos que hacen referencia a los diferentes comportamientos: guardar, ejecutar, ensamblar, continuar, ejecución paso a paso ...

Aunque la interfaz está bien estructurada, se ha decidido no optar por ella, ya que principalmente no cuenta con un editor de texto para que el usuario pueda escribir los programas y ejecutarlos directamente, que es un requisito importante de este trabajo.

2.3 WebSpim

Se trata de una versión creada como parte del trabajo final de carrera de Joan Fernández Pérez, siendo los directores del proyecto Salvador Petit y José Luis Poza. WebSpim [1] tiene una interfaz gráfica muy similar a la del PCSpim, ya que está basada en ella y cuenta con una funcionalidad casi idéntica a la que nos ofrece la primera versión, con la salvedad de que no es una aplicación de escritorio, sino que está disponible como aplicación web que se ejecuta en el servidor, requiriendo más potencia del lado del servidor.

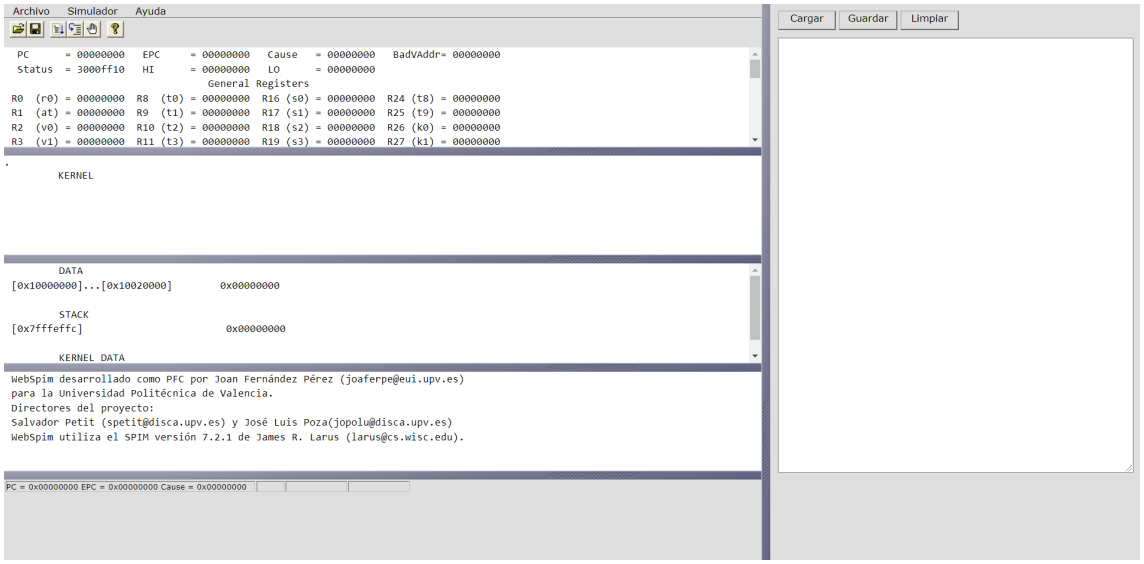

Figura 2.5: Webspim

Asimismo, WebSpim permite al usuario además de las funcionalidades básicas del Spim, tener un editor de texto donde cargar los programas.

El inconveniente principal que tiene esta versión es que utiliza un servidor Tomcat que en estos momentos está obsoleto. Es por ello por lo que nace la razón de ser de este TFG, ya que el departamento no le puede dar mantenimiento. Otra desventaja es que su ejecución se realiza en el lado del servidor, por lo que no es escalable.

2.4 JsSpim

JsSpim es un simulador online MIPS32 basado en el Spim de James Larus [7]. Este proyecto parte de la misma base que el nuestro, debido a que utiliza la herramienta Emscripten (que más tarde se profundizará en detalle) para transformar el archivo original escrito en C++ a un archivo en el lenguaje de programación JavaScript. Además, entre sus características podemos encontrar:

- Control de la velocidad de ejecución del programa por medio de un deslizador.
- Resaltado en los registros, segmentos de datos y las pilas que cambian. También cuenta con un resaltador de la instrucción en la que se encuentra en el momento en el que el simulador está ejecutando.
- Posibilidad de crear *breakpoints*(puntos de ruptura) a través de clic en las instrucciones.
- Interfaz similar a la nueva versión QTSpim.

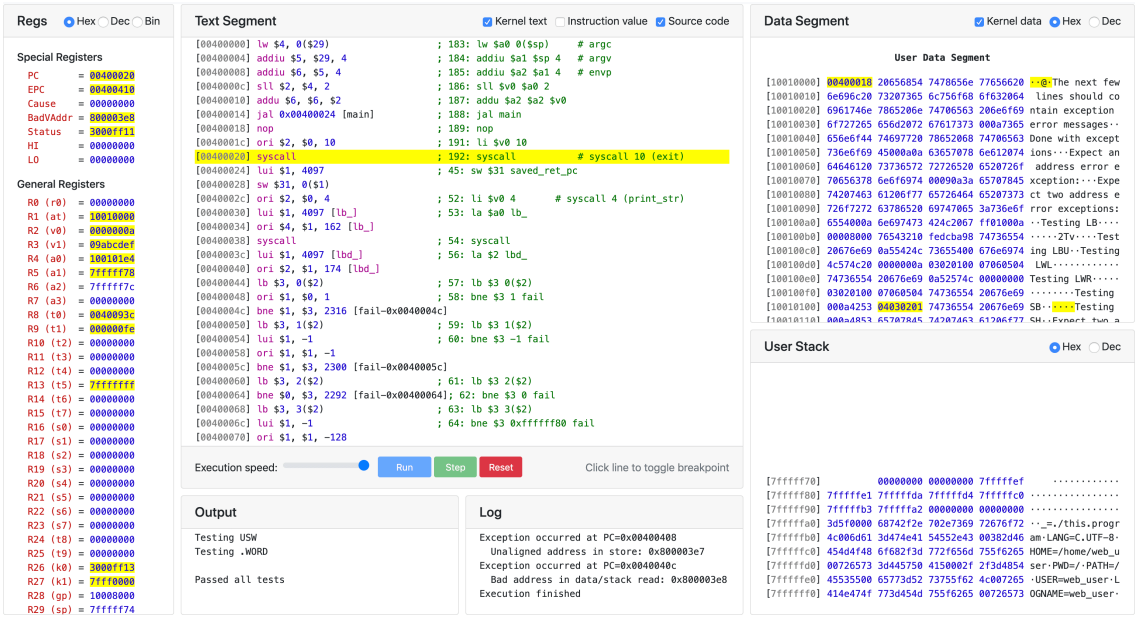

Figura 2.6: JsSpim

Cabe destacar que, aunque haya muchas similitudes respecto al proyecto actual (figura 2.6), también existen muchas diferencias que hacen que el desarrollo de este proyecto sea necesario. Estas diferencias estarán planteadas en el siguiente apartado.

2.5 Propuesta

Como ya ha sido descrito en los apartados anteriores, estas aplicaciones cuentan con buenas características para el uso docente, pero cabría modificar y añadir una serie de propiedades para adecuarlas al trabajo que las asignaturas requieren.

De esta manera, podemos diferenciar el proyecto actual de los citados con anterioridad a través de una serie de puntos:

- Acceso al simulador mediante web.
- La necesidad de un editor de texto para que los alumnos puedan escribir programas en ensamblador, y que cuente además con un resaltado para que sea más sencilla su escritura.
- Sistema de gestión de archivos: que permita descargar, subir programas en ensamblador, visualizarlos y eliminarlos.
- Sistema de registro de alumnos.
- Distribución de la interfaz similar a la del proyecto PCSpim y Xspim, que son las aplicaciones utilizadas durante los años anteriores por los alumnos y profesores.
- La carga computacional debe desplazarse al lado del cliente.

CAPÍTULO 3
Especificación de requisitos

3.1 Introducción

En la presente sección se van a especificar los requerimientos siguiendo el estándar IEEE 830/1998 [8].

3.1.1. Propósito

El propósito de realizar un documento de Especificación de Requisitos Software (ERS) es conseguir explicar las diferentes funcionalidades que ofrecerá esta aplicación web, de forma que cualquier persona que se pueda interesar por el proyecto pueda comprenderlo sin complicaciones.

3.1.2. Ámbito del sistema

La finalidad de este proyecto es desarrollar una aplicación web para las asignaturas Fundamentos de Computadores y Estructura de Computadores y de esta forma sustituir el proyecto WebSpim.

El alumno que acceda a este sistema podrá registrarse con: su nombre, un correo electrónico, una contraseña válida, el curso en el que se encuentra y la asignatura que está cursando. Una vez se haya registrado con éxito podrá iniciar sesión en la web, a través de los campos correo electrónico y contraseña. Si el usuario no se confunde e introduce los datos correctamente, pasará a la página del simulador, que cuenta con las funciones básicas que nos ofrece el spim de consola (que serán nombradas posteriormente en el apartado 3.2.2 funciones del producto). Asimismo, cuenta con funcionalidades extras como pueden ser el editor código y la posibilidad de almacenar y descargar los programas en disco o en el servidor. También podrá acceder a la web cualquier alumno no registrado, a través de la opción de acceder como invitado, que le ofrecerá todas las características descritas anteriormente pero sin la posibilidad de las funcionalidades que ofrece el servidor.

Por consiguiente, el principal beneficio que brinda este sistema es la oportunidad de que el alumnado pueda utilizar un entorno de aprendizaje sencillo, en el que no sea un requisito acceder a ninguna red privada virtual de la universidad y pueda disponer de una interfaz actualizada, renovando así la visión que tienen los usuarios sobre este tipo de simuladores.

3.2 Descripción General

3.2.1. Perspectiva del Producto

Este producto estará relacionado con el spim de consola que anteriormente fue mencionado y que se puede descargar en la página Sourceforge [2]. Este estará *transcompilado* por medio de Emscripten y añadido al proyecto en forma de *worker*. Por otro lado, el funcionamiento de esta aplicación web estará soportado por cualquier navegador web moderno, pudiendo de esta manera utilizarlo en diferentes plataformas como tabletas, ordenadores o portátiles.

3.2.2. Funciones del Producto

A grandes rasgos, el sistema deberá proporcionar soporte a las siguientes actividades:

- Registrar usuarios en la plataforma: cualquier usuario que acceda a la plataforma podrá registrarse en la misma.
- Iniciar sesión: aquellos usuarios que se hayan registrado previamente podrán iniciar sesión con las credenciales que habían establecido.
- Cerrar sesión: el usuario podrá *desloguearse* de la aplicación.
- Ensamblar el programa en el simulador: el usuario podrá ensamblar cualquier programa que haya escrito en el simulador. Si hay algún error en el código el sistema lo mostrará por pantalla.
- Ejecutar el programa completo: el usuario podrá ejecutar un programa completo.
- Ejecutar el programa por pasos: el usuario podrá ir ejecutando el programa instrucción a instrucción o elegir el número de instrucciones que quiere ejecutar.
- Establecer puntos de ruptura: el usuario podrá establecer *breakpoints* en la dirección que crea conveniente del código.
- Eliminar puntos de ruptura: el usuario podrá eliminar *breakpoints* que haya establecido previamente.
- Continuar con la ejecución: el usuario podrá continuar con la ejecución, hasta el final o, si hay algún punto de ruptura, hasta este.
- Recargar programa: el usuario puede volver a ensamblar el programa en el simulador.
- Modificar el código ensamblador en el editor: el usuario podrá tanto eliminar contenido como añadirlo.
- Subir programas desde disco y desde el servidor: el usuario puede subir programas de su sistema de archivos. El usuario registrado también podrá subirlo desde su carpeta de archivos del servidor.
- Guardar programas en disco: el usuario puede descargar el programa del editor de código a su sistema de ficheros.
- Guardar programas en el servidor: el usuario registrado podrá almacenar sus programas en el servidor.

3.2.3. Características de los Usuarios

Esta aplicación web irá destinada a aquellos alumnos de primero y segundo que estén matriculados en el Grado en Ingeniería Informática de la Universitat Politècnica de València y que cursen las asignaturas Fundamentos de Computadores y (o) Estructura de Computadores. Cabe destacar, que este simulador será creado para el refuerzo de las clases de teoría, con lo cual los alumnos ya tendrán un conocimiento previo del lenguaje ensamblador y la arquitectura MIPS R2000.

Así pues, los usuarios que la utilizarán contarán con estudios superiores (ya sea bachillerato o ciclos superiores) o serán personas mayores de 25 años que han accedido al grado anteriormente citado. Además, es preciso señalar que estos tendrán una experiencia informática mayor, y, habrán alumnos que hayan utilizado otros simuladores tales como WebSpim o PCSpim.

3.2.4. Restricciones

En cuanto a las restricciones, cabe destacar que solo existe una limitación y esta estará presente durante el desarrollo del proyecto. La interfaz debe de ser similar a la que ofrece PCSpim o WebSpim, ya que existe alumnado que ha trabajado con anterioridad con estas aplicaciones y repercutirá en un aprendizaje más rápido, siendo uno de los objetivos principales que busca este trabajo de fin de grado.

Con respecto las tecnologías utilizadas en el desarrollo, no existe ninguna restricción por lo que se podrá elegir cualquiera, eligiendo de esta forma la que mejor se adapte tanto a las características del proyecto como al desarrollador que lo implementa.

3.2.5. Suposiciones y Dependencias

Como ya se ha nombrado anteriormente, esta aplicación estará subida en un futuro en los servidores de la UPV, por ello será necesario que la entidad siga ofreciendo este sistema. Por otro lado, si no se dispusiera de este servidor, también podría desplegarse en otras plataformas tales como Heroku.

3.2.6. Requisitos Futuros

Si en un futuro se requiriera utilizar esta aplicación para que los alumnos pudieran entregar sus tareas en la plataforma, sería necesario tanto nuevos roles (como el de administrador utilizado por los profesores que imparten las asignaturas) como nuevas funcionalidades que faciliten este proceso y la validación de los usuarios con los alumnos matriculados en las asignaturas. Esta ampliación deberá hacerse con las mismas tecnologías utilizadas para el desarrollo del proyecto y priorizar siempre la facilidad de uso de la aplicación.

3.3 Requisitos Específicos

3.3.1. Funciones

1. En las siguientes tablas se muestran los requisitos para el registro de un usuario:

Número	1
Requisito	Introducir nombre.
Descripción	El sistema deberá proporcionar un campo de texto donde el usuario podrá introducir su nombre.

Tabla 3.1: Requisito funcional: Introducir nombre

Número	2
Requisito	Introducir correo
Descripción	El sistema deberá proporcionar un campo de texto donde el usuario podrá introducir su correo electrónico, con el cual se identificará en la aplicación.

Tabla 3.2: Requisito funcional: Introducir correo

Número	3
Requisito	Introducir contraseña.
Descripción	El sistema deberá proporcionar un campo de texto donde el usuario podrá introducir la contraseña que elija para su cuenta.

Tabla 3.3: Requisito funcional: Introducir contraseña.

Número	4
Requisito	Repetir contraseña.
Descripción	El sistema deberá proporcionar un campo de texto donde el usuario podrá volver a introducir su contraseña.

Tabla 3.4: Requisito funcional: Repetir contraseña.

Número	5
Requisito	Introducir asignatura matriculada.
Descripción	El sistema deberá proporcionar un campo de texto donde el usuario podrá introducir la asignatura en la cual está matriculado para utilizar el simulador.

Tabla 3.5: Requisito funcional: Introducir asignatura matriculada.

Número	6
Requisito	Introducir curso.
Descripción	El sistema deberá proporcionar un campo de texto donde el usuario podrá introducir el curso en el que está actualmente.

Tabla 3.6: Requisito funcional: Introducir curso.

Número	7
Requisito	Validar campos de registro
Descripción	El sistema validará que todos los campos cumplan una serie de características predefinidas cuando el usuario pulse el botón "Registrarse".

Tabla 3.7: Requisito funcional: Validar campos de registro

2. En las siguientes tablas se muestran los requisitos para el *login*:

Número	8
Requisito	Introducir correo electrónico.
Descripción	El sistema proporcionará un campo de texto donde el usuario podrá introducir su correo electrónico con el que está registrado.

Tabla 3.8: Requisito funcional: Introducir correo electrónico

Número	9
Requisito	Introducir contraseña
Descripción	El sistema proporcionará un campo de texto donde el usuario podrá introducir su contraseña.

Tabla 3.9: Requisito funcional: Introducir contraseña

Número	10
Requisito	Visualizar contraseña
Descripción	El sistema proporcionará un botón donde el usuario podrá visualizar la contraseña desprotegida.

Tabla 3.10: Requisito funcional: Visualizar contraseña

Número	11
Requisito	Validar campos de inicio de sesión
Descripción	El sistema buscará en la base de datos si el correo electrónico que ha introducido el usuario está en la base de datos y además si la contraseña coincide con la que está guardada en la base de datos.

Tabla 3.11: Requisito funcional: Validar campos de inicio de sesión

3. En las siguientes tablas se muestran los requisitos para el simulador:

Número	12
Requisito	Ensamblar programa
Descripción	El sistema proporcionará un botón donde el usuario podrá pulsar y cargará el programa en el simulador.

Tabla 3.12: Requisito funcional:Ensamblar programa

Número	13
Requisito	Ejecutar programa completo
Descripción	El sistema proporcionará un botón donde el usuario podrá pulsar "Ejecutar', mostrará un dialogo para elegir la dirección de memoria de inicio y se ejecutará el programa que se ha cargado previamente.

Tabla 3.13: Requisito funcional: Ejecutar programa completo

Número	14
Requisito	Ejecutar por pasos
Descripción	El sistema proporcionará un botón donde el usuario podrá pulsar "Pasos" y se ejecutará el programa que se ha cargado previamente. El usuario tendrá que elegir el número de pasos que desea, el sistema le mostrará otro botón para aceptar, cerrará la ventana y mostrará el programa en el paso que se ha indicado.

Tabla 3.14: Requisito funcional: Ejecutar por pasos

Número	15
Requisito	Continuar programa
Descripción	El sistema proporcionará un botón donde el usuario podrá pulsar "Continuar" y el simulador reanudará la ejecución del programa.

Tabla 3.15: Requisito funcional: Continuar programa

Número	16
Requisito	Recargar programa
Descripción	El sistema proporcionará un botón donde el usuario podrá pulsar "Recargar" y volverá a ensamblar el programa. Este botón debe estar junto a los botones que controlan la ejecución del programa para mejorar la usabilidad del simulador.

Tabla 3.16: Requisito funcional: Recargar programa

Número	17
Requisito	Establecer puntos de ruptura por botones
Descripción	El sistema deberá proporcionar botones donde el usuario pulsará para crear puntos de ruptura y se mostrará el botón relleno.

Tabla 3.17: Requisito funcional: Establecer puntos de ruptura por botones

Número	18
Requisito	Establecer puntos de ruptura por medio de menú
Descripción	El sistema deberá proporcionar un botón en el menú donde el usuario pulsará, el sistema mostrará una ventana con un campo de texto para añadir una dirección de memoria y un botón. El usuario pulsará y el sistema coloreará el botón, estableciendo un punto de ruptura en la dirección introducida.

Tabla 3.18: Requisito funcional: Establecer puntos de ruptura por medio de menú

Número	19
Requisito	Eliminar puntos de ruptura por botones
Descripción	El sistema deberá proporcionar botones donde el usuario pulsará alguno coloreado para eliminar puntos de ruptura y se mostrará el botón vacío.

Tabla 3.19: Requisito funcional: Eliminar puntos de ruptura por botones

Número	20
Requisito	Visualizar registros en hexadecimal
Descripción	El sistema deberá proporcionar una pestaña "HEX" donde el usuario podrá pulsar y la aplicación le mostrará un área de texto donde estarán escritos los registros del procesador en hexadecimal.

Tabla 3.20: Requisito funcional: Visualizar registros en hexadecimal

Número	21
Requisito	Visualizar registros en decimal
Descripción	El sistema deberá proporcionar una pestaña "DEC" donde el usuario podrá pulsar y la aplicación le mostrará un área de texto donde estarán escritos los registros del procesador en decimal.

Tabla 3.21: Requisito funcional: Visualizar registros en decimal

Número	22
Requisito	Visualizar el segmento de datos
Descripción	El sistema deberá proporcionar un área de texto que contenga el segmento de datos del simulador.

Tabla 3.22: Requisito funcional: Visualizar el segmento de datos

Número	23
Requisito	Visualizar segmento de código
Descripción	El sistema deberá proporcionar un área de texto que contenga el segmento de código del simulador.

Tabla 3.23: Requisito funcional: Visualizar el segmento de código

Número	24
Requisito	Visualizar diálogo de error
Descripción	El sistema deberá proporcionar un diálogo que contenga los errores que se puedan originar en el ensamblado del código.

Tabla 3.24: Requisito funcional: Visualizar dialogo de error

Número	25
Requisito	Visualizar consola
Descripción	El sistema deberá proporcionar un área de texto donde se muestre información sobre el ensamblaje de código en el simulador.

Tabla 3.25: Requisito funcional: Visualizar consola

4. En las siguientes tablas se muestran los requisitos para el editor de código:

Número	26
Requisito	Modificar contenido del editor de código
Descripción	El sistema deberá permitir borrar y añadir al usuario lineas en el editor de código.

Tabla 3.26: Requisito funcional: Modificar contenido del editor de código

Número	27
Requisito	Limpiar editor de código
Descripción	El sistema deberá proporcionar un botón "Limpiar" que permita al usuario borrar todo el contenido del editor de código.

Tabla 3.27: Requisito funcional: Limpiar editor de código

Número	28
Requisito	Cargar programa de disco en el editor de código
Descripción	El sistema deberá proporcionar un botón "Cargar" que permita al usuario cargar un programa de disco en el editor de código.

Tabla 3.28: Requisito funcional: Cargar programa de disco en el editor de código

Número	29
Requisito	Guardar programa del editor de código a disco
Descripción	El sistema deberá proporcionar un botón "Guardar" que permita al usuario guardar el contenido del editor de código en disco.

Tabla 3.29: Requisito funcional: Guardar programa del editor de código

Número	30
Requisito	Validar dirección de memoria
Descripción	El sistema deberá validar la dirección de memoria que introduce el usuario al ejecutar el programa o añadir un *breakpoint*.

Tabla 3.30: Requisito funcional: Validar dirección de memoria

5. En las siguientes tablas se muestran los requisitos para el sistema de archivos del servidor:

Número	31
Requisito	Cargar programa del servidor al editor de código
Descripción	El sistema deberá proporcionar un botón con un icono de *input* para permitir al usuario cargar el programa que ha seleccionado en el editor de código.

Tabla 3.31: Requisito funcional: Cargar programa del servidor al editor de código

Número	32
Requisito	Subir un programa al servidor desde disco
Descripción	El sistema deberá proporcionar dos botones: uno para elegir un programa en disco ("Seleccionar archivo") y otro para subir el programa al servidor ("Subir archivo").

Tabla 3.32: Requisito funcional: Subir programa al servidor

Número	33
Requisito	Subir el contenido del editor de código al servidor
Descripción	El sistema deberá proporcionar un botón "Subir" que muestre un dialogo en el que pueda elegir un nombre para el archivo y un botón para subir el programa escrito en el editor de código al servidor.

Tabla 3.33: Requisito funcional: Subir el contenido del editor de código al servidor

Número	34
Requisito	Eliminar programa del servidor
Descripción	El sistema deberá proporcionar un icono representado con una papelera que le permita al usuario eliminar un programa (de su propiedad) que se encuentre en el servidor.

Tabla 3.34: Requisito funcional: Eliminar programa del servidor

Número	35
Requisito	Visualizar programa del servidor
Descripción	El sistema deberá proporcionar un icono representado por una carpeta que le permita al usuario visualizar los programas que tiene subidos en el servidor.

Tabla 3.35: Requisito funcional: Visualizar programas del servidor

Número	36
Requisito	Validar tamaño del programa en el servidor
Descripción	El sistema deberá validar si el programa que se quiere subir al servidor no supera el tamaño indicado (1MB).

Tabla 3.36: Requisito funcional: Validar tamaño del programa en el servidor

Número	37
Requisito	Validar nombre del programa en el servidor
Descripción	El sistema deberá validar si no existe ya un programa con el mismo nombre relacionado con el usuario que intenta subir el programa.

Tabla 3.37: Requisito funcional: Validar nombre del programa en el servidor

6. En las siguientes tablas se muestran los requisitos para las opciones de cuenta:

Número	38
Requisito	Cerrar sesión
Descripción	El sistema deberá permitir al usuario cerrar su sesión.

Tabla 3.38: Requisito funcional: Cerrar sesión

Número	39
Requisito	Borrar cuenta
Descripción	El sistema deberá permitir al usuario borrar su cuenta.

Tabla 3.39: Requisito funcional: Borrar cuenta

Número	40
Requisito	Restablecer contraseña
Descripción	El sistema deberá permitir al usuario restablecer la contraseña de su cuenta.

Tabla 3.40: Requisito funcional: Restablecer contraseña

3.3.2. Requisitos de Rendimiento

En este apartado se van a exponer los diferentes requisitos en cuanto a rendimiento que el proyecto conllevará. De esta forma, se pueden encontrar:

- Requisitos por parte del servidor: La exigencia impuesta en cuanto al rendimiento del servidor será baja, esto es debido a que será el *worker* el que realizará el trabajo más pesado (ya que se encargará de la simulación) y este *worker* se ejecuta en el lado del cliente.
- Requisitos por parte de la base de datos: Se estima que habrá unos 800 alumnos por cada asignatura. De esta forma, si cada alumno genera 20 programas (hecho bastante improbable) se hablaría de un total de 16.000 archivos. Estos archivos son texto utf-8, no contienen ningún tipo de archivo multimedia, y los programas que se realizarán en estas asignaturas son sencillos, por lo que su tamaño máximo rondará los 10KB, lo que daría un total de 160.000 KB por cada año académico, lo que es insignificante para cualquier equipo moderno. Cabe destacar, que será importante que el administrador que tenga acceso a la base de datos elimine los archivos con cierta antigüedad. Este trabajo se facilitará a través de la incorporación de un nuevo campo “date“ en la colección de fichero que contiene la fecha de subida del archivo.

CAPÍTULO 4
Casos de uso

En este apartado se va a proceder a realizar un análisis de los casos de uso, siendo un apartado importante en el desarrollo de un proyecto software [9], y logrando así capturar las acciones que realizarán los actores del sistema. De esta manera, y por medio de los diagramas de casos de uso y su descripción en lenguaje natural, conseguiremos llevar a cabo una descripción detallada de aquellas actividades necesarias en la aplicación.

Con el objetivo de conseguir un diagrama que nos ayude en la visualización, especificación, construcción y documentación de nuestro sistema software, se hará uso del lenguaje unificado de modelado o UML [10] por sus siglas en inglés: *"Unified Modeling Language"*.

Así como ha sido visto en las asignaturas de la carrera tales como Ingeniería del software o Interfaces persona computador, los diagramas de casos de uso [11] facilitan la descripción de una acción o actividad, haciendo hincapié en la interacción que subyace entre el usuario y el sistema. Además, su uso es comúnmente utilizado para la captura de requisitos funcionales y ayuda a validar y verificar el sistema mientras se está desarrollando. Por consiguiente, pasaremos a describir los casos de uso que conforman los requisitos de la interfaz web solicitada, realizados a través de la aplicación draw.io [12].

4.1 Descripción casos de uso

- **Cargar programa de disco:** Cargar un programa del dispositivo donde el usuario está trabajando. La información detallada sobre este caso de uso se puede observar en la tabla 4.1:

Cargar programa de disco	
Actores	Visitante y Registrado.
Resumen	El usuario puede acceder a su sistema de archivos para cargar en el editor de texto su programa.
Precondición	El usuario tendrá el programa que desee cargar en su sistema.
Postcondición	El usuario cargará el programa en el editor de texto y lo podrá visualizar.
Flujo de eventos	
Actor	*Sistema*
1. El usuario *clica* en la opción de "Subir Fichero".	2. El sistema muestra el explorador de archivos del usuario.
3. El usuario busca y elige el programa que desea.	4. El sistema muestra el programa en el editor de texto.

Tabla 4.1: Caso de uso: Cargar programa de disco

- **Guardar programa en disco:** Guardar un programa en el dispositivo donde el usuario está trabajando. La información detallada sobre este caso de uso se puede observar en la tabla 4.2:

Guardar programa en disco	
Actores	Visitante y Registrado.
Resumen	El usuario puede almacenar en su máquina el programa que ha creado en el editor de código.
Precondición	El usuario debe tener espacio en su disco.
Postcondición	El usuario podrá visualizar que se ha descargado el archivo en su navegador.
Flujo de eventos	
Actor	*Sistema*
1. El usuario *clica* en la opción de "Guardar"	2. El sistema accede al contenido del editor de código.
	3. El sistema solicita al navegador que realice la descarga del contenido del editor.
4. El usuario hace los cambios pertinentes en el nombre y ubicación del archivo y acepta el dialogo.	

Tabla 4.2: Caso de uso: Guardar programa en disco

- **Ensamblar programa:** Ensamblar programa en el simulador de la aplicación. La información detallada sobre este caso de uso se puede observar en la tabla 4.3:

Ensamblar programa	
Actores	Visitante y Registrado.
Resumen	El usuario puede ensamblar el programa que está cargado en el editor de código en el simulador.
Precondición	El usuario debe cargar o escribir un programa en el editor de código.
Postcondición	El usuario ensamblará el código y visualizará el código en el segmento de código del simulador.
Flujo de eventos	
Actor	*Sistema*
1. El usuario carga o escribe un programa en el editor de código	
2. El usuario *clica* en la opción "Ensamblar"	3. El sistema valida el programa.
	4. El sistema ensambla el programa en el simulador.
	5. El sistema muestra al usuario los campos de registro, datos y con el programa ensamblado si no ha habido errores.

Tabla 4.3: Caso de uso: Ensamblar programa en el simulador

- **Recargar programa:** Recargar el programa en el simulador de la aplicación. La información detallada sobre este caso de uso se puede observar en la tabla 4.4:

Recargar programa	
Actores	Visitante y Registrado.
Resumen	El usuario puede ensamblar el programa que está cargado en el editor de código en el simulador. Este caso, es equivalente al de ensamblar, incluido para mejorar la usabilidad del simulador.
Precondición	El usuario debe cargar o escribir un programa en el editor de código.
Postcondición	El usuario ensamblará el código y visualizará el código en el segmento de código del simulador.
Flujo de eventos	
Actor	*Sistema*
1. El usuario carga o escribe un programa en el editor de código	
2. El usuario *clica* en la opción "Recargar"	3. El sistema valida el programa.
	4. El sistema ensambla el programa en el simulador.
	5. El sistema muestra al usuario los campos de registro, datos y con el programa ensamblado si no ha habido errores.

Tabla 4.4: Caso de uso: Recargar programa

- **Modificar código:** Modificar el programa del editor de código, tanto eliminando lineas/caracteres como añadiéndolos. La información detallada sobre este caso de uso se puede observar en la tabla 4.5:

Modificar código	
Actores	Visitante y Registrado.
Resumen	El usuario puede escribir, modificar y borrar el código del editor.
Precondición	El usuario quiere modificar el código.
Postcondición	El usuario modificará su código a su preferencia personal.
Flujo de eventos	
Actor	*Sistema*
1. El usuario cambia código en el editor	2. El sistema muestra los cambios en el editor.

Tabla 4.5: Caso de uso: Modificar el editor de código

- **Limpiar editor:** Limpiar todas las lineas o caracteres que hay en el editor de código. La información detallada sobre este caso de uso se puede observar en la tabla 4.6:

Limpiar editor	
Actores	Visitante y Registrado.
Resumen	El usuario puede borrar todo el contenido del editor de código.
Precondición	
Postcondición	El usuario eliminará todo el contenido del editor de código.
Flujo de eventos	
Actor	*Sistema*
1. El usuario *clica* en el botón de "Limpiar".	2. El sistema muestra el editor vacío.

Tabla 4.6: Caso de uso: Limpiar el editor de código

- **Ejecutar programa completo:** Ejecutar el programa hasta finalizarlo por completo. La información detallada sobre este caso de uso se puede observar en la tabla 4.7:

Ejecutar programa completo	
Actores	Visitante y Registrado.
Resumen	El usuario puede ejecutar el programa que previamente ha ensamblado en el simulador.
Precondición	El usuario habrá ensamblado un programa en el simulador.
Postcondición	El usuario ejecutará un programa en la dirección establecida.
Flujo de eventos	
Actor	*Sistema*
1. El usuario *clica* en el botón "Ejecutar".	
2. El usuario escribe la dirección de memoria desde donde quiere ejecutar el programa.	
3. El usuario *clica* en el botón "Aceptar".	4. El sistema ejecuta el programa en el simulador.
	5. El sistema muestra los nuevos valores de los registros y segmentos de datos y de texto al finalizar la ejecución del programa.

Tabla 4.7: Caso de uso: Ejecutar programa completo

- **Ejecutar por pasos:** Ejecutar pasos determinados por el usuario en el programa. La información detallada sobre este caso de uso se puede observar en la tabla 4.8:

Ejecutar por pasos	
Actores	Visitante y Registrado.
Resumen	El usuario puede ejecutar por pasos el programa.
Precondición	El usuario debe ensamblar un programa en el simulador.
Postcondición	El usuario habrá ejecutado el número de instrucciones especificadas.
Flujo de eventos	
Actor	*Sistema*
1. El usuario *clica* en el menú "Simulador".	
2. El usuario *clica* en la opción de desplegable "Pasos".	
3. El usuario elige el número de instrucciones que desea.	
4. El usuario *clica* en el botón de "Aceptar".	5. El sistema ejecuta el número de pasos seleccionados.
	6. El sistema muestra la instrucción en la que ha parado la ejecución resaltada, los nuevos valores de los registros y segmentos de datos y de texto.

Tabla 4.8: Caso de uso: Ejecutar por pasos

- **Ejecutar un paso :** Ejecutar un paso en el programa. La información detallada sobre este caso de uso se puede observar en la tabla 4.9:

Ejecutar un paso	
Actores	Visitante y Registrado.
Resumen	El usuario puede ejecutar un paso en el programa.
Precondición	El usuario debe ensamblar un programa en el simulador.
Postcondición	El usuario habrá ejecutado el programa una instrucción.
Flujo de eventos	
Actor	*Sistema*
1. El usuario *clica* en el menú "Paso".	2. El sistema ejecuta una instrucción en el programa.
	3. El sistema muestra la instrucción en la que ha parado resaltada, los nuevos valores de los registros y segmentos de datos y de texto.

Tabla 4.9: Caso de uso: Ejecutar un paso

- **Establecer puntos de ruptura por medio de botones :** Establecer *breakpoints* por medio de los botones que hay al lado de las instrucciones en el segmento de texto. La información detallada sobre este caso de uso se puede observar en la tabla 4.10:

Establecer puntos de ruptura por medio de botones	
Actores	Visitante y Registrado.
Resumen	El usuario puede establecer puntos de ruptura a través de pulsar los botones.
Precondición	No debe estar el botón que quiere establecer pulsado.
Postcondición	El usuario logrará establecer el punto de ruptura en la dirección que desea.
Flujo de eventos	
Actor	*Sistema*
1. El usuario busca la dirección que desea en la lista de instrucciones.	
2. El usuario pulsa el botón que desea	3. El sistema colorea el *botón* que ha sido pulsado.

Tabla 4.10: Caso de uso: Establecer puntos de ruptura por medio de *botones*

- **Establecer puntos de ruptura por medio de menú:** Establecer *breakpoints* por medio de escribir la dirección de memoria en una opción que ofrece el menú. La información detallada sobre este caso de uso se puede observar en la tabla 4.11:

Establecer puntos de ruptura por medio de menú	
Actores	Visitante y Registrado.
Resumen	El usuario escribir la dirección de memoria de la instrucción en la que quiere añadir el *breakpoint*
Precondición	No debe estar el botón que quiere establecer pulsado.
Postcondición	El usuario logrará establecer el punto de ruptura en la dirección que desea.
Flujo de eventos	
Actor	*Sistema*
1. El usuario pulsa el menú "Simulador".	
2. El usuario pulsa en la opción "Puntos de ruptura".	3. El sistema muestra un dialogo para introducir una instrucción.
4. El usuario introduce la dirección de memoria de la instrucción que desea establecer el punto de ruptura y pulsa en aceptar	5. El sistema valida que la dirección sea correcta.
	6. El sistema muestra en el segmento de código el *breakpoint* establecido.

Tabla 4.11: Caso de uso: Establecer puntos de ruptura por medio de menú

- **Eliminar puntos de ruptura por medio de botones:** Eliminar *breakpoints* por medio de los botones que hay al lado de las instrucciones en el segmento de texto. La información detallada sobre este caso de uso se puede observar en la tabla 4.12:

Eliminar puntos de ruptura por medio de botones	
Actores	Visitante y Registrado.
Resumen	El usuario puede eliminar puntos de ruptura a través de pulsar los botones.
Precondición	El botón que desea eliminar tiene que estar seleccionado.
Postcondición	El usuario podrá eliminar el punto de ruptura.
Flujo de eventos	
Actor	*Sistema*
1. El usuario pulsa un botón que ya está coloreado.	2. El sistema elimina el punto de ruptura y muestra el botón sin selección.

Tabla 4.12: Caso de uso: Eliminar puntos de ruptura por medio de *botones*

- **Continuar con la ejecución:** Continuar con la ejecución del programa después de haber parado por un *breakpoint* hasta finalizarlo por completo o encontrar un *breakpoint*. La información detallada sobre este caso de uso se puede observar en la tabla 4.13:

Continuar con la ejecución	
Actores	Visitante y Registrado.
Resumen	El usuario continua con la ejecución del programa.
Precondición	El usuario debe tener un programa ensamblado en el simulador.
Postcondición	El usuario continuará con la ejecución del programa hasta finalizarlo por completo o encontrar un *breakpoint*.
Flujo de eventos	
Actor	*Sistema*
1. El usuario *clica* en la opción "Continuar".	2. El sistema continua con la ejecución hasta finalizarlo por completo o encontrar un *breakpoint*

Tabla 4.13: Caso de uso: Continuar con la ejecución

- **Ver registros en decimal:** Visualizar los registros en formato decimal. La información detallada sobre este caso de uso se puede observar en la tabla 4.14:

Ver registros en decimal	
Actores	Visitante y Registrado.
Resumen	El usuario visualiza los valores de los registros del procesador en decimal.
Precondición	El usuario debe tener un programa ensamblado en el simulador.
Postcondición	El usuario visualizará los registros en decimal del programa cargado.
Flujo de eventos	
Actor	*Sistema*
1. El usuario pulsa en la pestaña *DEC*	2. El sistema muestra los registros en decimal.
3. El usuario visualiza los registros en decimal.	

Tabla 4.14: Caso de uso: Ver registros en decimal

- **Ver registros en hexadecimal:** Visualizar los registros en formato hexadecimal. La información detallada sobre este caso de uso se puede observar en la tabla 4.15:

Ver registros en hexadecimal	
Actores	Visitante y Registrado.
Resumen	El usuario visualiza los valores de los registros del procesador en hexadecimal.
Precondición	El usuario debe tener un programa ensamblado en el simulador.
Postcondición	El usuario visualizará los registros en hexadecimal del programa cargado.
Flujo de eventos	
Actor	*Sistema*
1. El usuario pulsa en la pestaña *HEC*	2. El sistema muestra los registros en hexadecimal.
3. El usuario visualiza los registros en hexadecimal.	

Tabla 4.15: Caso de uso: Ver registros en hexadecimal

- **Registrarse:** Registrarse en la aplicación SpimUPV. La información detallada sobre este caso de uso se puede observar en la tabla 4.16:

Registrarse	
Actores	Visitante
Resumen	El usuario crea una cuenta en la aplicación
Precondición	El usuario desea registrarse en la aplicación y no tiene cuenta.
Postcondición	El usuario se registra, teniendo una cuenta creada en la aplicación
Flujo de eventos	
Actor	*Sistema*
1. El usuario desea registrarse en la aplicación.	
2. El usuario pulsa en el icono superior derecho de cuenta.	
	3. El sistema muestra la página de “Login“
4. El usuario pulsa en “Crear Cuenta“	
	5. El sistema muestra la página de “Registro“
6. El usuario rellena los campos solicitados.	
7. El usuario *clica* en el botón “Crear cuenta“	
	8. El sistema valida los datos.
	9. El sistema da de alta al usuario en la base de datos
	10. El sistema muestra la página principal

Tabla 4.16: Caso de uso: Registrarse

- **Iniciar sesión:** Iniciar sesión en la aplicación SpimUPV. La información detallada sobre este caso de uso se puede observar en la tabla 4.17:

Iniciar sesión	
Actores	Visitante
Resumen	El usuario introduce los datos con los que se había registrado en la aplicación e inicia sesión en la web.
Precondición	El usuario debe registrarse previamente.
Postcondición	El usuario pasará a ser Registrado
Flujo de eventos	
Actor	*Sistema*
1. El usuario introduce su correo electrónico y contraseña.	
2. El usuario *clica* en el botón de iniciar sesión.	3. El sistema valida los datos.
	5. El sistema *loguea* al usuario.
	6. El sistema muestra la aplicación con las herramientas habilitadas para un usuario registrado.

Tabla 4.17: Caso de uso: Iniciar sesión.

- **Ver información del simulador:** Aunténticarse en la aplicación SpimUPV. La información detallada sobre este caso de uso se puede observar en la tabla 4.17:

Ver información del simulador	
Actores	Visitante y Registrado.
Resumen	El usuario puede visualizar la información acerca del simulador: quienes lo han creado o el curso que se realizó este.
Precondición	
Postcondición	El usuario visualizará la información anteriormente descrita en un dialogo.
Flujo de eventos	
Actor	*Sistema*
1. El usuario *clica* en el botón de "Ayuda"	2. El sistema muestra la información en un dialogo.

Tabla 4.18: Caso de uso: Ver información del simulador

- **Cerrar sesión:** *Desloguearse* en la aplicación SpimUPV. La información detallada sobre este caso de uso se puede observar en la tabla 4.19:

Cerrar sesión	
Actores	Registrado.
Resumen	El usuario cierra su sesión en la aplicación.
Precondición	El usuario tiene que estar *logueado* previamente.
Postcondición	El usuario cerrará sesión satisfactoriamente.
Flujo de eventos	
Actor	*Sistema*
1. El usuario *clica* en el icono para cerrar sesión.	2. El sistema cierra sesión de la cuenta.
	3. El sistema muestra la pantalla de inicio de sesión.

Tabla 4.19: Caso de uso: Cerrar sesión.

- **Ver contraseña:** Visualizar la contraseña sin protección de asteriscos tanto en los campos de contraseña de "Iniciar sesión" y "Registrarse". La información detallada sobre este caso de uso se puede observar en la tabla 4.20:

Ver contraseña	
Actores	Visitante y Registrado.
Resumen	El usuario puede ver la contraseña que ha escrito.
Precondición	El usuario debe escribir una contraseña.
Postcondición	El usuario puede visualizar la contraseña desprotegida.
Flujo de eventos	
Actor	*Sistema*
1. El usuario *clica* en el botón con el icono de un ojo.	2. El sistema muestra la contraseña sin protección de asteriscos.

Tabla 4.20: Caso de uso: Ver contraseña.

- **Restablecer contraseña:** Editar la contraseña de acceso a la aplicación. La información detallada sobre este caso de uso se puede observar en la tabla 4.21:

Restablecer contraseña	
Actores	Registrado.
Resumen	El usuario puede editar la contraseña de acceso a la aplicación.
Precondición	El usuario debe tener acceso al correo electrónico con el cual se registró.
Postcondición	El usuario puede acceder con una contraseña distinta a la aplicación SpimUPV.
Flujo de eventos	
Actor	*Sistema*
1. El usuario *clica* en el hipervínculo "¿Has olvidado la contraseña? ".	2. El sistema muestra una pantalla que solicita al usuario el correo vinculado a la cuenta a la que se quiere cambiar la contraseña.
3. El usuario introduce su *email* y *clica* en el botón enviar.	4. El sistema comprueba que el correo existe en la base de datos y envía un correo de restablecimiento de contraseña.
4. El usuario accede a su correo electrónico, y pulsa en el enlace.	5. El sistema muestra una pantalla donde se solicita que el usuario introduzca su nueva contraseña.
6. El usuario introduce su nueva contraseña.	7. El sistema valida que la contraseña introducida sea correcta y cambia la contraseña al usuario.

Tabla 4.21: Caso de uso: Restablecer contraseña.

- **Eliminar cuenta:** Borrar una cuenta de usuario de la base de datos. La información detallada sobre este caso de uso se puede observar en la tabla 4.22:

Eliminar cuenta	
Actores	Registrado.
Resumen	El usuario puede borrar su cuenta de la base de datos de la aplicación.
Precondición	El usuario debe estar previamente registrado y acceder a su cuenta.
Postcondición	El usuario borrará su cuenta de la base de datos de la aplicación.
Flujo de eventos	
Actor	*Sistema*
1. El usuario *clica* en el botón de cuenta del usuario.	2. El sistema muestra un dialogo que contiene la información del usuario *logueado*.
3. El usuario *clica* en el botón de borrar cuenta.	4. El sistema elimina la cuenta del usuario en la base de datos.

Tabla 4.22: Caso de uso: Eliminar cuenta.

- **Ver información del usuario:**Visualizar la información del usuario. La información detallada sobre este caso de uso se puede observar en la tabla 4.23:

Ver información del usuario	
Actores	Registrado.
Resumen	El usuario visualiza su información de usuario: nombre, correo electrónico, curso y asignatura matriculada.
Precondición	El usuario debe estar registrado y *logueado* en la aplicación.
Postcondición	El usuario visualizará su información de usuario: nombre, correo electrónico, curso y asignatura matriculada.
Flujo de eventos	
Actor	*Sistema*
1. El usuario *clica* en el icono que representa el perfil de la cuenta.	2. El sistema muestra un dialogo con la información del usuario *logueado*.
3. El usuario *clica* en el icono que representa el perfil de la cuenta.	4. El sistema muestra un dialogo con la información del usuario *logueado*.

Tabla 4.23: Caso de uso:Ver información del usuario

- **Ver ayuda:** Visualizar la ayuda que ofrece SpimUPV para los usuarios que accedan a la aplicación. La información detallada sobre este caso de uso se puede observar en la tabla 4.24:

Ver ayuda	
Actores	Visitante y Registrado.
Resumen	El usuario visualiza la ayuda referente a como utilizar la aplicación.
Precondición	
Postcondición	El usuario visualizará la ayuda que proporciona la aplicación.
Flujo de eventos	
Actor	*Sistema*
1. El usuario *clica* en el botón de "Ayuda"	2. El sistema muestra la información de ayuda en un dialogo.

Tabla 4.24: Caso de uso:Ver ayuda

- **Subir programa al servidor desde el editor:** Subir el programa que esté cargado en el editor. La información detallada sobre este caso de uso se puede observar en la tabla 4.25:

Subir programa al servidor desde el editor.	
Actores	Registrado.
Resumen	El usuario sube un programa al servidor.
Precondición	El usuario tendrá el programa que desee cargar en su sistema.
Postcondición	El usuario subirá un programa al servidor.
Flujo de eventos	
Actor	*Sistema*
1. El usuario *clica* en el botón de subir archivos.	2. El sistema valida el programa que se quiere subir.
	3. El sistema muestra que se ha subido un programa satisfactoriamente en el servidor.

Tabla 4.25: Caso de uso: Subir programa al servidor desde el editor.

- **Eliminar programa del servidor:** Eliminar un programa que se almacene en el servidor. La información detallada sobre este caso de uso se puede observar en la tabla 4.26:

Eliminar programa del servidor.	
Actores	Registrado.
Resumen	El usuario elimina un programa que ha subido al servidor.
Precondición	El usuario debe tener algún programa subido al servidor.
Postcondición	El usuario eliminará los programas que desee.
Flujo de eventos	
Actor	*Sistema*
1. El usuario *clica* en botón de eliminar, representado por un icono de papelera.	2. El sistema eliminará el programa del servidor.
	3. El sistema muestra la lista de programas sin el que ha eliminado el usuario.

Tabla 4.26: Caso de uso: Eliminar programa del servidor.

- **Subir programa desde disco al servidor:** Subir cualquier programa de disco. La información detallada sobre este caso de uso se puede observar en la tabla 4.27:

Subir programa desde disco al servidor.	
Actores	Registrado.
Resumen	El usuario sube un programa al servidor.
Precondición	El usuario tendrá el programa que desee cargar en su sistema.
Postcondición	El usuario subirá un programa al servidor.
Flujo de eventos	
Actor	*Sistema*
1. El usuario *clica* en el botón de subir archivos.	2. El sistema valida el programa que se quiere subir no supera el tamaño máximo de subida (1MB).
	3. El sistema muestra que se ha subido un programa satisfactoriamente en el servidor.

Tabla 4.27: Caso de uso: Subir programa desde disco al servidor.

- **Cargar programa del servidor:** Cargar el programa que está subido en el servidor al editor de código. La información detallada sobre este caso de uso se puede observar en la tabla 4.28:

Cargar programa del servidor.	
Actores	Registrado.
Resumen	El usuario carga un programa del servidor en el simulador.
Precondición	El usuario debe haber subido algún programa en el servidor.
Postcondición	El usuario cargará un programa en el servidor.
Flujo de eventos	
Actor	*Sistema*
1. El usuario *clica* un programa que quiere cargar	
2. El usuario pulsa en el botón "Cargar en editor"	3. El sistema carga el programa en el editor.
	4. El sistema muestra el programa en el editor de código.

Tabla 4.28: Caso de uso: Cargar programa del servidor.

- **Ver programa del servidor:** Visualizar los programas del usuario que están subidos en el servidor. La información detallada sobre este caso de uso se puede observar en la tabla 4.29:

Ver programa del servidor.	
Actores	Registrado.
Resumen	El usuario puede visualizar los programas que tiene subidos en el servidor.
Precondición	El usuario debe subir algún programa anteriormente.
Postcondición	El usuario podrá visualizar los programas que tiene en el servidor.

Flujo de eventos	
Actor	*Sistema*
1. El usuario accede a la ventana de sus programas.	2. El sistema muestra una lista de los programas que tiene ese usuario en el servidor.

Tabla 4.29: Caso de uso: Ver programa del servidor.

4.2 Modelado por actores

A causa del elevado número de casos de uso, y para ayudar a la legibilidad de los diagramas, se ha optado por dividirlos.

4.2.1. Visitante

Este actor hace referencia a cualquier visitante en la web que no esté registrado. En la figura 4.1 podemos observar el diagrama de casos de uso para el actor visitante.

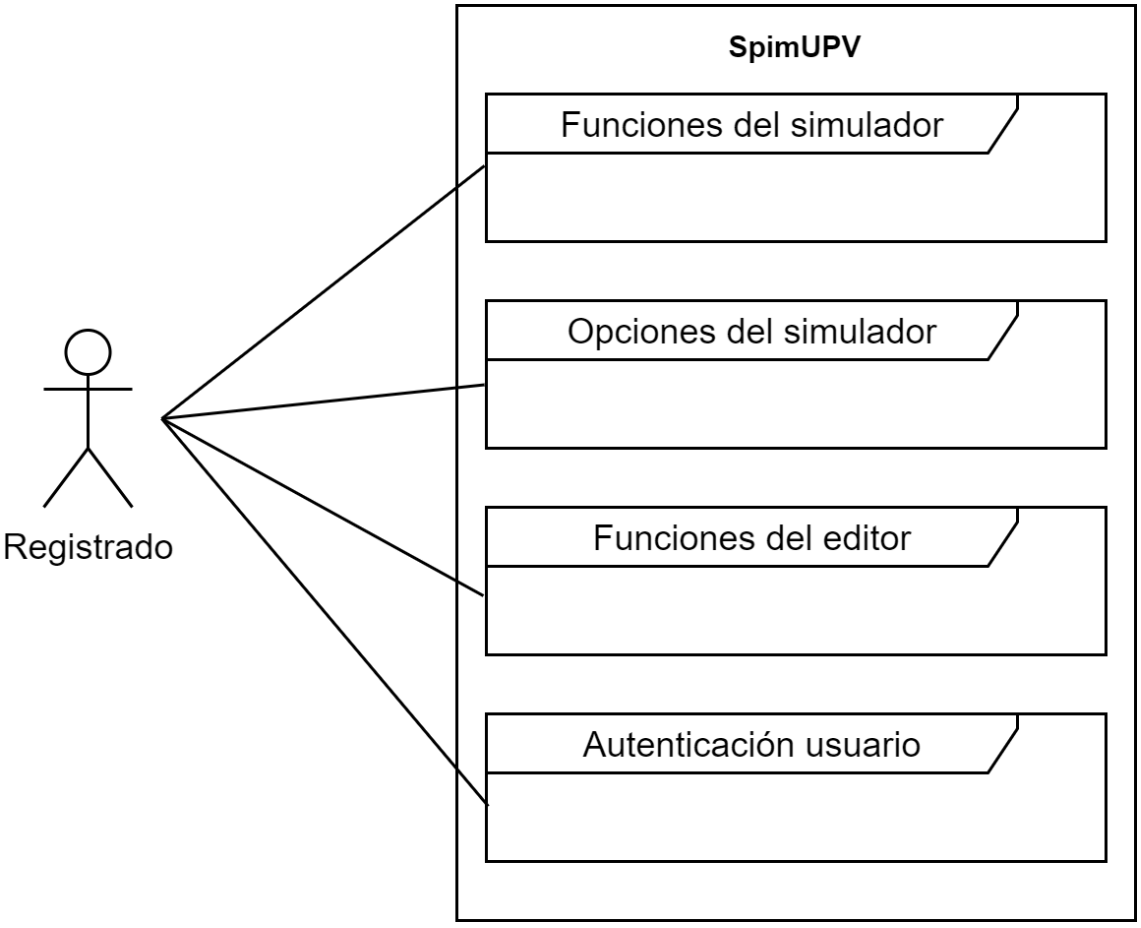

Figura 4.1: Diagrama de casos de uso para el actor visitante.

A continuación se procederá a dividir el diagrama para ayudar a su legibilidad tal y como se puede observar en las figuras 4.2, 4.3, 4.4 y 4.5.

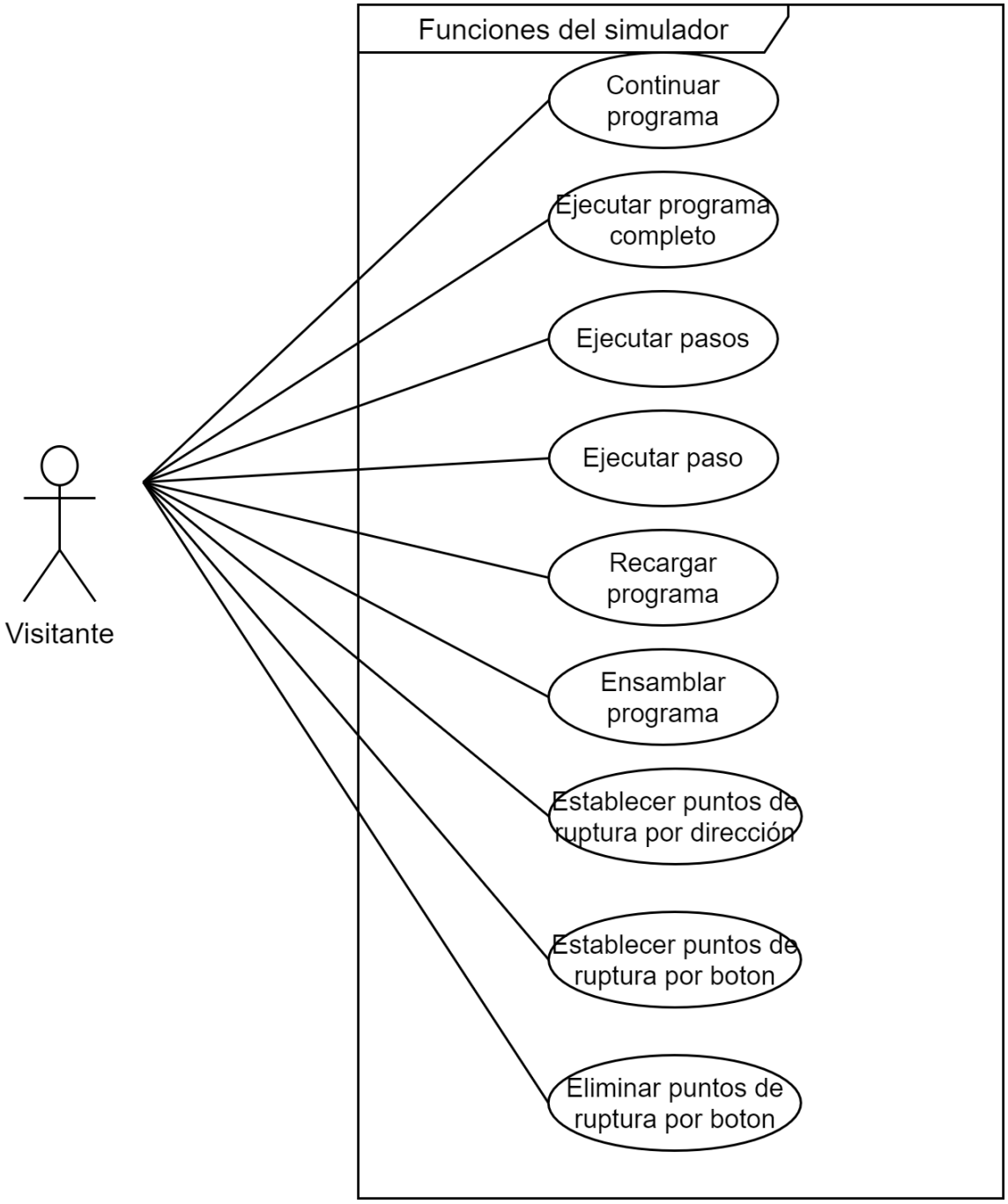

Figura 4.2: Diagrama de casos de uso: Funciones del simulador para el actor visitante.

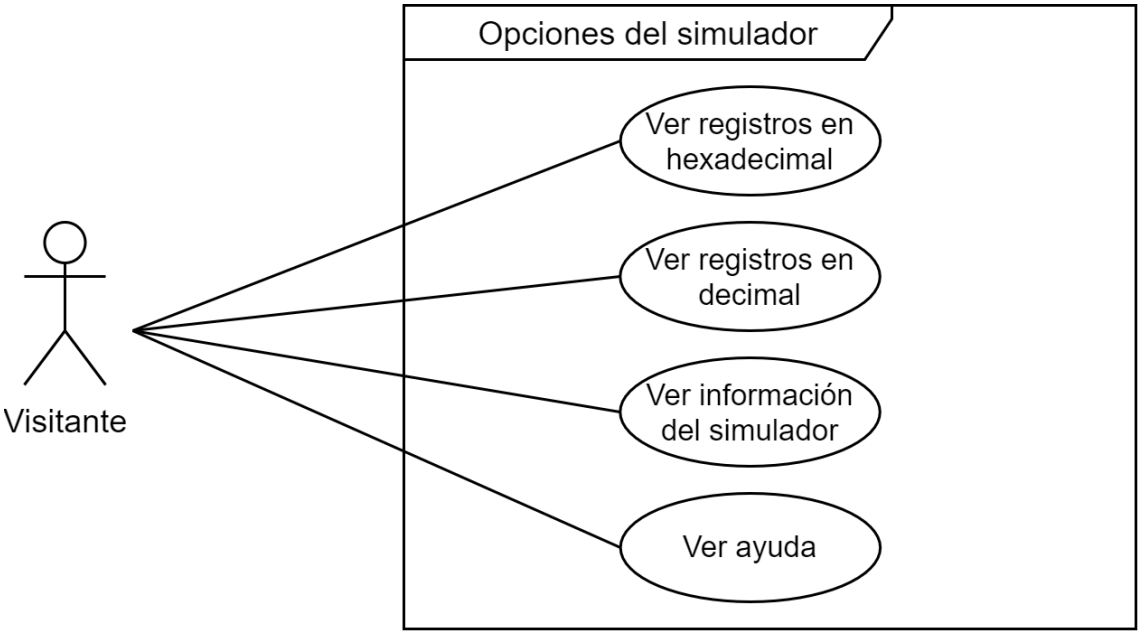

Figura 4.3: Diagrama de casos de uso: Opciones del simulador para el actor visitante.

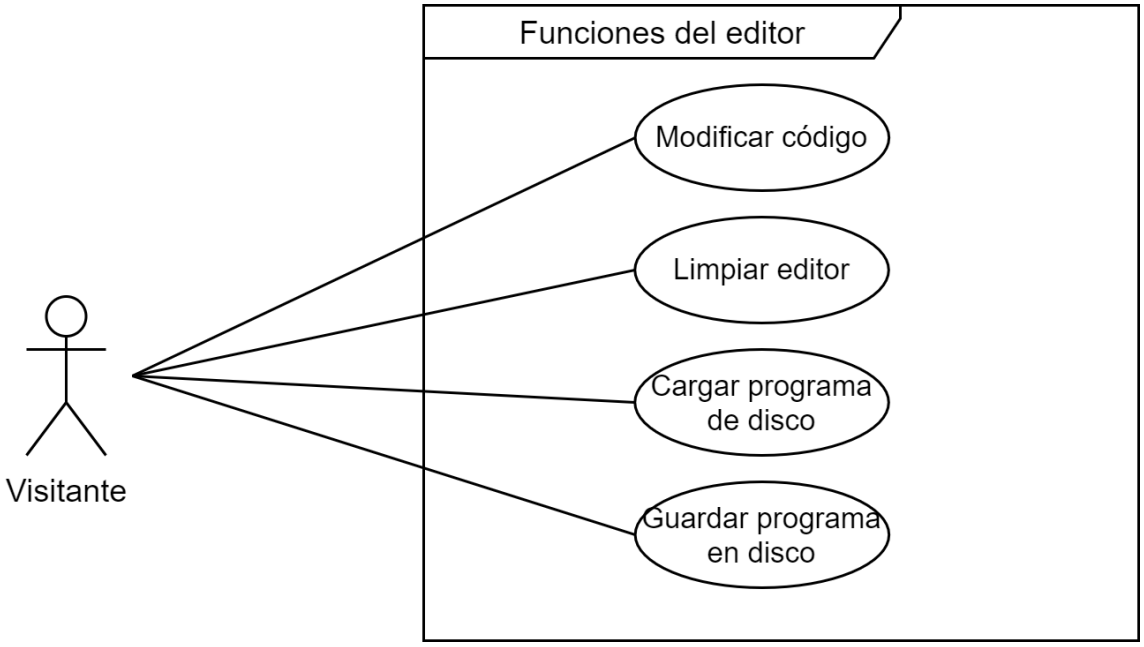

Figura 4.4: Diagrama de casos de uso: Funciones del editor para el actor visitante.

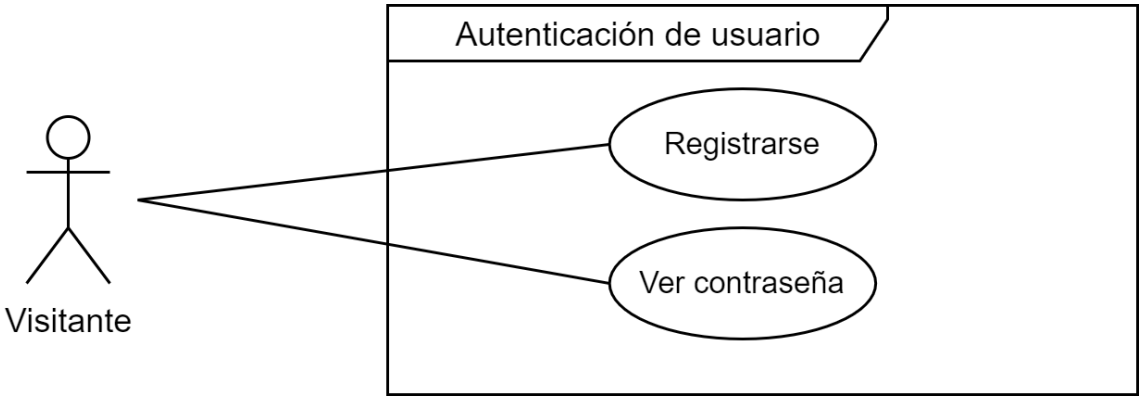

Figura 4.5: Diagrama de casos de uso: Autenticación de usuario para el actor visitante.

4.2.2. Registrado

Este actor hace referencia a cualquier usuario que se registre en la web. En la figura 4.6 podemos observar el diagrama de casos de uso para el actor registrado.

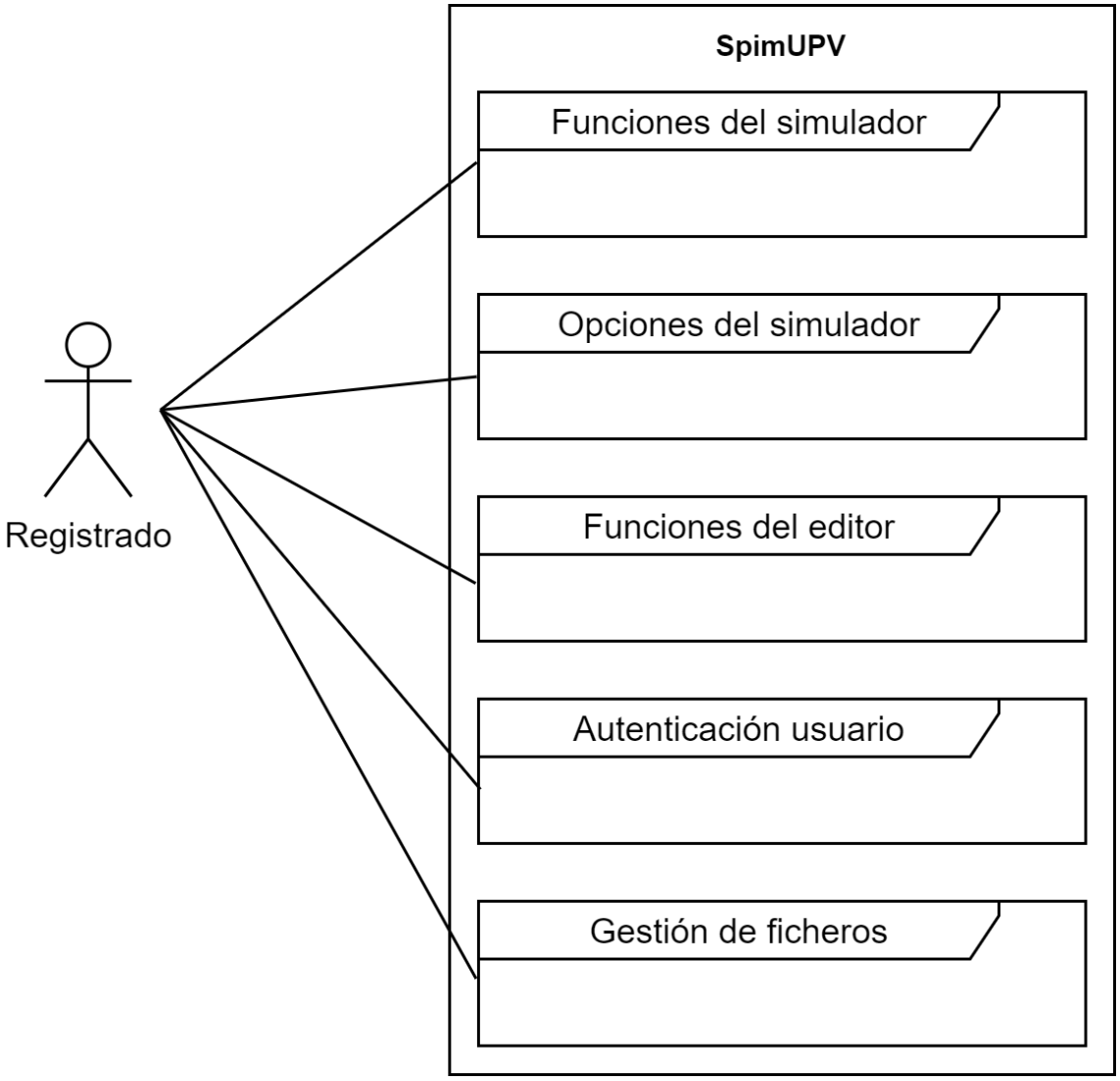

Figura 4.6: Diagrama de casos de uso para el actor registrado.

A continuación se procederá a dividir el diagrama para ayudar a su legibilidad tal y como se puede observar en las figuras 4.7, 4.8, 4.9, 4.10 y 4.11.

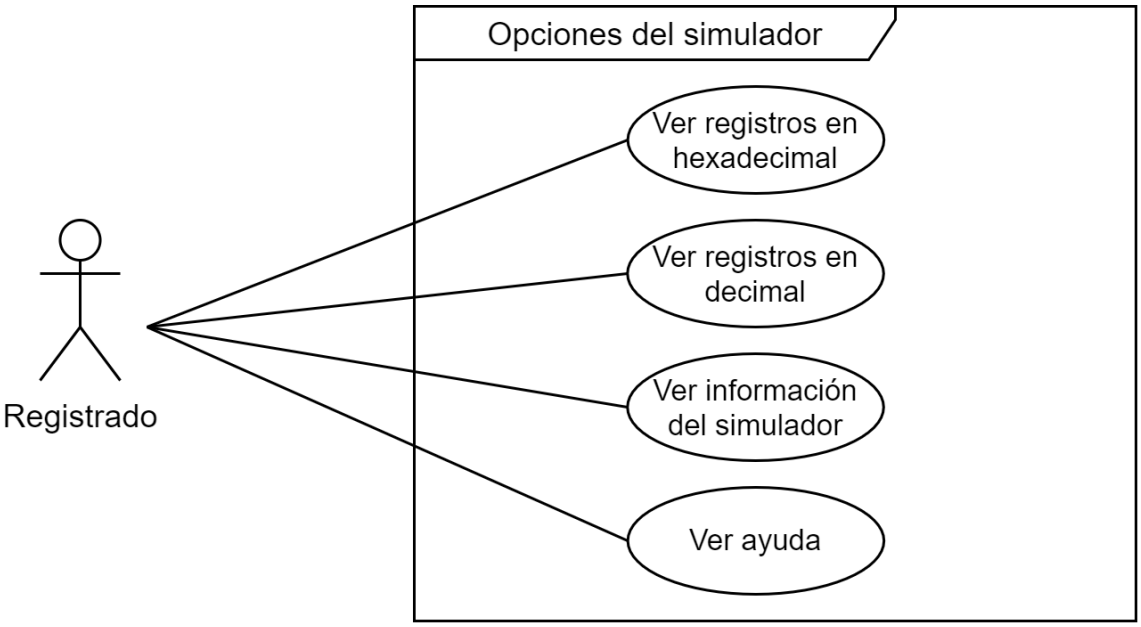

Figura 4.7: Diagrama de casos de uso: Opciones del simulador para el actor registrado.

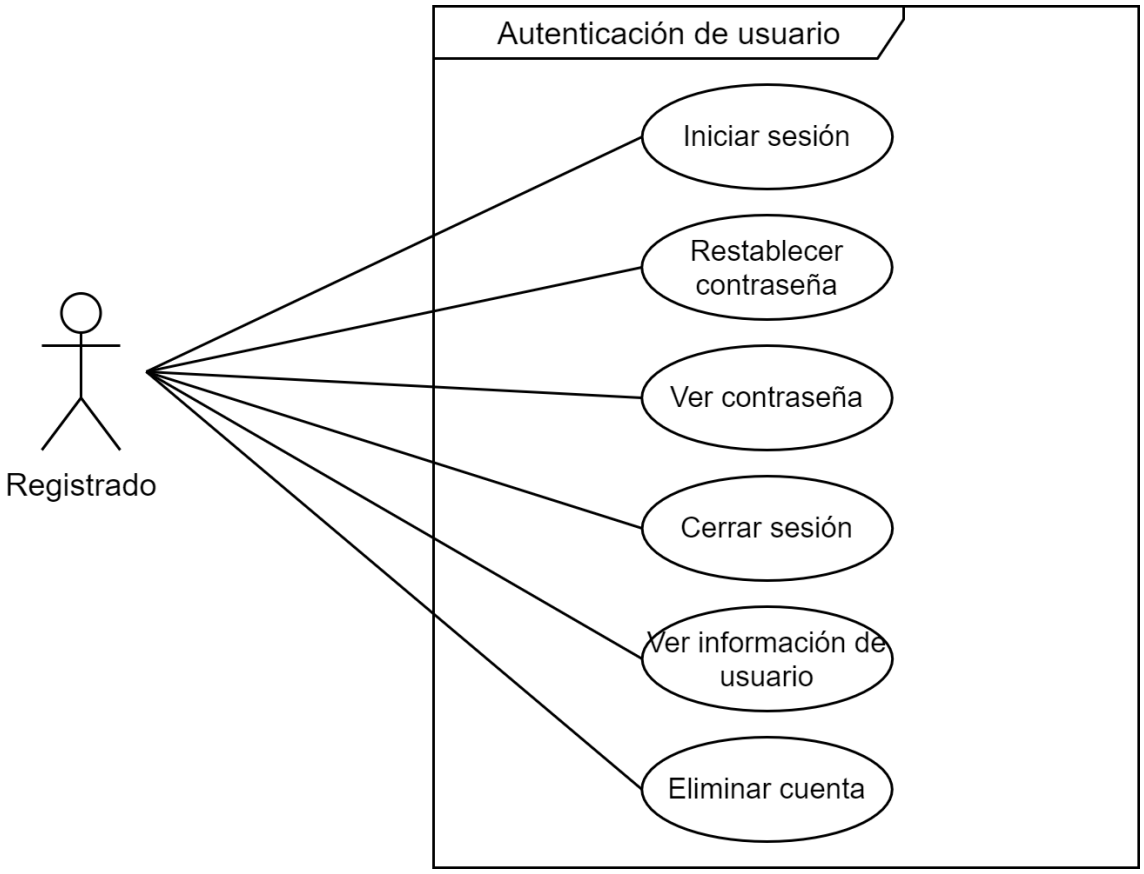

Figura 4.8: Diagrama de casos de uso: Autenticación para el actor registrado.

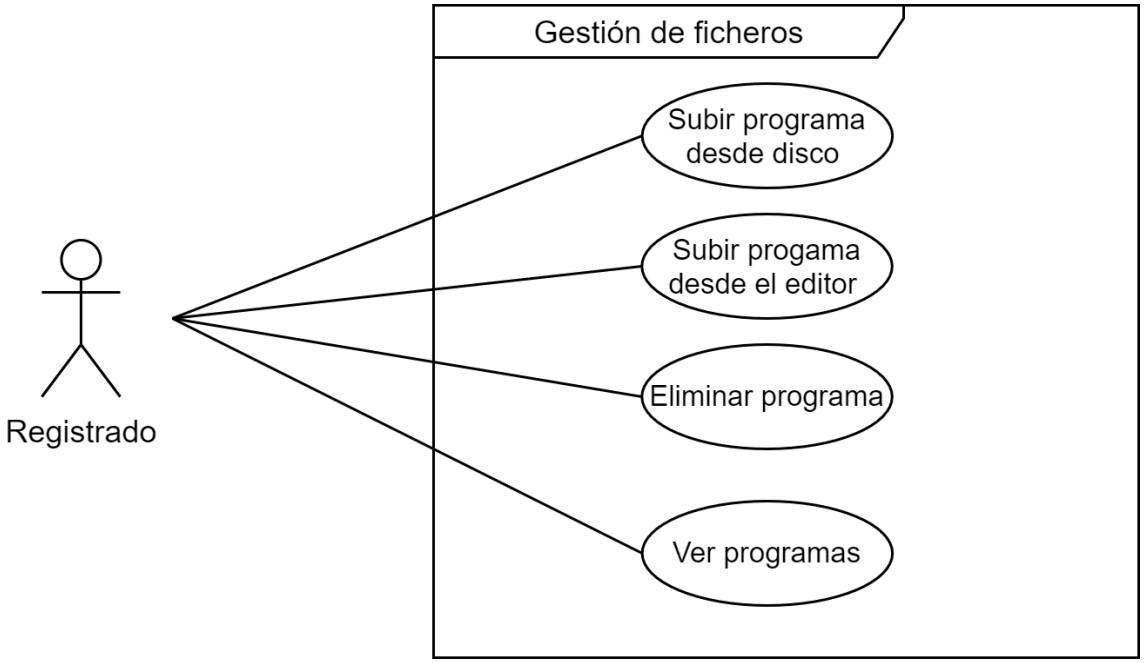

Figura 4.9: Diagrama de casos de uso: Gestión de ficheros para el actor registrado.

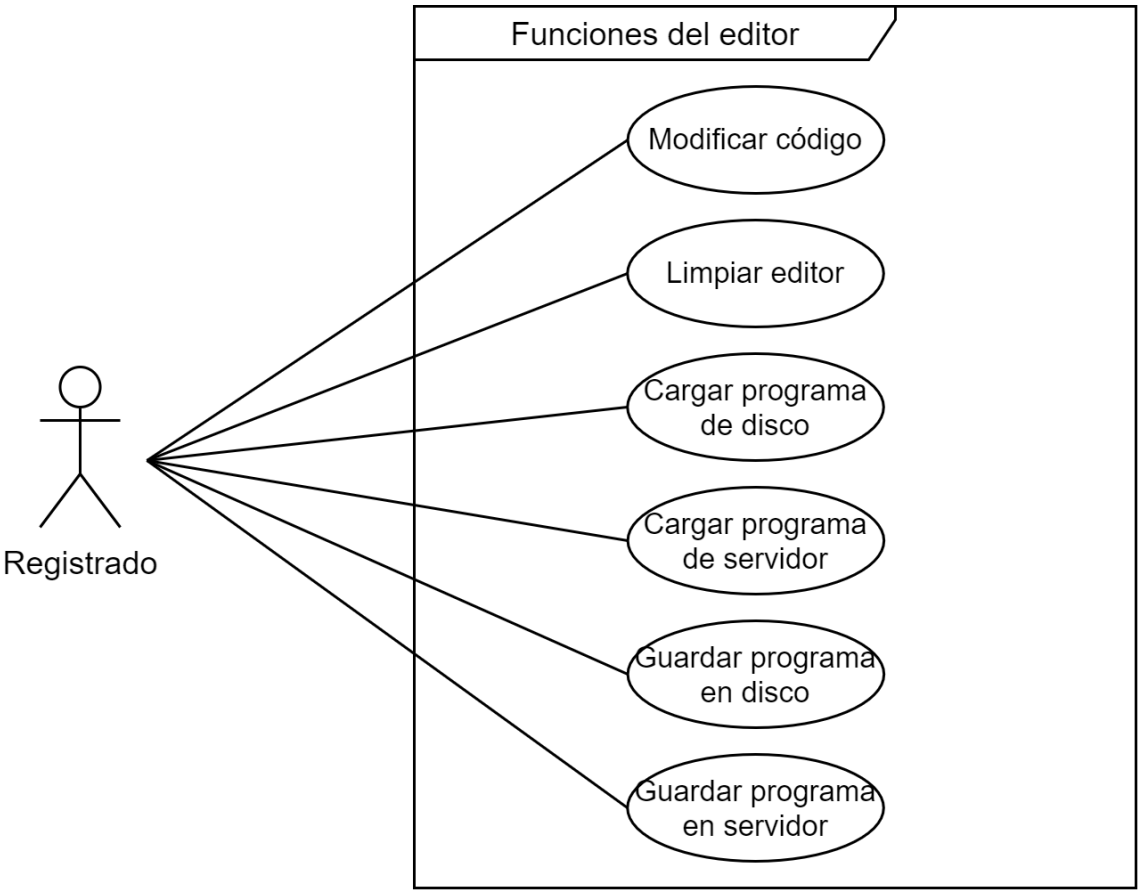

Figura 4.10: Diagrama de casos de uso: Funciones del editor para el actor registrado.

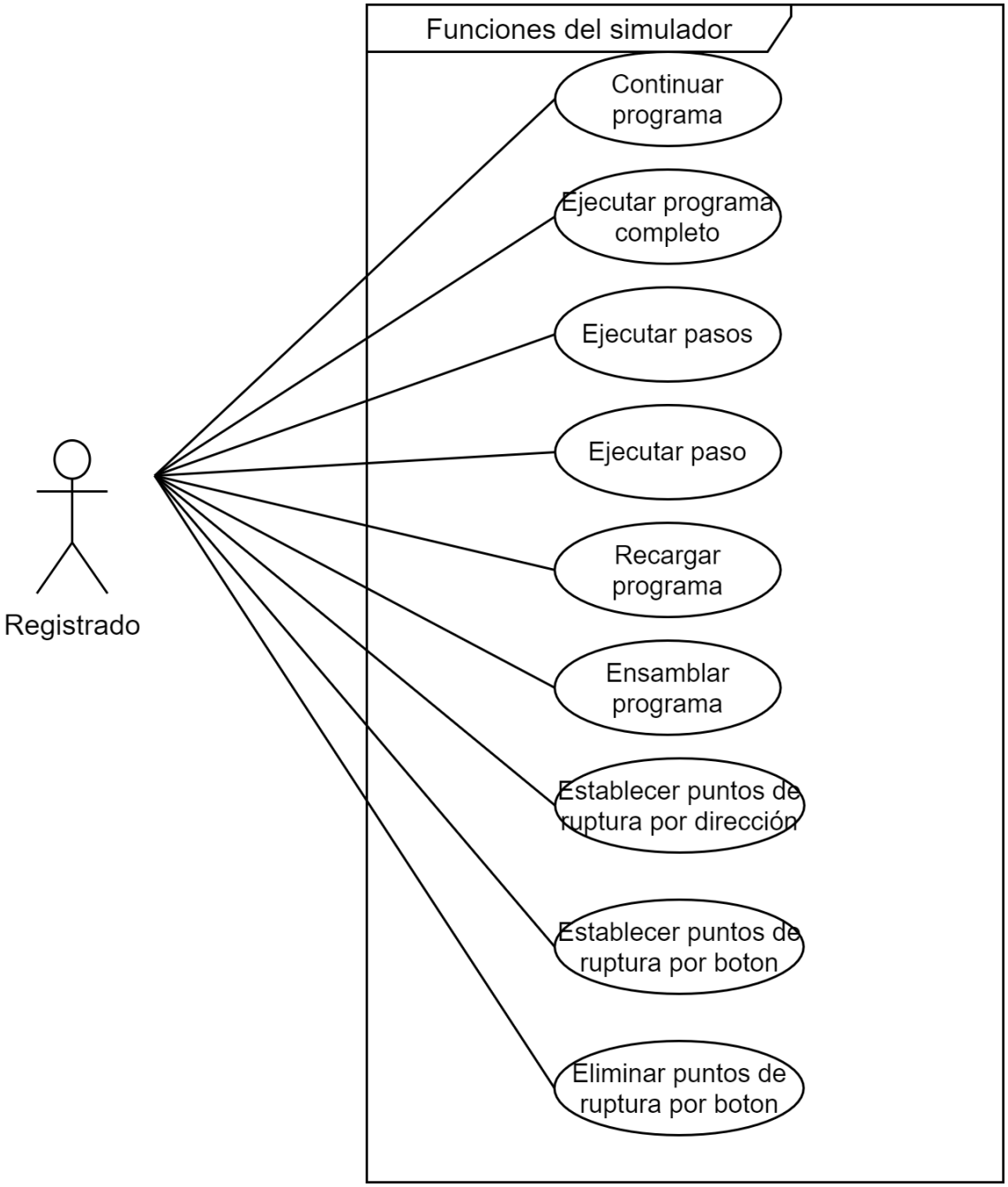

Figura 4.11: Diagrama de casos de uso: Funciones del simulador para el actor registrado.

CAPÍTULO 5
Diseño de la solución

En el presente capitulo se establecerán las estructuras que serán implementadas en el proyecto, haciendo hincapié en el diseño de la arquitectura del software, el de la base de datos y la interfaz web.

5.1 Diseño MVC

El diseño arquitectónico que se empleará en el presente proyecto será el patrón MVC (Modelo-Vista-Controlador). Así pues, MVC propone una separación en tres componentes diferentes:

- **Modelo:** encargado de manejar los datos y la lógica de negocios de la aplicación.
- **Vista:** encargado del diseño y la presentación de los datos.
- **Controlador:** encargado de enrutar los comandos a los modelos y las vistas.

En la figura 5.1 se pueden ver representados los tres componentes que forman el patrón MVC y como se comunican tanto entre ellos como con el usuario de la aplicación.

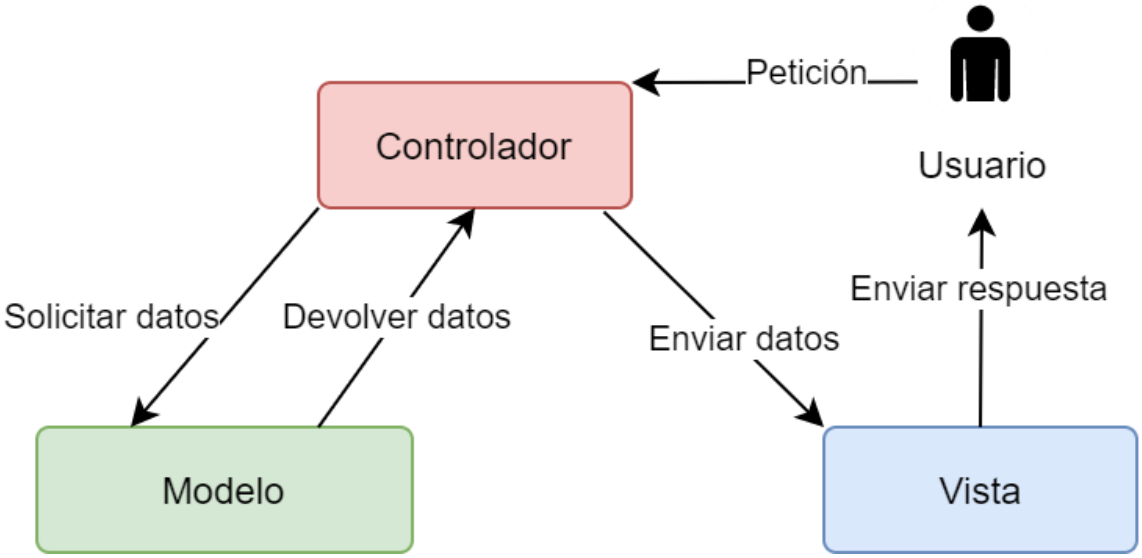

Figura 5.1: Patrón de diseño arquitectónico MVC.

Por otro lado, las ventajas que ofrece este patrón y por las cuales ha sido elegido son:

- Permite la reutilización de código.
- Facilita su escalabilidad.
- Permite la separación de la estructura en diferentes archivos para conseguir tanto facilitar el desarrollo como el futuro mantenimiento.

5.2 Diseño de la base de datos

En el presente apartado se va a exponer el diseño previo que se ha realizado para la base de datos. Así pues, podemos encontrar dos colecciones distintas:

- **User (Usuario):** Referente al usuario que logrará registrarse en la aplicación. Contendrá los campos (todos ellos serán requeridos):
  - **_id:** Identificación única proporcionada por la base de datos elegida.
  - **name:** Nombre que introduce el usuario que solo podrá contener letras, y que tendrá: un mínimo de 3 y un máximo de 255 caracteres.
  - **email:** Correo electrónico que introduce el usuario, tendrá que tener un formato acorde al de un correo electrónico convencional, un mínimo de 6 y un máximo de 255 caracteres.
  - **password:** Contraseña que introduce el usuario y que será encriptada. Tendrá un mínimo de 6 y un máximo de 1024 caracteres.
  - **course:** Curso en el que se encuentra el usuario registrado. Tendrá un formato “yyyy-yyyy"siendo un ejemplo valido "2020-2021". Este dato se recogerá para fines meramente estadísticos.
  - **resetToken:** Código único formado de manera aleatoria que se vincula a un usuario cuando este solicita el cambio de contraseña.
  - **expireToken:** Fecha en la que el *resetToken* caduca. Siempre será una hora posterior a la hora en la que se ha solicitado el cambio de contraseña, para así garantizar seguridad en la aplicación.
  - **subject:** Asignatura en el que está matriculado el usuario. Tendrá un mínimo de 3 y un máximo de 255 caracteres. Este dato se recogerá para fines meramente estadísticos.
  - **date:** Fecha de creación del usuario, proporcionada por la base de datos utilizada. Este dato se recogerá para fines meramente estadísticos.
- **File(Archivo):** Referente al programa en código ensamblador que un usuario subirá a la aplicación. Contendrá los campos (todos ellos serán requeridos):
  - **_id:** Identificación única proporcionada por la base de datos elegida.
  - **userID:** Identificación del usuario, para asegurar que es poseedor de ese programa.
  - **fileName:** Nombre proporcionado por el usuario. Cada usuario les asignará un nombre único para su relación usuario y programa. Este nombre tendrá un mínimo de 3 y un máximo de 255 caracteres.
  - **path:** Ruta en la que se almacenará el programa guardado por el usuario.
  - **date:** Fecha de creación del archivo, proporcionada por la base de datos utilizada.

En la figura 5.2 podemos observar el esquema de la base de datos, en ella además podemos ver una relación que nos mostrará que un usuario podrá tener cualquier número de programas, pero que un programa solo podrá tener un dueño (usuario).

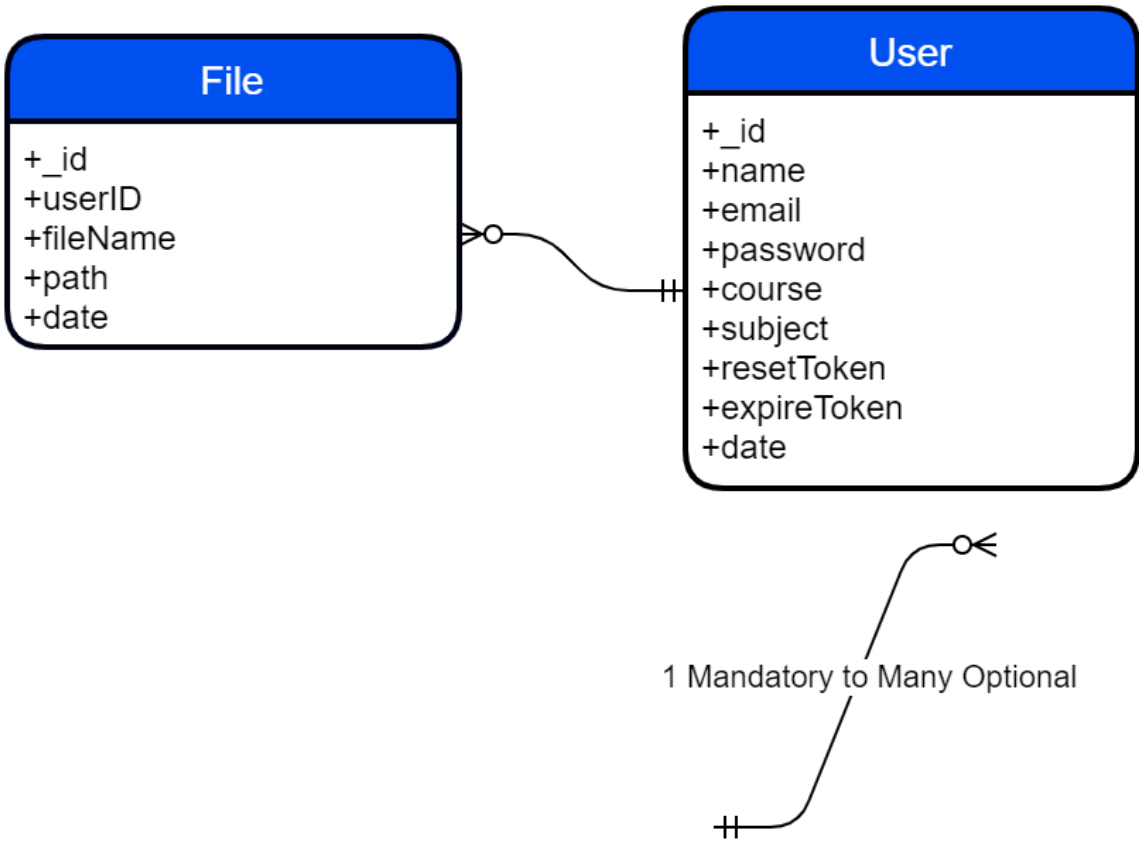

Figura 5.2: Diseño de la base de datos

5.3 Diseño de la interfaz web

Para el diseño de la interfaz web se ha optado por asemejarla todo lo posible al programa PCSpim, esto es debido a, que si los alumnos ya están familiarizados con la interfaz de la que se parte, siendo la curva de aprendizaje menor, se conseguirá que estos usuarios puedan desenvolverse sin ningún problema en ella. Para el proceso de diseño se ha optado por crear diferentes *wireframes* para ayudar en la estructura visual con la que contará la web. Cabe destacar que existen muy pocas diferencias entre la página del visitante y la del usuario registrado, de esta forma se consigue mejorar la usabilidad de la aplicación. Así pues, pasaremos a exponer estos bocetos hechos con la herramienta Balsamiq [13]:

Página principal del simulador, visitante: En la figura 5.4 podemos observar el prototipo para la página del simulador, accediendo en modo visitante.

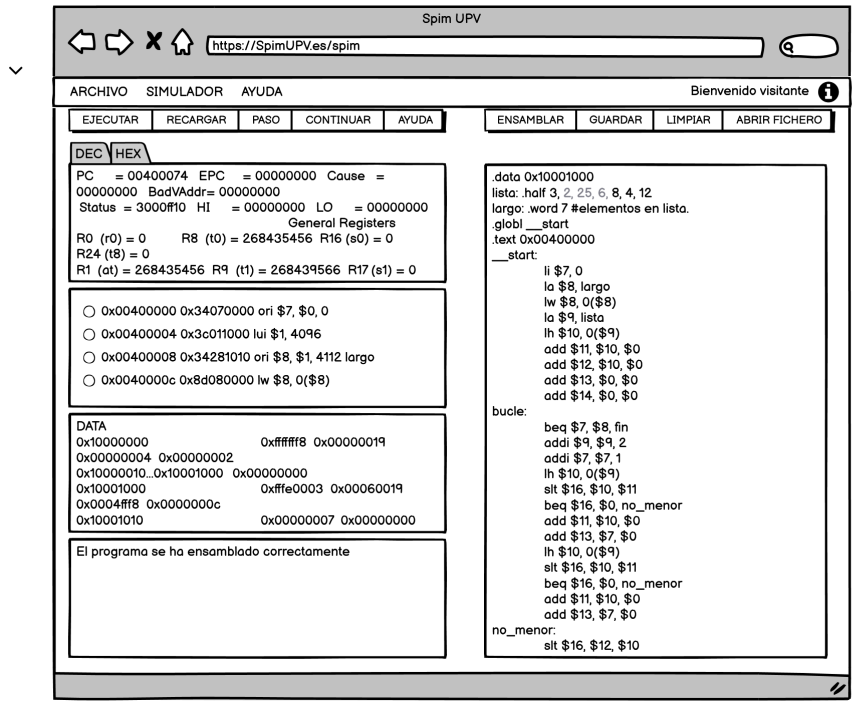

Figura 5.3: Wireframe: Página principal del simulador para el usuario visitante

Página principal del simulador, registrado: En la figura 5.4 podemos observar el prototipo para la página del simulador, accediendo en modo usuario registrado.

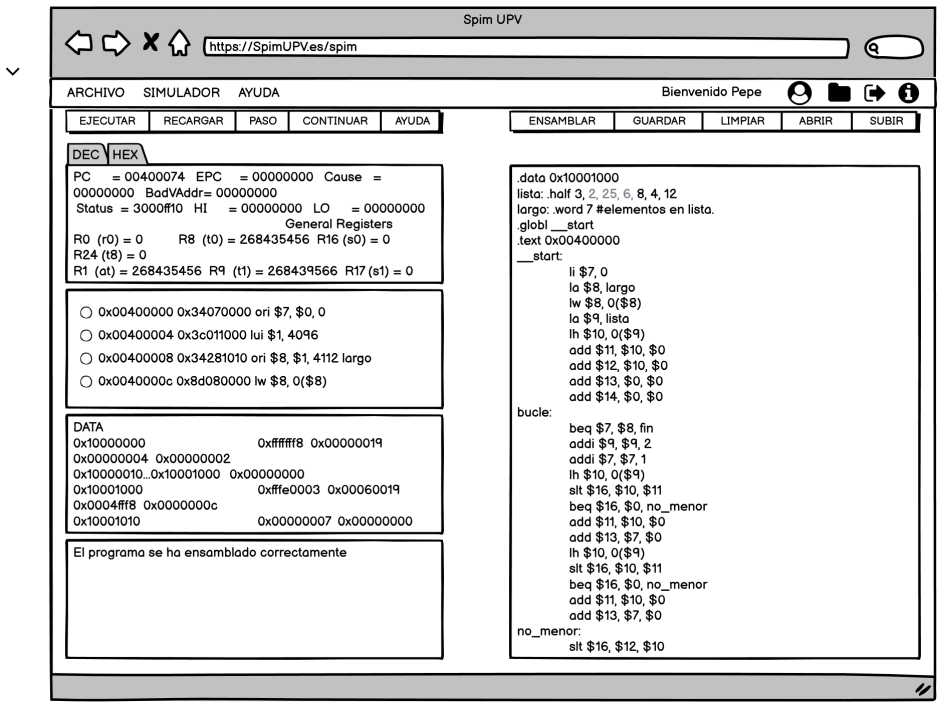

Figura 5.4: Wireframe: Página principal del simulador para el usuario registrado

Página de inicio de sesión: En la figura 5.5 podemos observar el prototipo para la página de inicio de sesión.

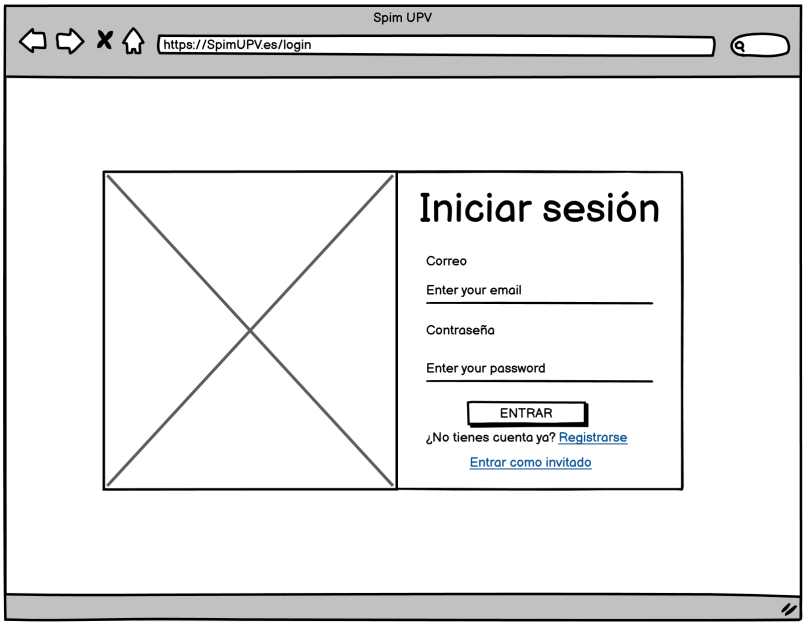

Figura 5.5: Wireframe: Página inicio de sesión

Página de registro: En la figura 8.3 podemos observar el prototipo para la página de registro.

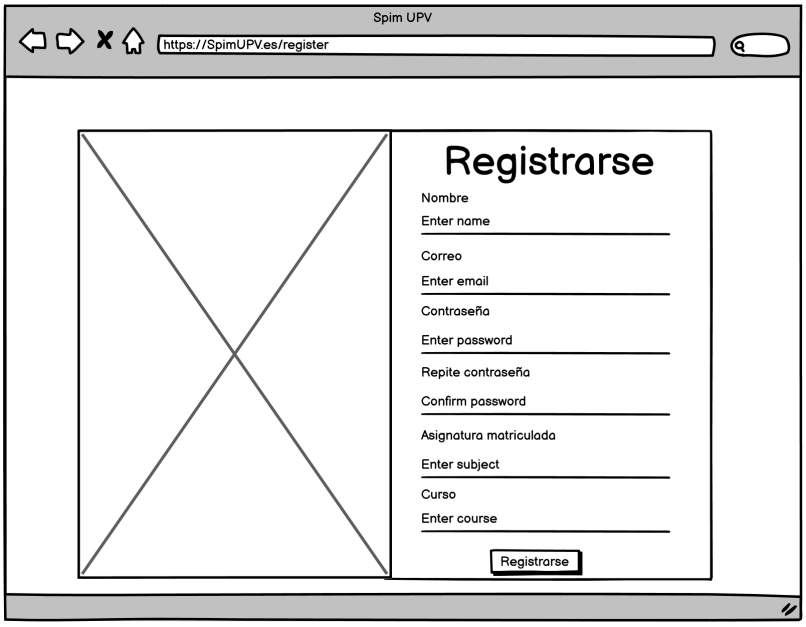

Figura 5.6: Wireframe: Página de registro

Página para restablecer la contraseña: En la figura 5.7 podemos observar el prototipo para la página del restablecimiento de contraseña.

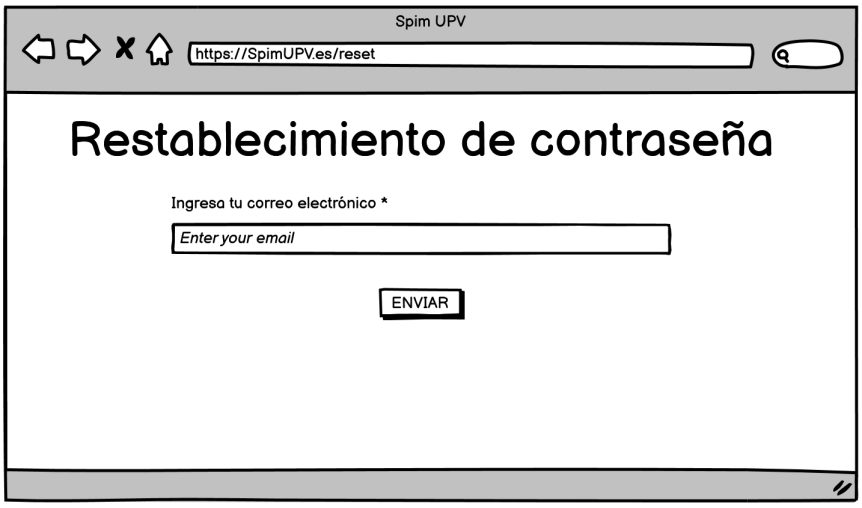

Figura 5.7: Wireframe: Página para restablecer la contraseña

Página para cambiar la contraseña: En la figura 5.8 podemos observar el prototipo para la página de edición de la contraseña del usuario.

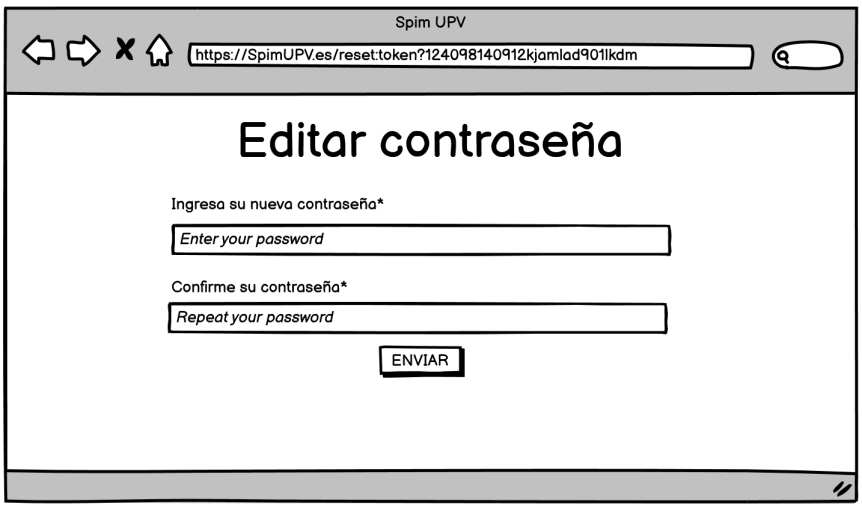

Figura 5.8: Wireframe: Página para cambiar la contraseña

Ventana de información: En la figura 5.9 podemos observar el prototipo para la ventana de información.

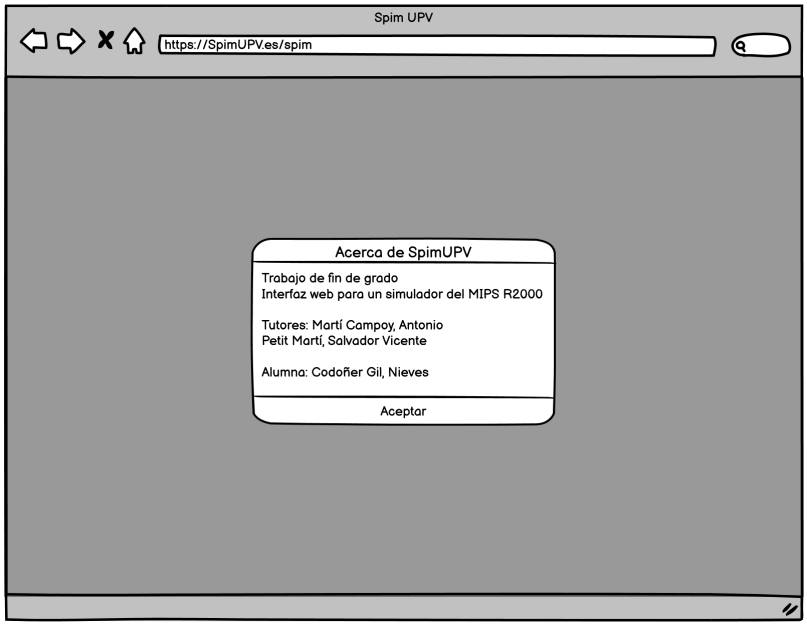

Figura 5.9: Wireframe: Ventana de información.

Ventana de información del usuario registrado: En la figura 5.10 podemos observar el prototipo para la ventana de información del usuario *logueado* que contiene su nombre, correo, curso y asignatura matriculada.

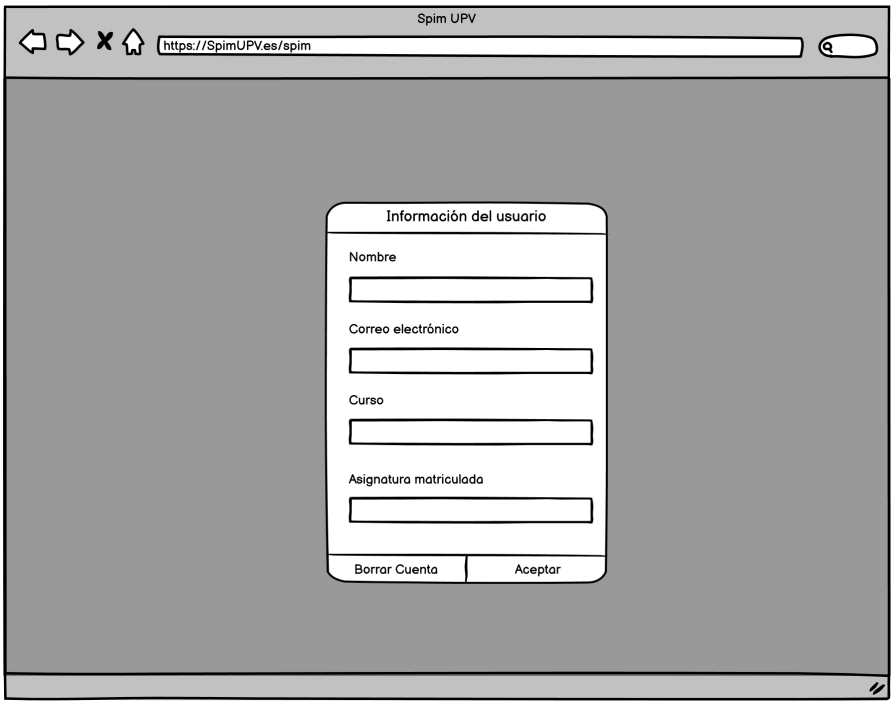

Figura 5.10: Wireframe: Ventana de información del usuario registrado

Ventana de ayuda: En la figura 5.11 podemos observar el prototipo para la ventana de ayuda de la aplicación, contiene una explicación de las diferentes funcionalidades que ofrece la web.

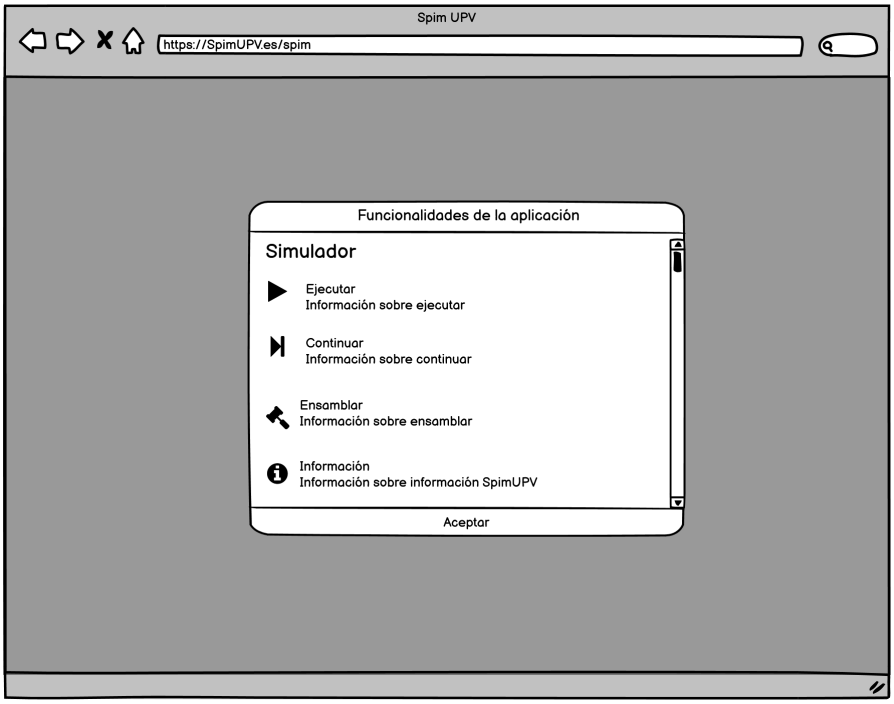

Figura 5.11: Wireframe: Ventana de ayuda

Diálogo para subir el programa del editor: En la figura 5.12 podemos observar el prototipo para permitir que el usuario pueda subir al servidor el programa que ha escrito en el editor de código.

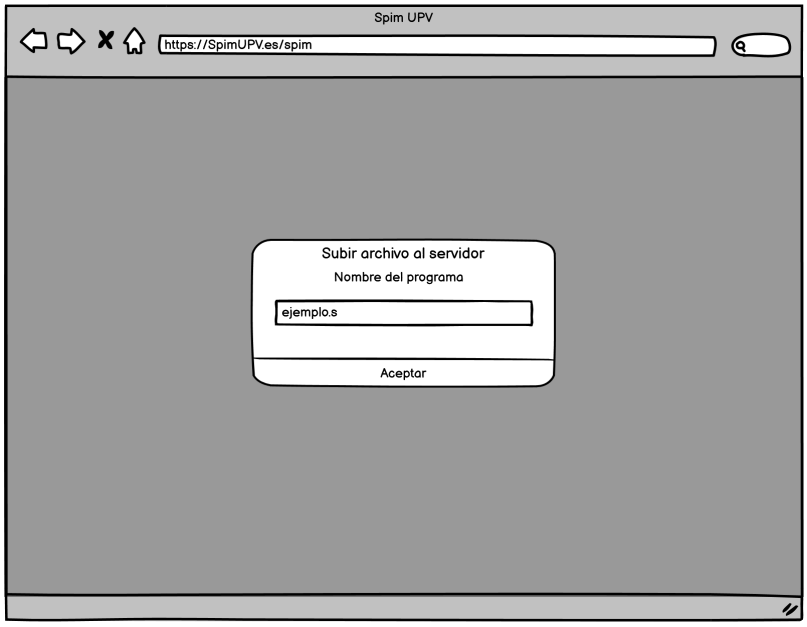

Figura 5.12: Wireframe: Diálogo para la subida del programa del editor

Ventana para elegir la dirección de ejecución: En la figura 5.13 podemos observar el prototipo para la ventana en la que el usuario elegirá la dirección de memoria inicial de ejecución del programa.

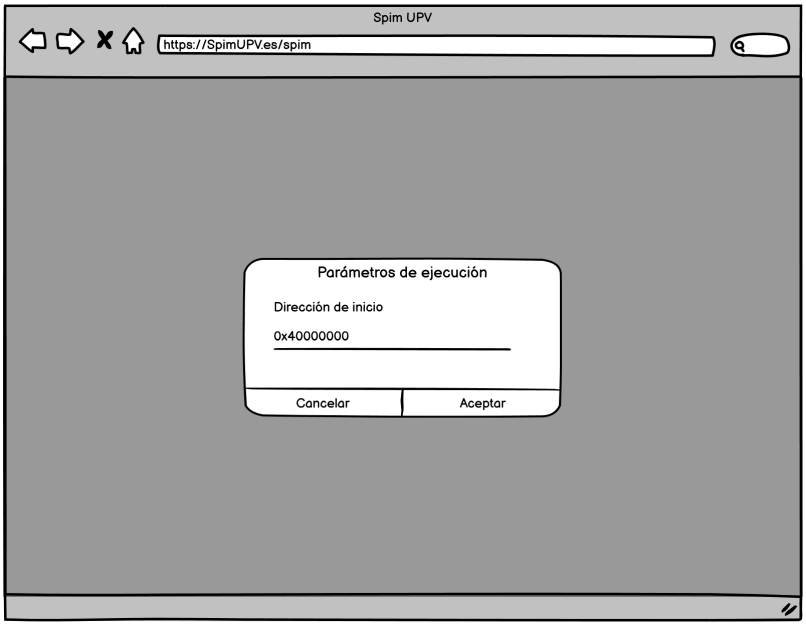

Figura 5.13: Wireframe: Ventana de ejecución.

Ventana para establecer puntos de ruptura: En la figura 5.14 podemos observar el prototipo para ventana donde el usuario elegirá la dirección donde quiere establecer un punto de ruptura.

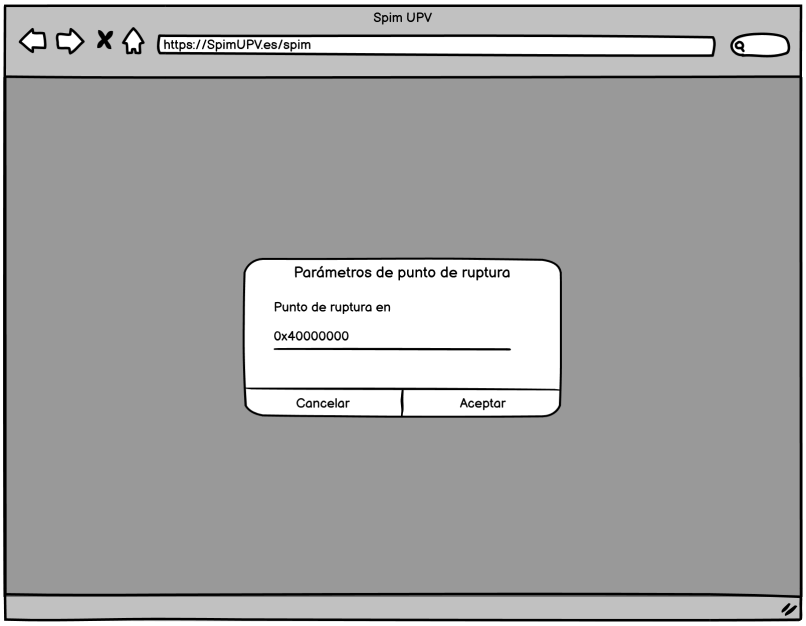

Figura 5.14: Wireframe: Ventana puntos de ruptura

Alerta de error del simulador: En la figura 5.15 podemos observar el prototipo para la alerta de error que el simulador lanza cuando hay algún problema o fallo en el programa que se intenta ensamblar.

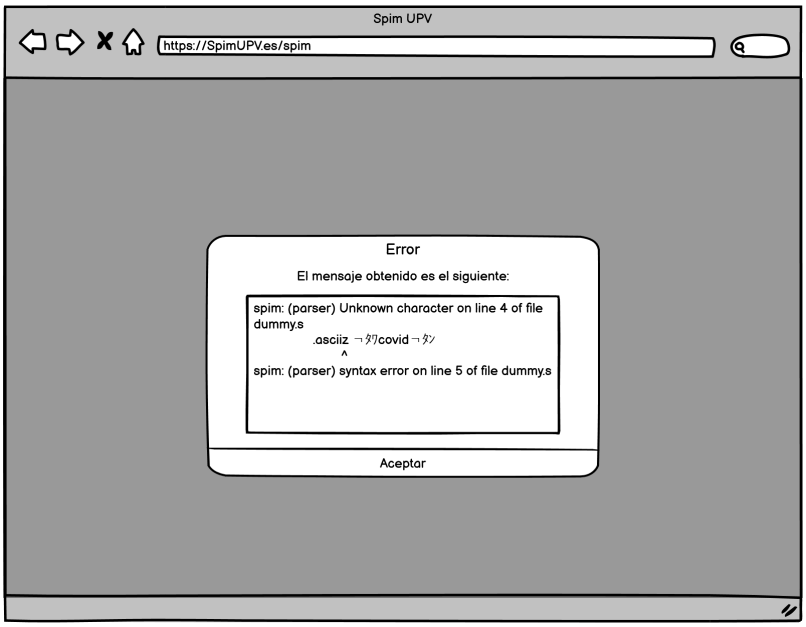

Figura 5.15: Wireframe: Alerta de error

Modal para la gestión de archivos del servidor: En la figura 5.16 podemos observar el prototipo para la gestión de archivos del servidor que se muestra cuando el alumno pulsa en el icono representado por una carpeta de usuario.

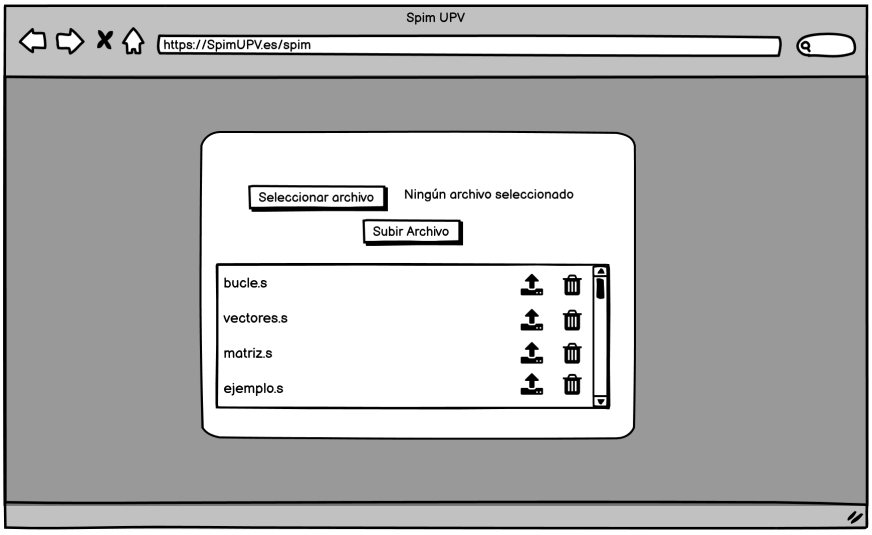

Figura 5.16: Wireframe: Modal para la gestión de archivos del servidor

CAPÍTULO 6
Tecnologías utilizadas

En este capitulo se van a detallar una serie de tecnologías aplicadas en el proyecto y el motivo de la elección de las mismas.

6.1 Entorno

La elección de un entorno de desarrollo es esencial para conseguir facilitar el proceso a un programador de software. Se ha optado por el editor de código fuente Visual Studio Code [14] como podemos observar en la figura 6.1. Entre las opciones que nos ofrece este editor están:

- Depuración de código
- *Plugins* descargables que facilitan la escritura de código (temas, auto-completados, *live share*...).
- Refactorización de código.
- Resaltado de sintaxis

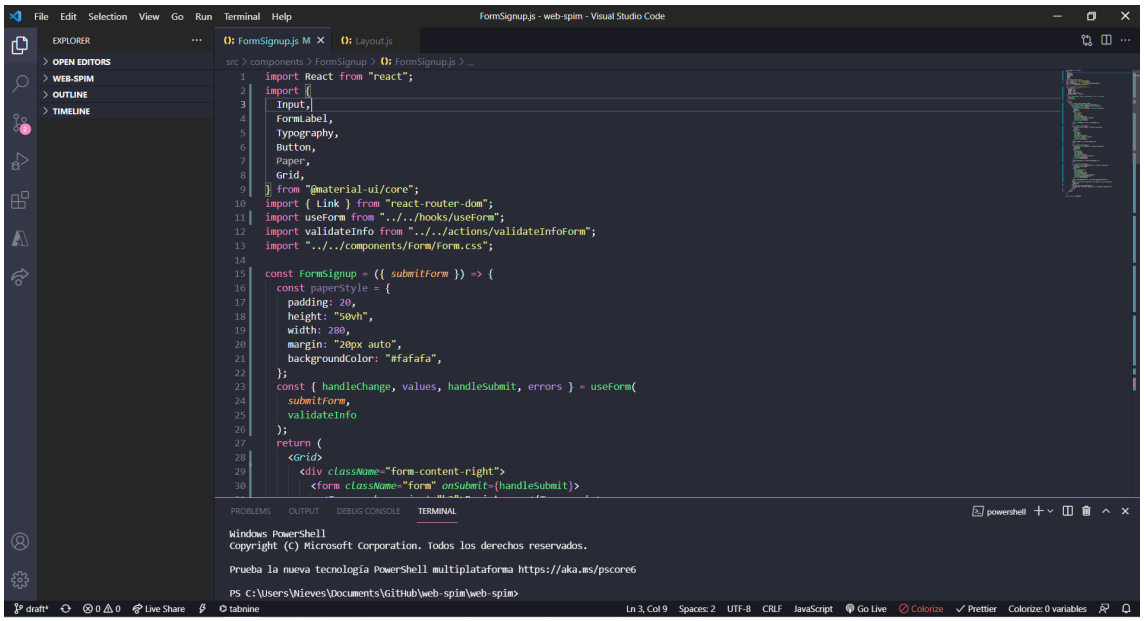

Figura 6.1: Editor de código fuente Visual Studio Code

Por otro lado, se ha utilizado Git [15] como sistema de control de versiones, siendo GitHub [16] la plataforma elegida para alojar el proyecto. Esta decisión recae en la familiaridad que esta nos ofrece, añadiendo además el acceso gratuito (por ser estudiante) a GitHub PRO[1]

Además, es necesario mencionar que se ha utilizado una GUI (GitHub Desktop [18]), para facilitar la interacción con GitHub. Así, el programador puede evitar el uso de la consola y agilizar de esta manera el proceso de desarrollo de la aplicación.

Por último, y para facilitar el trabajo, se ha creado un tablón en la plataforma Trello, pudiendo de esta manera apuntar todas las tareas pendientes para que así, de un día para otro estén claros los objetivos o el trabajo diario que hay que añadir en la aplicación. Asimismo, se han creado etiquetas para saber de un simple vistazo a que parte del desarrollo apunta cada una. En la figura 6.2 podemos observar una serie de columnas como por ejemplo:

- **To do:** Tareas pendientes que se tienen que hacer, se van apuntando conforme se van necesitando.
- **In progress:** Tareas que se están realizando en el momento, una vez que esta ha finalizado se manda a la columna Done.
- **Done:** Tareas finalizadas.
- **Bug:** Problemas pendientes que pueden ir surgiendo en el desarrollo y que requieren un proceso de reflexión para conseguir solucionarlos.
- **Fixed bug:** Problemas resueltos.

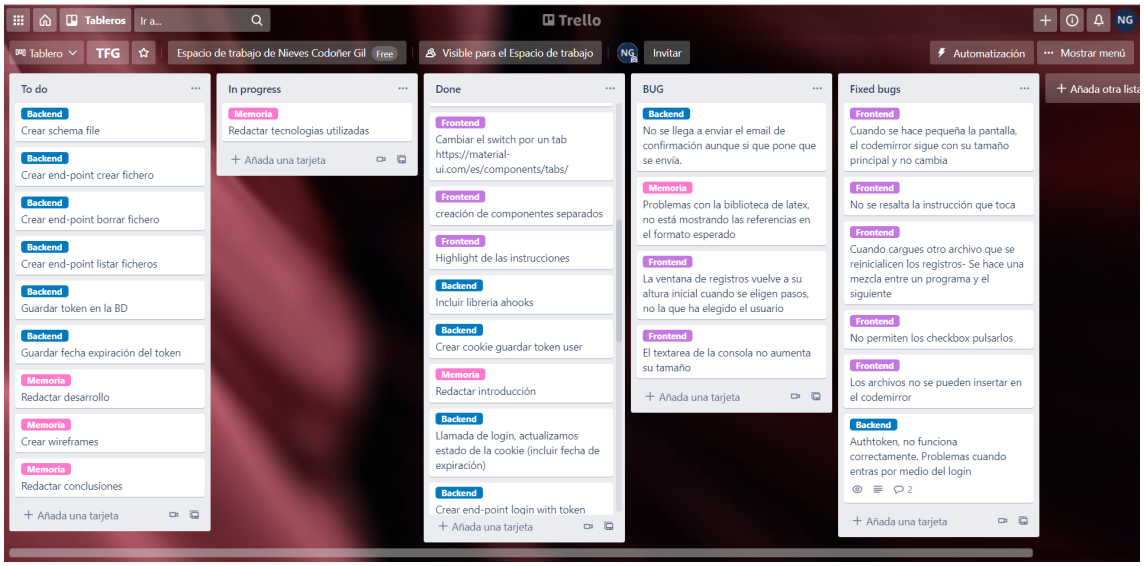

Figura 6.2: Plataforma Trello para la organización de las tareas.

6.2 Lenguaje de desarrollo

Para este proyecto se ha optado por el lenguaje de programación interpretado JavaScript [19],ya que cuenta con una comunidad activa y es sencilla su implementación con

[1]GitHub PRO es un plan de uso que ofrece GitHub y que tiene una serie de ventajas tales como: ramas protegidas, páginas y wikis... Para más información consultar [17]

HTML y CSS [20]. Además, en la asignatura Tecnología de sistemas de información en la red (TSR) se ha aprendido a programar desde cero en JavaScript, contando de esta manera con un conocimiento previo y ahorrando tiempo en el aprendizaje.

6.3 Tecnología

La elección de las tecnologías del proyecto ha sido uno de los procesos más importantes para el desarrollo, ya que escoger unas u otras tecnologías puede marcar la diferencia tanto en el proceso como en el resultado final. De este modo, las tecnologías utilizadas siguen un conjunto de subsistemas software denominados MERN Stack que incluye las tecnologías: React.js para el frontend, Express.js y Node.js para el backend y MongoDB para base de datos. En la figura 6.3 se puede observar la comunicación que maneja este subsistema MERN.

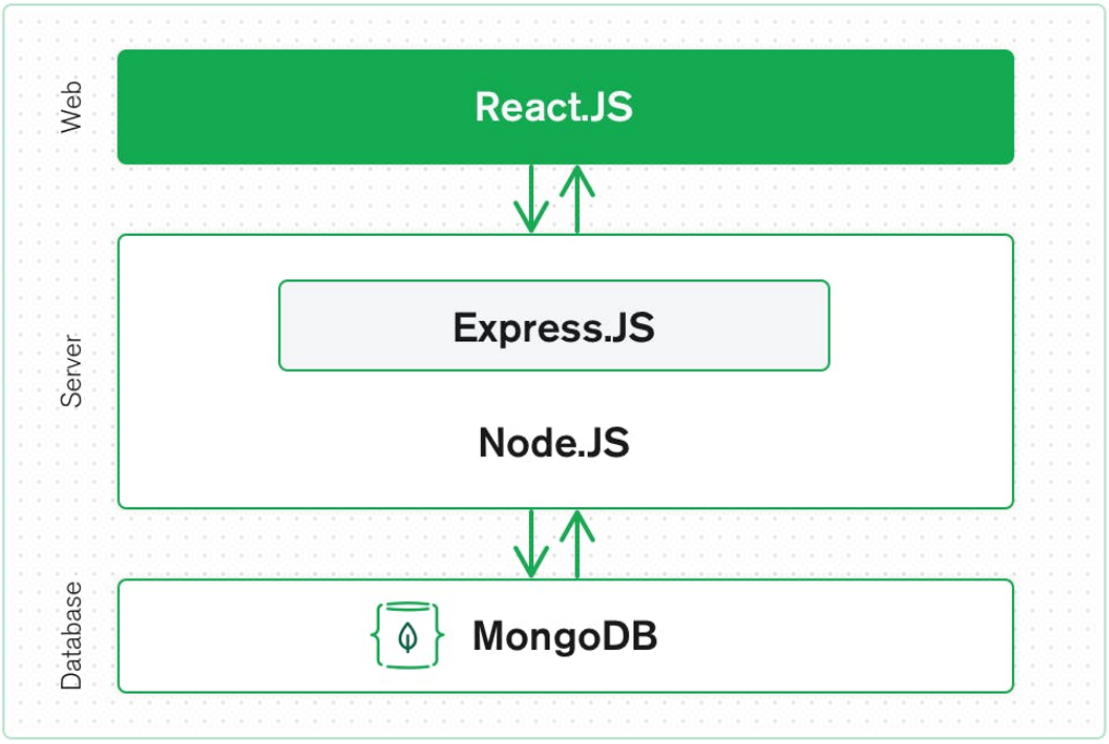

Figura 6.3: MERN stack [21]

Además, todas estas herramientas están basadas en Javascript, permitiendo al programador que las maneja unificar el lenguaje de programación tanto para el lado del servidor como en el lado del cliente, proporcionando facilidades en cuanto al aprendizaje de las tecnologías.

En los siguientes apartados se profundizará en estas tecnologías y se darán los motivos por los que han sido elegidas, añadiendo a ellas el transcompilador Emscripten para lograr embeber el simulador SPIM en la aplicación Web.

6.3.1. Emscripten

Emscripten [22] es una herramienta de compilación para *WebAssembly*. Este, se encarga de compilar código C o C++, devolviéndonos como salida un programa en JavaScript

o HTML. Esta herramienta es necesaria para el proyecto, ya que partimos del simulador spim en C++ y por medio de Emscripten lo portamos a JS para ejecutarlo en un worker y comunicarnos con este.

6.3.2. Frontend

En el presente apartado se mostrarán una serie de tecnologías utilizadas para el desarrollo del frontend.

React

React [23] es una libreria de JavaScript de código abierto creada por Facebook y que busca facilitar el desarrollo a los programadores que hagan uso de ella, ayudándoles a construir interfaces de usuario de manera sencilla [24]. Además, existen múltiples guías e incluso entornos de programación ya configurados para que el proceso de aprendizaje sea más rápido. Uno de los entornos que existe se llama *Create React App* [25] que permite al programador tener una cadena de herramientas ya configuradas que añaden funcionalidades como por ejemplo: ejecutar la aplicación en modo desarrollador con el comando *npm start* (accediendo a través del navegador al servidor virtual: http://localhost:3000), *npm test* para las pruebas o *npm run build* para construir y optimizar lo máximo posible la aplicación para el proceso de producción. Por otro lado, este entorno no se encarga ni de la lógica del backend ni de la bases de datos, con lo cual el programador puede hacer uso del que prefiera.

Existen otras alternativas como Angular [26] y Vue.js [27]. Dado que no existia experiencia previa en ninguna de las tecnologías y herramientas, y todas presentan ventajas e inconvenientes, se eligió React tras apreciar que la curva de aprendizaje podía ser menor gracias a la facilidad de encontrar ejemplos y ayuda en su comunidad.

Material-Ui

Cabe destacar que para agilizar el proceso de desarrollo de la web se ha utilizado el lenguaje de diseño *Material UI* [28],ya que proporciona algunos componentes para React ya creados con una funcionalidad y un estilo definido, aunque estos son totalmente personalizables.

Otras librerías

- **Codemirror [29]:** Librería que proporciona un editor de texto como componente de React. Este componente tiene características muy básicas, como el tipo de lenguaje de programación que va a ser incluido o el tema visual que se le añade.
- **Axios [30]:** Librería que proporciona la posibilidad de crear peticiones HTTP a través de promesas simples para node.js.
- **Ahooks [31]:** Librería que proporciona la posibilidad de guardar un valor o estado en una cookie por medio de la función *useCookieState*.

6.3.3. Backend

En el presente apartado se mostrarán una serie de tecnologías utilizadas para el desarrollo del backend.

Node.js

Node.js es un entorno de ejecución de JS para el backend de una aplicación, siendo multiplataforma y caracterizándose por ser de código abierto. Asimismo, una de las ventajas más significativas con las que cuenta Node.js es que logra unificar las aplicaciones web en torno a un solo lenguaje de programación, siendo este JS. De esta forma, tanto el backend como el frontend tienen el mismo lenguaje de programación, facilitando de esta forma el proceso de desarrollo del programador.

Por otro lado, Node cuenta con npm, un sistema de gestión de paquetes. En este proyecto se hará uso de esta herramienta, y en los siguientes puntos se listarán una serie de módulos utilizados:

- **dotenv:** Módulo que sirve para cargar las variables de entorno del archivo ".env" en el "process.env", al cuál se accederá para utilizar las variables.
- **mongoose:** Módulo que permite trabajar con la base de datos de MongoDB.
- **joi:** Módulo encargado de la descripción de esquemas y de la validación de los datos.
- **bcryptjs:** Módulo que permite encriptar la información: en nuestro caso las contraseñas de los usuarios, para que una vez guardadas en el servidor, nadie pueda acceder a estos datos. Bycrypt ofrece una función de *hashing* que trabaja con un valor llamado salt. Este es un fragmento aleatorio que sirve para generar el *hash* asociado a las contraseñas y así conseguir que dos contraseñas iguales no generen el mismo *hash*.
- **jsonwebtoken:** Modulo que permite generar tokens para el inicio de sesión.
- **cors:** Módulo basado en el mecanismo intercambio de recursos de origen cruzado o **CORS** (*Cross-origin resource sharing*), que nos permite realizar peticiones desde otros dominios, en nuestro caso: desde el cliente al servidor.
- **express:** Módulo que integra express en Node.js.
- **express-fileupload:** Módulo que funciona como *middleware* para la carga de archivos.
- **nodemailer:** Módulo que permite el envío de correos electrónicos en la aplicación de manera sencilla.

Express.js

Express.js es un *framework* para Node.js que proporciona un conjunto de características tanto para las aplicaciones web como móviles. Además, contiene miles de métodos que utilizan HTTP y *midldleware*. Express.js está caracterizado también por ser una herramienta ligera, rápida y flexible que será de gran ayuda para completar los objetivos que requiere este proyecto.

6.3.4. Despliegue y pruebas

Heroku

Heroku [32] es una plataforma en la nube que permite subir aplicaciones web en diferentes lenguajes de programación (Ruby, Java, PHP, NodeJS...). Además, cuenta con

una versión gratuita [33] para la subida de proyectos personales que ofrece una serie características óptimas, algunas de las cuales son: posibilidad de despliegue desde Git y Docker, personalización de dominios,una memoria RAM de 512MB para cada aplicación ejecutada, parcheo automático del sistema operativo...

En proyecto se utilizará la plataforma Heroku para alojar el servidor que se desarrollará.

Netfily

Netfily [34] es una plataforma que permite automatizar la subida de páginas web a través de repositorios de plataformas tales como GitHub o GitLab. Además, cuenta con una versión gratuita [35] para la subida de proyectos personales que ofrece una serie características óptimas, algunas de las cuales son: compilación automática desde Git, número de páginas ilimitadas, posibilidad de revertir a cualquier versión,UI conjunta para la colaboración de varios miembros de un mismo equipo...

En el proyecto se utilizará la plataforma Netfily para alojar el cliente que se desarrollará.

Postman

Postman [36] es una plataforma que permite crear peticiones HTTP (con parámetros si es necesario) a un servidor e inspeccionar los resultados sin la necesidad de la creación de un cliente. Esta herramienta se ha utilizado para hacer pruebas directamente al servidor y comprobar que su comportamiento es el esperado.

6.3.5. Redacción de la memoria

Con el fin de la creación de esquemas se ha hecho uso de la herramienta draw.io [12] que permite crear diagramas en linea de forma gratuita. Por otro lado, se ha utilizado Balsamiq [13] para la creación de los *wireframes*, utilizados en el apartado de diseño de la interfaz de la aplicación.

Para concluir con esta sección cabe destacar que se ha utilizado el procesador de textos LaTeX [37] para la redacción de la memoria. Concretamente se ha hecho uso del editor de texto Overleaf, que emplea LaTeX y ofrece control de versiones, plan gratuito, librerías para facilitar el desarrollo de la memoria... Además, la Escuela Técnica Superior de Ingeniería Informática (ETSINF) ofrece una plantilla con el formato que se requiere en el proyecto. Los paquetes utilizados han sido:

- **biblatex:** paquete que ayuda en el procesamiento de la información bibliográfica de la memoria.
- **float:** paquete que que proporciona la posibilidad de colocar varios elementos en una sola página (ya sean tablas o incluso imágenes).
- **csquotes:** paquete que proporciona funciones avanzadas para las citas, admitiendo para ello una gama de comandos y citas definibles por el usuario.
- **longtable:** paquete que proporciona la posibilidad de crear tablas que ocupen diferentes páginas.
- **hyperref:** paquete que proporciona comandos para la referencia de capítulos o secciones.

- **color:** paquete que proporciona la posibilidad de definir colores e implementarlos en la memoria. Se ha utilizado para resaltar frases o párrafos que había que revisar.

CAPÍTULO 7
Desarrollo de la solución propuesta

En el presente capitulo se expondrá el proceso de desarrollo del proyecto y se plantearán los diferentes problemas que han ido surgiendo.

7.1 Preparación del simulador

En el presente apartado se expondrán las características que nos ofrece el simulador elegido, su funcionamiento en rasgos generales y los ajustes mínimos que tendremos que introducir en el mismo para lograr integrarlo en una aplicación web.

7.1.1. Spim, simulador de consola

Como el propio título indica, Spim es un simulador de consola creado por James R. Larus y que a día de hoy puede ser descargado desde el siguiente enlace `https://sourceforge.net/p/spimsimulator/code/HEAD/tree/spim/`. En el se pueden encontrar tres archivos: uno que contiene toda la funcionalidad del simulador (escrito en C++), un *Makefile* cuyo objetivo es compilar el programa y un *readme* que ofrece información sobre los archivos de la carpeta y de lo relativo al copyright, tal y como muestra la figura 7.1.

File	Date	Author
Makefile	2020-06-02	jameslarus
README	2010-09-04	Jim
spim.cpp	2016-11-25	jameslarus

Figura 7.1: Archivos del simulador de consola spim

Para comprobar el correcto funcionamiento del simulador de consola, una vez han sido descargados los archivos citados, se procede a compilar el simulador a través del archivo Makefile, utilizando para ello el comando *make*. Cuando este proceso termine, se ejecuta en la consola el simulador mediante el comando *./spim*. El comando de ayuda

(*help*) del simulador de consola permite averiguar cuales son las funcionalidades ofrecidas por el simulador. En la figura 7.2 se pueden observar los diferentes comandos soportados por el simulador.

```
SPIM is a MIPS32 simulator.
Its top-level commands are:
exit  -- Exit the simulator
quit  -- Exit the simulator
read "FILE" -- Read FILE containing assembly code into memory
load "FILE" -- Same as read
run <ADDR> -- Start the program at (optional) ADDRESS
step <N> -- Step the program for N instructions (default 1)
continue -- Continue program execution without stepping
print $N -- Print register N
print $fN -- Print floating point register N
print ADDR -- Print contents of memory at ADDRESS
print_symbols -- Print all global symbols
print_all_regs -- Print all MIPS registers
print_all_regs hex -- Print all MIPS registers in hex
reinitialize -- Clear the memory and registers
breakpoint <ADDR> -- Set a breakpoint at address ADDR
delete <ADDR> -- Delete breakpoint at address ADDR
list -- List all breakpoints
dump [ "FILE" ] -- Dump binary code to spim.dump or FILE in network byte order
dumpnative [ "FILE" ] -- Dump binary code to spim.dump or FILE in host byte order
. -- Rest of line is assembly instruction to execute
<cr> -- Newline reexecutes previous command
? -- Print this message

Most commands can be abbreviated to their unique prefix
e.g., ex(it), re(ad), l(oad), ru(n), s(tep), p(rint)

(spim)
```

Figura 7.2: Comandos que ofrece el simulador Spim

Entre otras funcionalidades, el usuario a través de la interfaz de consola puede ensamblar programas, ejecutarlos y acceder a los datos de los registros.

7.1.2. Adaptación del simulador

Una vez estudiado el simulador y las funcionalidades que este ofrece se procederá a adaptarlo a las demandas que se requieren en el proyecto. Cabe destacar que se podría haber hecho todo el proceso con fuerza bruta, es decir, reescribiendo desde cero todo el código directamente en JS, pero esta solución requeriría un gran esfuerzo y no daría garantías de que su ejecución fuera completamente correcta y fiel al comportamiento de SPIM, simulador verificado por su uso por un gran número de usuarios y años de desarrollo y mejora.

La solución escogida ha sido Emscripten para transcompilar el código a JS y así integrarlo en la aplicación Web. Las principales ventajas de esta manera de proceder son las siguientes:

- El compilador incluido en Emscripten garantiza que la funcionalidad del código JS generado es la misma que la del simulador original, el cuál es bien conocido y ha sido validado por una amplia cantidad de docentes y académicos.
- Permite que en este TFG se trabaje con una tecnología estándar para la transcompilación de múltiples aplicaciones programadas en C/C++ a JS. Por ejemplo, diversas

empresas de motores de videojuegos utilizan Emscripten para realizar este tipo de traducciones de código [38].

- Permite alcanzar uno de los objetivos principales del trabajo, mover la carga computacional de la simulación al lado del cliente Web.

Instalación y explicación de emsdk

Como ya se ha explicado en el capitulo 6, Emscripten es un compilador de código que se utilizará para transformar el programa de spim de consola escrito en C++ a JavaScript. Esta herramienta puede ser descargada en el siguiente enlace `https://emscripten.org/docs/getting_started/downloads.html` y además cuenta con una serie de guías de instalación e información de como hacer uso de ella.

A grandes rangos, Emscripten utiliza en Linux los comandos ./emcc (para el lenguaje C) y ./em++ (para C++) como remplazo a los compiladores estándar tales como gcc o clang. Además, es preciso señalar que la herramienta nos proporciona la posibilidad de que el usuario añada una serie de parámetros en la linea de compilación para que, de este modo, se puedan incorporar al proceso diferentes funcionalidades, algunas de las cuales se explican en el siguiente enlace `https://emscripten.org/docs/tools_reference/emcc.html`.

Transcompilación del simulador

Una vez familiarizados con el transcompilador y después de haber hecho una serie de pruebas compilando proyectos de dimensiones menores, se ha procedido a modificar el Makefile. De esta forma, se han añadido en la linea de compilación una serie de parámetros listados a continuación y que se pueden observar en la figura 7.3

1. *"-o js_$@.js":* Definición del tipo de salida que esperamos, en este caso un archivo en JS con el nombre js_spim.

2. *"WASM=0":* Parámetro que deshabilita el WebAssembly, emitiendo de esta manera solamente el código en JavaScript.

3. *"FORCE_FILESYSTEM=1":* Parámetro para habilitar el soporte a sistema de archivos y entrada/salida estándar en el programa transcompilado.

4. *"EXPORTED_FUNCTIONS="["_js_spim_main","_js_spim_top_level"]":* Parámetro para exportar las funciones "_js_spim_main()" y "_js_spim_top_level()" del código del simulador y de que esta manera sean accesibles desde JavaScript ejecutándose en el navegador.

5. *"EXPORTED_RUNTIME_METHODS="["ccall","cwrap"]":* Parámetro para exportar los métodos del runtime de Emscripten ccall (que permite llamar directamente a una función del código transcompilado) y cwrap (que permite construir funciones JavaScript a las que se puede llamar para acceder a funciones del código transcompilado).

```
CXX = em++
CXXFLAGS += -I. -I$(CPU_DIR) $(DEFINES) -O -g -Wall -pedantic -Wextra -Wunused -Wno-write-strings -x c++
YCFLAGS +=
LDFLAGS += -lm
CSH = bash

# lex.yy.cpp is usually compiled with -O to speed it up.

LEXCFLAGS += -O $(CXXFLAGS)



OBJS = spim.o spim-utils.o run.o mem.o inst.o data.o sym-tbl.o parser_yacc.o lex.yy.o \
       syscall.o display-utils.o string-stream.o


spim:   $(OBJS)
    $(CXX) -g $(OBJS) $(LDFLAGS) -o js_$@.js -s SINGLE_FILE -s WASM=0 -s FORCE_FILESYSTEM=1 -s
    EXPORTED_FUNCTIONS='["_js_spim_main", "_js_spim_top_level"]' -s EXPORTED_RUNTIME_METHODS='["ccall","cwrap"]'
```

Figura 7.3: Parámetros añadidos en la compilación del simulador

Tras haber ejecutado el Makefile, y sabiendo que la compilación ha sido satisfactoria, buscaremos en nuestra carpeta el archivo “js_spim.js"que utilizaremos más tarde.

Modificaciones al código del simulador

Para poder utilizar el código transcompilado como *worker* de JS en nuestra aplicación, es necesario realizar modificaciones mínimas en el código fuente del simulador de consola (programado en C++) y en el resultado de la transcompilación (fichero “js_spim.js”).

En el código original en C++ del simulador se ha modificado la función principal “main” cambiando su nombre por “js_spim_main” y evitando que entre en un bucle (llamado *top_level* en el codigo original del simulador) que espera un comando por la entrada estándar. La nueva versión de la función simplemente inicializa las estructuras internas del simulador y devuelve el control para que nuestra aplicación pueda realizar otras tareas.

Para permitir que el simulador interprete un comando por la entrada estándar, se ha añadido al código del simulador la función “js_spim_top_level”. Cuando se llama a esta función se realiza una iteración del mencionado bucle top_level. En cada iteración de este bucle, el simulador lee un comando por la entrada estándar (*stdin*), lo ejecuta, e imprime el resultado por la salida estándar (*stdout*) y la salida de error (*stderr*). Una vez “js_spim_top_level” ejecuta una iteración, devuelve el control a nuestra aplicación para que pueda procesar el resultado.

Además, se han añadido soporte a dos nuevos comandos (*print_data* y *print_text*) al simulador para que pueda proporcionar por la salida estándar información global sobre el segmento de datos y el segmento del código. Estos comandos simplemente permiten acceder a funciones de spim que no estaban disponibles en la versión de consola.

En lo que respecta al resultado de la transcompilación (fichero “js_spim.js”), es necesario incluir 3 funciones JS para alimentar la entrada estándar del código transcompilado y almacenar lo que este código emite por la salida estándar y de error. La entrada estándar se alimenta con los comandos generados desde la aplicación, mientras que la salida y el error se almacenan en cadenas de carácteres que son procesadas por la aplicación.

Finalmente, puesto que el código en el fichero “js_spim.js” se utiliza dentro de un worker JS, es necesario añadir un manejador de eventos que reciba las peticiones de la interfaz de nuestra aplicación. Estas peticiones son comandos para el simulador que son procesadas por la función “js_spim_top_level”. El worker simplemente prepara la entra-

da estándar (es decir, el comando) para esta función y recoge los resultados emitidos por la salida estándar y de error para devolverlos a la interfaz. Su funcionamiento se explica con detalle en el apartado 7.2.2.

7.2 Frontend

7.2.1. Preparación del entorno

Una vez realizada la compilación del simulador, pasaremos a la preparación del entorno para la parte del cliente.

En primer lugar, crearemos un repositorio privado en GitHub para almacenar el proyecto, se crearán las ramas pertinentes y se empezarán a subir los primeros archivos para el frontend.

Como ya se explicó en el apartado 6.3.2, para el proyecto se ha utilizado un entorno de programación ya configurado denominado *Create React App*, por medio del comando: *"npx create-react-app spim-app"*. Esta estructura será modificada en el desarrollo, siendo en la sección 7.2.3, dónde se hará más hincapié sobre la estructura final del proyecto y la organización del mismo.

Por último, se instalarán las diferentes librerías y se importarán en el código para hacer uso de ellas.

7.2.2. Implementación del *worker*

Por definición, un *worker* es un *script* que se ejecuta en un segundo plano y no interfiere con la interfaz de usuario. De esta forma, si no se hiciese uso de los *workers* la página web se volvería no responsiva [39] ya que el usuario no podría interactuar con el interfaz hasta que el script finalizara la ejecución. Asimismo, se puede establecer comunicación con un *worker* a través de mensajes con los métodos postMessage (que envía una cadena o objeto JSON) y onmessage (que recibe lo que le haya enviado el método postMessage) [40].

En JavaScript se pueden denominar también *Web Workers*, y para crearlos se necesita llamar al constructor *"Worker()"*, donde se le especificará la URI del *script* que se desea ejecutar (en nuestro caso la del fichero que contiene el simulador). Para intercambiar información entre el interfaz y el worker se utilizan los métodos onmessage y postmessage.

Una vez efectuada la investigación del funcionamiento de los *Web Workers* y después de realizar una serie de pruebas más sencillas pasaremos a modificar el *worker* de nuestro proyecto.

En primer lugar se incluirá el archivo del simulador al proyecto que hemos creado en GitHub. En nuestro programa donde se implementa la interfaz web de usuario del simulador ("Spim.js") llamaremos al constructor antes citado y le pasaremos la *URI* donde se localiza el simulador. Además, guardaremos el *worker* en una variable global que pasaremos a todos los componentes que lo requieran, creado para ello lo que se conoce como un contexto,y proveyendo a todos los hijos de este componente ("Spim.js") su posible acceso al *worker*.

De esta forma, cuando el usuario solicite (mediante la interfaz Web) realizar una acción con el simulador, el hilo principal enviará el comando SPIM correspondiente al worker, utilizando postMessage.

En el worker se crea un método “onmessage()"que será el encargado de ejecutar el simulador y que recibirá todos los comandos que sean solicitados del hilo principal. De esta forma, el proceso que seguirá la comunicación será la siguiente:

1. El hilo principal mandará un mensaje a través del metodo “postMessage()", que contendrá el comando de spim asociado a la acción que ha realizado el usuario en la interfaz.

2. El worker recibirá en el método “onmessage()“ un mensaje del hilo principal. Verifica que la ejecución del comando anterior ha finalizado, si no es así espera a que esté libre. Si sí ha finalizado ejecuta el simulador con el comando recibido.

3. Cuando el simulador finaliza la ejecución del comando envía un mensaje con la información correspondiente al hilo principal. Como ya se ha comentado anteriormente, se recogerá la salida estándar y la de error de la respuesta que haya dado el simulador y se enviará directamente al hilo principal.

4. El hilo principal recibe la respuesta y guarda los resultados en las variables de entorno pertinentes, cambiando de esta forma el contenido de los elementos de la interfaz web del simulador.

Además, cabe destacar que se ha creado un archivo que almacena todas las peticiones que se pueden solicitar al simulador, en diferentes funciones para evitar tener que repetir código en cada uno de los componentes y lograr así tener el código más centralizado.

En la figura 7.4 se puede observar su comportamiento en forma de gráfico.

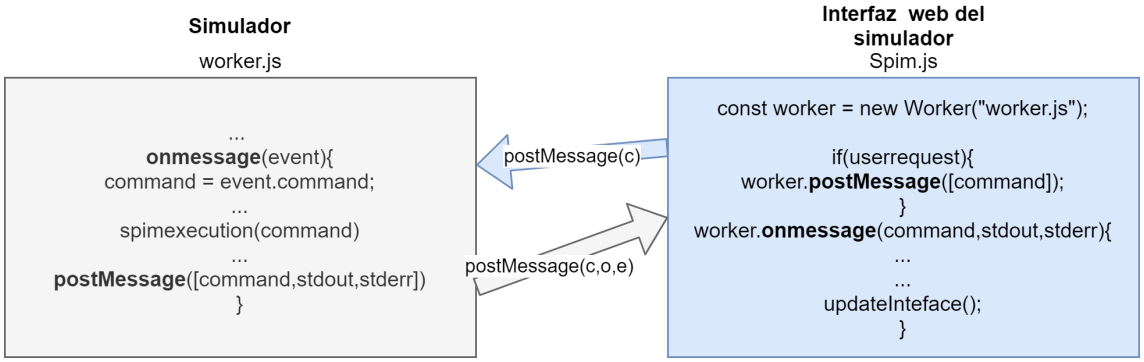

Figura 7.4: Comunicación del *worker* con el programa

7.2.3. Distribución de la estructura de archivos

En la presente sección se describirán las diferentes carpetas y archivos que conforman el proyecto. En la figura 7.6 se puede observar de forma global la estructura.

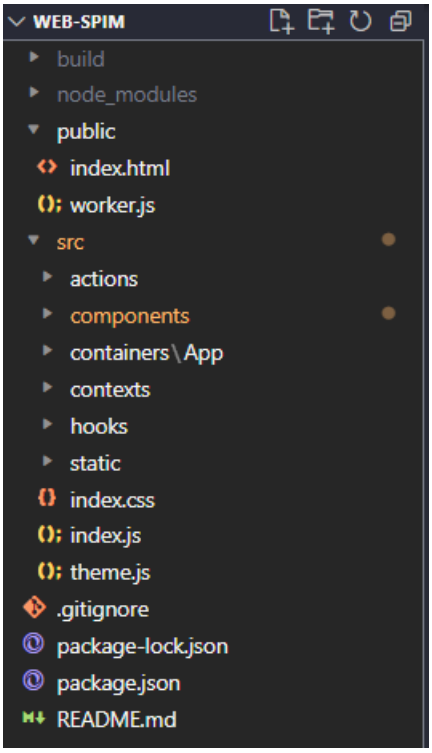

Figura 7.5: Estructura global del proyecto en el *frontend*

A continuación se va a proceder a listar las diferentes carpetas y después se detallarán aquellas que son más importantes:

- **Carpeta *build*:** En ella irá todo lo relacionado con la fase de despliegue de la del cliente. Esta carpeta se creará desde consola cuando añadamos el comando *"npm run build"*.

- **Carpeta *node-modules*:** En ella se descargarán todos los módulos de Node y cuando sean referenciados en el código, se buscarán en esta carpeta.

- **Carpeta *public*:** En ella se guardan tanto la plantilla de la página, como el *worker* del simulador que ha sido configurado.

- **Carpeta *src*:** En ella se guardan una serie de subdirectorios dónde se encuentra el desarrollo de la interfaz de usuario: los componentes y diferentes funciones que más tarde nombraremos.

También existen los archivos: "gitignore" para ignorar aquellas carpetas o archivos que sean indicados y que no se suban a la plataforma GitHub, un"README.md" para explicar a rasgos generales el proyecto y un "package-lock.json" donde se guardan las dependencias o los paquetes del proyecto.

En los siguientes apartados se desglosará la carpeta **scr** para lograr así entender la estructura del proyecto:

Carpeta *actions*:

Como podemos ver en la figura 7.6, la carpeta cuenta con tres archivos:

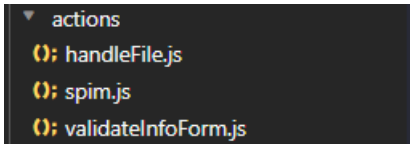

Figura 7.6: Estructura global del proyecto en el *frontend*

- **handleFile.js:** es una función que importa el fichero del sistema de archivos del cliente, para que de este modo se pueda guardar en una variable y cargarlo en el editor de código.

- **spim.js:** contiene un conjunto de funciones que componen un mensaje con la información correspondiente a la acción del usuario en la interfaz, y luego se llama a una función que conforma la estructura del "postmessage", de esta manera solo tendremos que llamar a la función desde nuestro código y no habrá que escribir en todo momento la estructura del "postMessage" y pasarle el comando. Además, algunas funciones se llaman entre si para crear nuevos comportamientos como por ejemplo: recargar un programa requiere tanto que se limpien los registros como que se vuelva a ensamblar el programa en el simulador.

- **validateInfoForm.js:** es una función que se encarga de validar toda la información que nos proporciona el usuario a la hora de registrarse. Si hay algún problema, nos devolverá un objeto con los diferentes errores que ha encontrado, si no, este objeto estará vacío.

Carpeta *components*:

Los componentes en React son piezas independientes y reutilizables que permiten separar la interfaz de usuario y contar con su propio estado, propiedades y lógica. Cada componente en el proyecto estará en una carpeta donde podremos observar tanto un archivo JS que se refiere al componente propiamente dicho, como de un archivo CSS que se encarga de los estilos del componente. En la figura 7.7 podemos observar una serie de ejemplos de componentes que han sido creados para el proyecto.

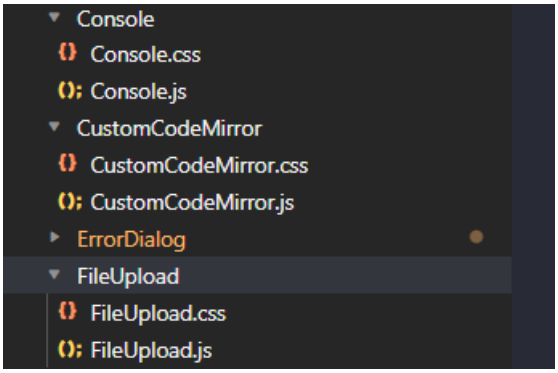

Figura 7.7: Ejemplo de algunos componentes creados en el proyecto

Carpeta *containers*:

Dónde se encuentra el contenedor principal de la aplicación. Este se encarga de ir cambiando entre las diferentes rutas del proyecto: simulador, registro e inicio de sesión.

Carpeta *hooks*:

Contiene la función para guardar en estados los valores que el usuario envía cuando se registra. También se guarda si ha habido problemas de validación, pasándole toda esta información al registro para que reaccione conforme a esos resultados.

Carpeta *context*:

Contiene todos los estados globales de la aplicación como pueden ser: valores de entrada del editor de código, errores que retorna el simulador cuando se ha intentado ensamblar un programa, el segmento de datos, el de texto, los registros ... Esta estructura sirve para tener un estado global al cual todos los componentes que señalemos, podrán proveerse de estos datos y de esta manera no tener que estar repitiendo continuamente en cada uno de los ficheros estados locales que luego se tendrán que pasar por las "props" o propiedades (cómo si fuesen los parámetros que se reciben en una función de JS) de un componente a otro.

Carpeta *static*:

Esta carpeta contiene el programa ensamblador que se cargará por defecto en el editor de código y los iconos e imágenes que se utilizarán en el desarrollo de la interfaz.

7.3 Backend

7.3.1. Preparación del entorno

Una vez expuesto el desarrollo del *frontend*, se procederá a la preparación del entorno para el servidor y la base de datos.

Así pues, crearemos una carpeta en el mismo repositorio dónde está almacenado el cliente. Una vez creado, se procederá a instalar los diferentes módulos de Node que serán utilizados y los importaremos en el código. En la figura 7.8 podemos observar como se han importado los diferentes módulos en el archivo principal (cabe destacar que en esa imagen no están todos los que se utilizan en el proyecto, la lista de todos ellos se localiza en el apartado 6.3.3 de tecnologías).

```
const express = require('express');
const app = express();
const mongoose = require('mongoose');
const dotenv = require('dotenv');
const cors = require('cors');
const fileUpload = require('express-fileupload');
```

Figura 7.8: Módulos importados en el archivo principal del servidor

Paralelamente a la creación del proyecto, crearemos una cuenta en la página de MongoDB Atlas (en el apéndice A se puede observar el proceso que se ha seguido para la creación de la cuenta) que será la base de datos donde se alojará toda la información referente a los usuarios y los programas en ensamblador que los usuarios almacenen.

7.3.2. Distribución de la estructura de archivos

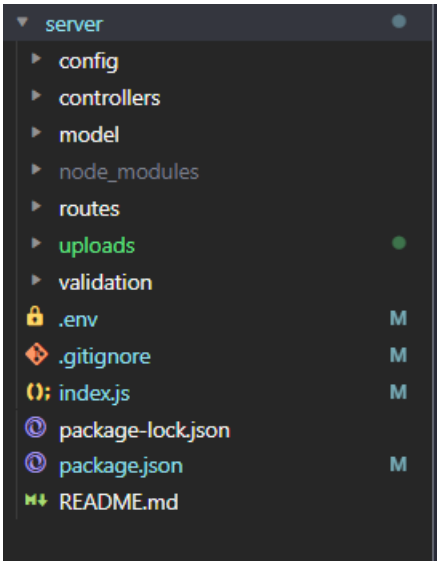

Figura 7.9: Estructura de archivos del servidor

En las siguientes secciones se procederá a detallar el contenido de las carpetas y archivos [1] del proyecto. En la figura 7.9 se puede observar una visión global del proyecto.

Carpeta *controllers*:

Se detallarán los controladores de la aplicación encargados de tratar las peticiones de los usuarios. Como podemos ver en la figura 7.10, contamos con dos controladores: uno para el usuario registrado y otro para los archivos.

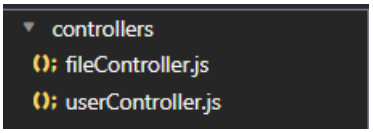

Figura 7.10: Carpeta controllers

Carpeta *models*:

Se detallarán los modelos para la base de datos con la ayuda de moongose. Como podemos ver en la figura 7.11, contamos con dos modelos: uno para el usuario registrado y otro para los archivos que almacenan los programas de ensamblador de los usuarios.

Figura 7.11: Carpeta models

[1] Cabe destacar que tanto la carpeta node-modules, los package.lock.json y el README.md tiene el mismo cometido que en el apartado del cliente, con lo que se procederá a obviarlos de la explicación

En la figura 7.12 podemos observar un ejemplo del modelo que sigue “File.js". Los campos que aparecen en el modelo están explicados en el apartado 5.2 de la memoria.

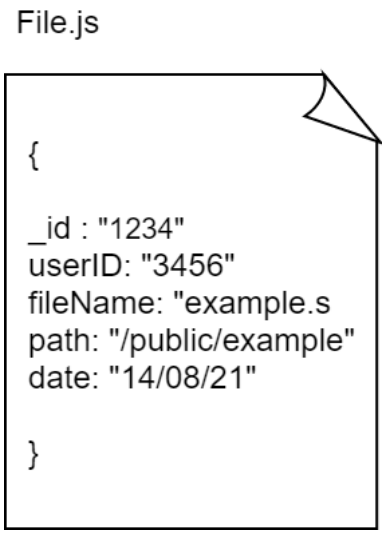

Figura 7.12: Modelo para los programas en ensamblador

En la figura 7.13 podemos observar un ejemplo del modelo que sigue “User.js", la linea *password* estará encriptada ya que no es seguro guardar las contraseñas en la base de datos sin cifrar. Los campos que aparecen en el modelo están explicados en el apartado 5.2 de la memoria.

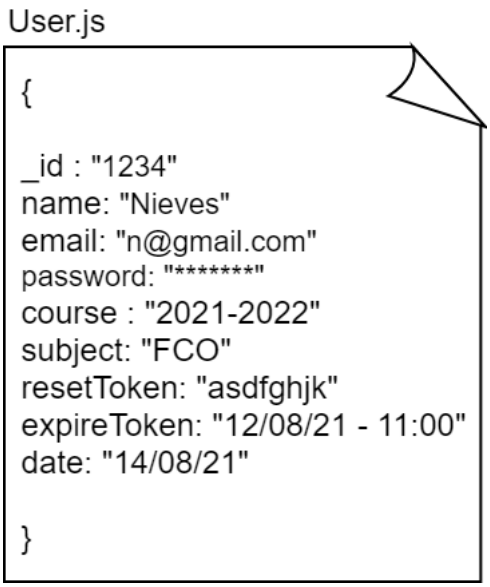

Figura 7.13: Modelo para los usuarios

Carpeta *uploads*:

Una vez el usuario suba un archivo a través del cliente, el servidor creará un sistema de archivos, donde cada carpeta dentro de “uploads” estará destinada a un usuario diferente y en cada una se almacenarán los programas que el usuario ha elegido subir.

Carpeta routes:

Se detallan las diferentes rutas montables y modulares que servirán a la aplicación. En el direccionador de la aplicación (“index.js") habrá que cargar estos módulos para que de esta manera se puedan manejar solicitudes en las rutas. En la figura 7.14 se pueden observar las diferentes rutas de la aplicación:

- **auth.js:** almacena todas las rutas referentes a la gestión de usuarios (registrarse, iniciar sesión, restablecer contraseña, nueva contraseña y cerrar sesión).
- **file.js:** almacena todas las rutas referentes a la gestión de los ficheros (añadir fichero, listar ficheros, obtener fichero, borrar fichero, borrar ficheros).
- **verifyToken.js:** almacena la ruta referente a la comprobación del token de acceso del usuario a la aplicación.

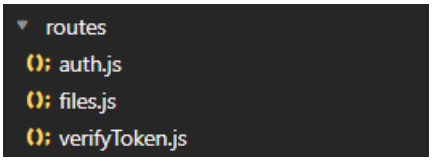

Figura 7.14: Carpeta routes

Carpeta *validation*:

Carpeta que contiene el archivo de validación para el registro, el inicio de sesión y los archivos almacenados en la aplicación. La validación sirve para comprobar que todos los datos que el cliente envía son correctos y así guardarlos en la base de datos.

Archivo *.env*:

Se trata de un archivo que sirve para guardar las variables de entorno, las cuales estarán *encriptadas* para que ningún usuario pueda acceder a ellas (solo el programador del sitio web). Este archivo contendrá tanto la conexión con la base de datos como la firma secreta para la creación del token de acceso.

Archivo *index.js*:

Se trata de un archivo que se encarga de importar las rutas, realizar la conexión a la base de datos (accediendo a la variable de entorno del archivo .env), establecer el puerto de escucha, habilitar CORS y utilizar los diferentes *middlewares* y módulos que utiliza el servidor.

7.4 Seguridad

En los siguientes puntos se van a detallar algunos aspectos de seguridad que se han implementando tanto en el front-end como en el back-end.

- **Validación en los campos de registro:** Si un usuario se quiere registrar en la aplicación, debe pasar por una primera validación de los datos en el lado del cliente. De

esta forma, se comprueba cada campo a través de una serie de expresiones regulares para que estos tengan el formato establecido.

- **Guardado de la contraseña del usuario en la base de datos:** Guardar la contraseña que el usuario ha introducido en el registro supone un gran problema de seguridad, por ello, se ha procedido a cifrar la contraseña a través de un modulo: "bycrpt". De esta forma, se utilizará la función de *hashing* en la contraseña introducida y se guardará su resultado en la base de datos.
- **Inicio de sesión invalido:** Si un usuario quiere iniciar sesión pero se equivoca en sus datos, el sistema muestra un mensaje de fallo de autenticación general. De esta forma, no da ningún tipo de información, evitando así que se puedan deducir las credenciales de otros usuarios registrados.
- **Inicio de sesión valido:** Si un usuario inicia su sesión, se procede a crear una *cookie* en el navegador que ayuda a averiguar si el usuario está *logueado* en la aplicación. Esta cookie caducará cada 12 horas, pero se eliminará automáticamente si el usuario procede a cerrar su sesión.
- **Restablecer contraseña:** Para que el proceso de cambio de contraseña no afecte a a la vulnerabilidad de la seguridad de la aplicación se ha procedido a realizar una serie de pasos.

  1. Solicitar el correo electrónico con el cual se ha registrado el usuario.
  2. Mandar un *email* con un hipervínculo que contiene un *token* aleatorio, único y con fecha de expiración.
  3. El enlace mandará al usuario a una página para que pueda introducir su nueva contraseña, esta se volverá a *hashear* y se intercambiará por la contraseña en la base de datos.

CAPÍTULO 8
Pruebas

Una vez la aplicación ha sido desarrollada, se ha procedido a realizar una serie de pruebas funcionales para validar su correcto funcionamiento. Este paso es esencial, y es necesario realizarlo antes de la fase de despliegue.

8.1 Pruebas funcionales

8.1.1. Autenticación de usuarios

Registrarse

Para conseguir que un usuario pueda registrarse en la aplicación, debe completar su información de manera correcta. Si el usuario realiza un registro valido, se procede a redirigirlo a la página de inicio, se le señala con un diálogo que su registro ha sido satisfactorio (figura 8.1) y recibe un correo electrónico de bienvenida (figura 8.2). Este diálogo se elimina cuando transcurre un minuto, aunque el usuario puede esconderlo antes por medio del botón que tiene a su derecha.

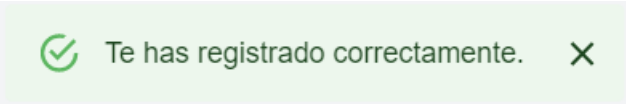

Figura 8.1: Prueba de validación: Registro correcto

Figura 8.2: Prueba de validación: Correo de bienvenida de la aplicación

En la siguiente lista se detallan las diferentes situaciones que se pueden dar en el proceso de registro.

- **Nombre incorrecto:** Si el usuario introduce un número en el campo de nombre, se muestra un mensaje señalándole que el nombre solo puede contener letras. En la figura se puede observar esta situación.

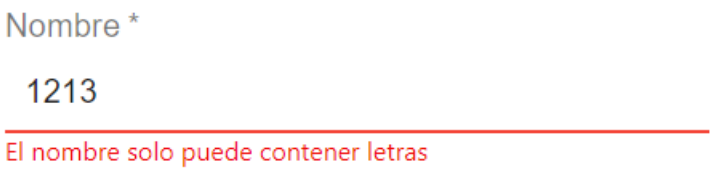

Figura 8.3: Prueba de validación: Registrarse con nombre incorrecto

- **Asignatura matriculada incorrecta:** En este caso se pueden dar dos situaciones: que el usuario introduzca una asignatura con menos de 3 caracteres o una asignatura que contenga números o símbolos. En cualquiera de los dos casos, se muestra un error distinto, tal y como podemos ver en la figura 8.4 y en la 8.5 respectivamente.

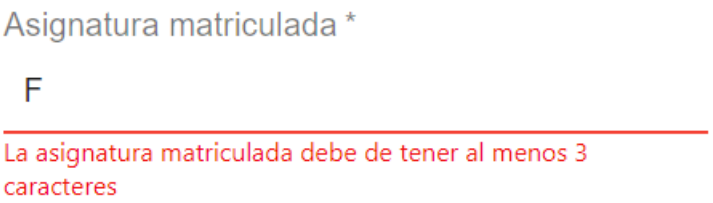

Figura 8.4: Prueba de validación: Asignatura matriculada incorrecta, menos de 3 caracteres

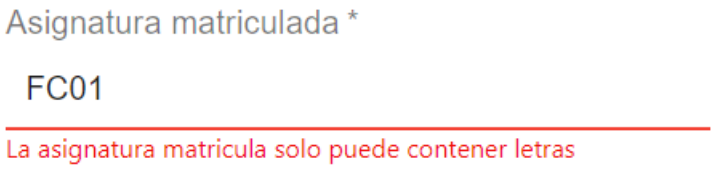

Figura 8.5: Prueba de validación: Asignatura matriculada incorrecta, contiene números y símbolos

- **Contraseña incorrecta:** Si el usuario introduce una contraseña que no contenga al menos 6 caracteres e intenta enviar la información, se le muestra un error, tal y como podemos ver en la figura 8.6. Cabe destacar que en el mismo campo, antes de enviar la información, se le informa al usuario que tiene que cumplir con el mínimo establecido de caracteres para que sea una contraseña valida.

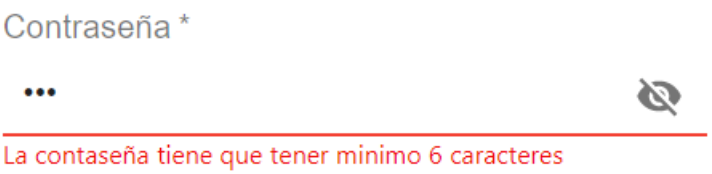

Figura 8.6: Prueba de validación: Contraseña incorrecta, 6 caracteres mínimo

- **Confirmar contraseña incorrecta:** Si el usuario intenta introducir una contraseña diferente a la que ha introducido anteriormente en el campo "contraseña", se le informa a través de un mensaje de error en el campo pertinente. En la figura 8.7 podemos ver representada esta situación.

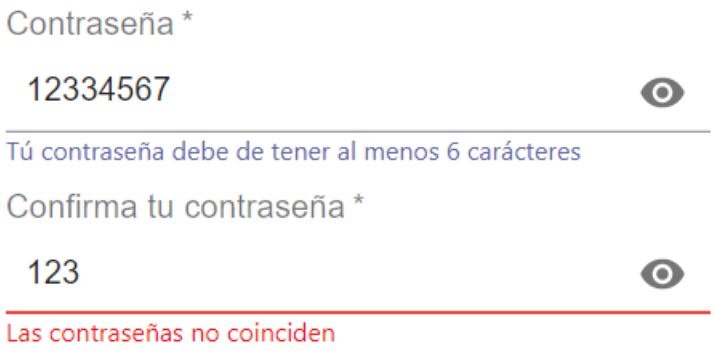

Figura 8.7: Prueba de validación: Confirmar contraseña incorrecta

- **Campos vacíos:** Si el usuario no introduce ningún dato (o deja algún campo sin rellenar) e intenta mandar la información al servidor, este le muestra una serie de errores en todos los campos que ha dejado vacíos, mostrándole que son necesarios esos datos. En la figura 8.8 podemos ver representada esta situación.

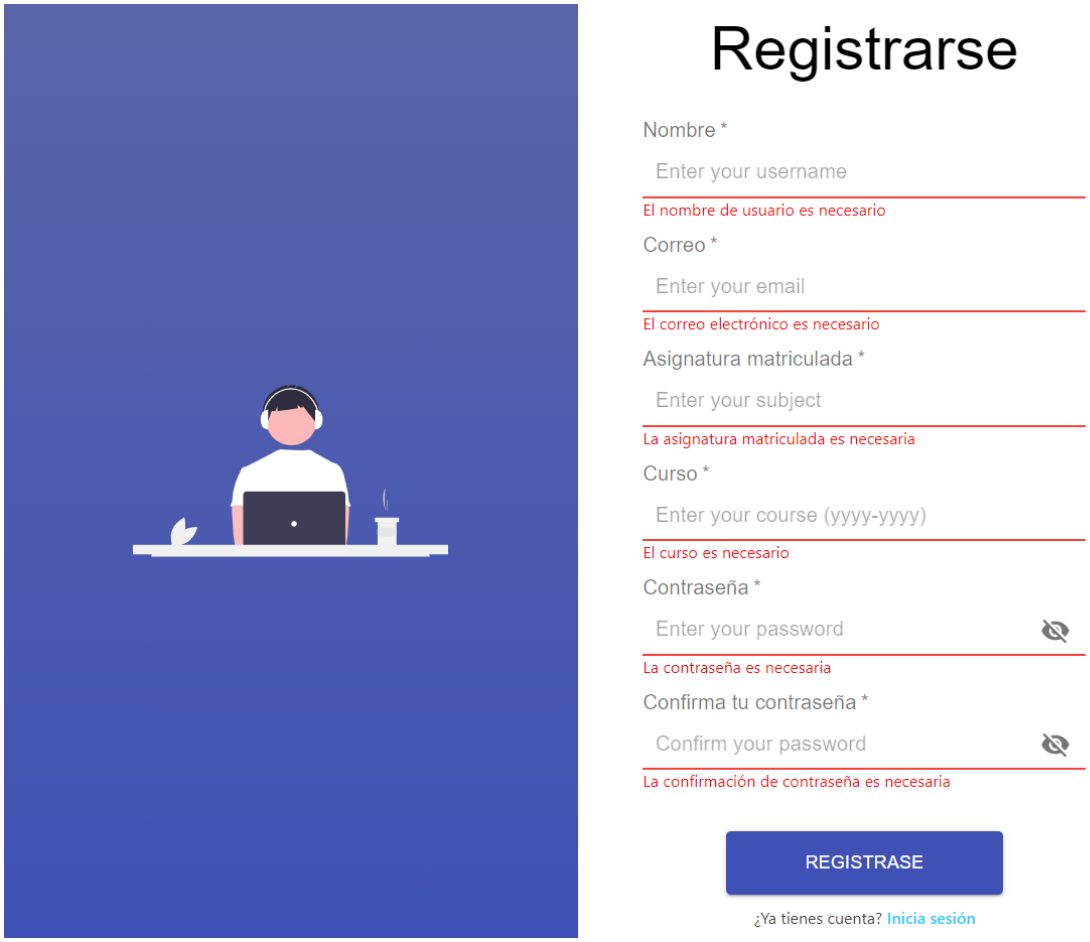

Figura 8.8: Prueba de validación: Registrarse con campos vacíos

Iniciar sesión

En los campos de inicio de sesión, se comprueba que el usuario esté registrado y que la contraseña sea válida para ese mismo usuario. No se hará distinción entre un caso y el otro, todo ello para conseguir una seguridad más robusta, evitando así que un usuario

cualquiera pueda tener información sobre datos de otro alumno. En la figura 8.9 podemos ver representada esta situación. Por el contrario, si el usuario introduce sus credenciales de forma correcta, conseguirá acceder a la página del simulador.

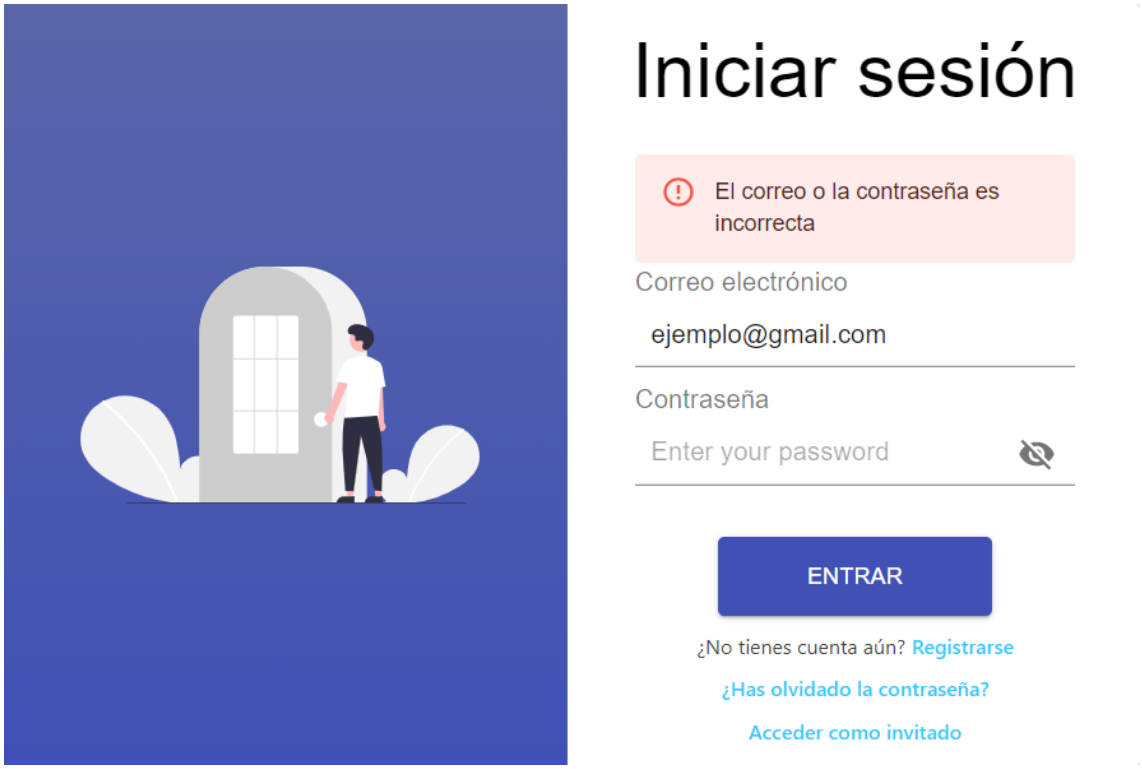

Figura 8.9: Prueba de validación: Contraseña o correo incorrecto

Recuperar contraseña

Si un usuario no recuerda la contraseña, pero si el *email* con el que se ha registrado, tiene la posibilidad de cambiar su contraseña. Para ello, debe acceder a través del hipervínculo que se muestra en la figura 8.10, alojado en la pantalla de inicio de sesión.

Figura 8.10: Prueba de validación: Hipervínculo para restaurar la contraseña

Una vez pulsado, se le redireciona a una página donde tiene que introducir su correo electrónico (figura 8.11) y se muestra un dialogo que indica que el usuario debe revisar el email que ha introducido anteriormente (figura8.12).

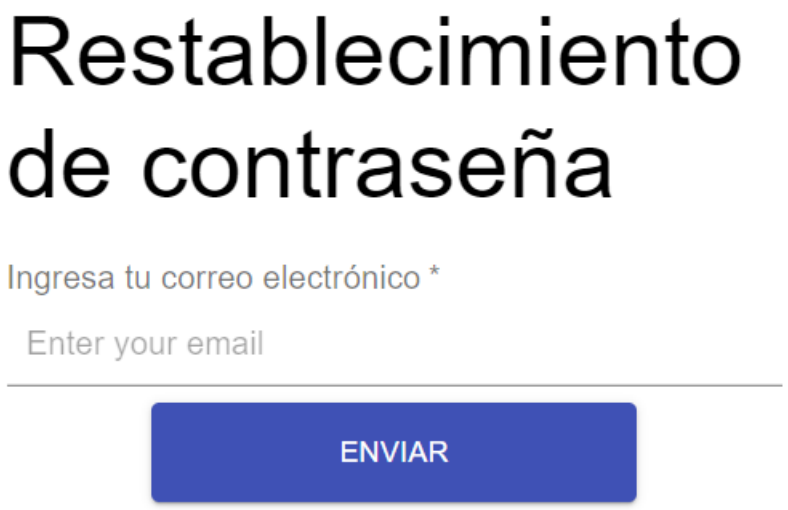

Figura 8.11: Prueba de validación: Página para restablecer la contraseña a través del email

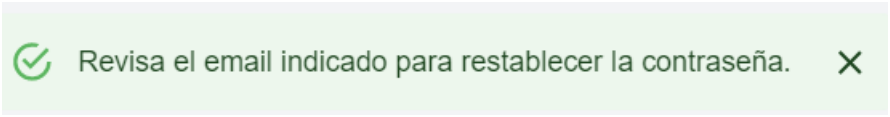

Figura 8.12: Prueba de validación: Diálogo de información, revisar*email*

Si el usuario ha introducido la información correctamente, se le envía un mensaje al correo que ha introducido (figura 8.13). Si no, se le muestra un mensaje de error tal y como muestra la figura 8.14.

Figura 8.13: Prueba de validación: Correo electrónico para cambiar la contraseña

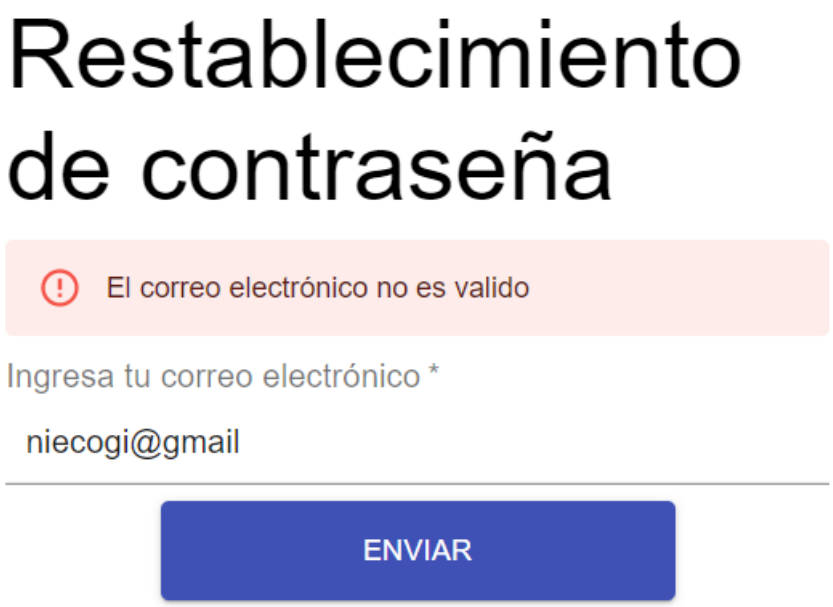

Figura 8.14: Prueba de validación: Correo no válido en el restablecimiento de contraseña

Una vez el usuario pulsa en el enlace que se le ha enviado al correo electrónico, se le muestra una pantalla para que introduzca la nueva contraseña que quiere para su cuenta. Si introduce todos los datos correctos, se redireciona otra vez al inicio de sesión y se le muestra un mensaje para indicar al usuario que se ha procedido a cambiar la contraseña (figura 8.15). Si por el contrario, ha introducido una contraseña sin un mínimo de 6 caracteres o no confirma su contraseña de manera correcta, se le muestra un mensaje de error (figura 8.16).

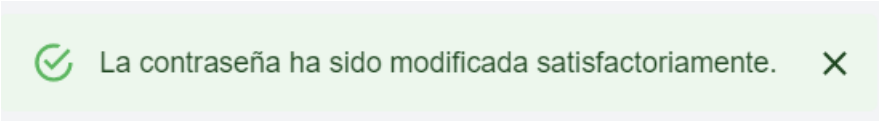

Figura 8.15: Prueba de validación: Mensaje de información, contraseña modificada

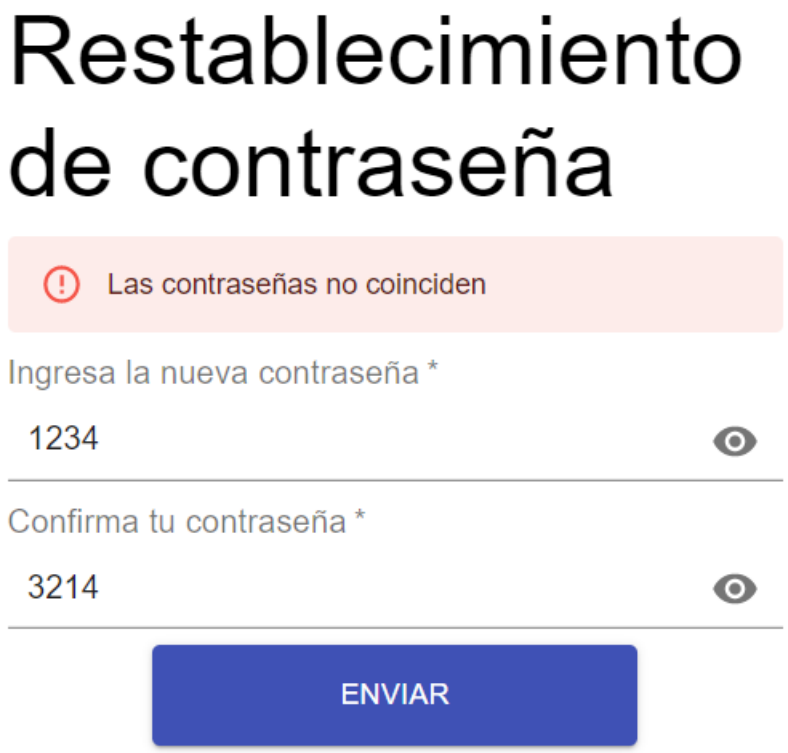

Figura 8.16: Prueba de validación: Correo no válido en el restablecimiento de contraseña

8.1.2. Pantalla redimensionable

Inicio sesión

En la figura 8.17 se puede observar el inicio de sesión que se muestra al usuario para una pantalla que cuenta con 950 o más píxeles de ancho.

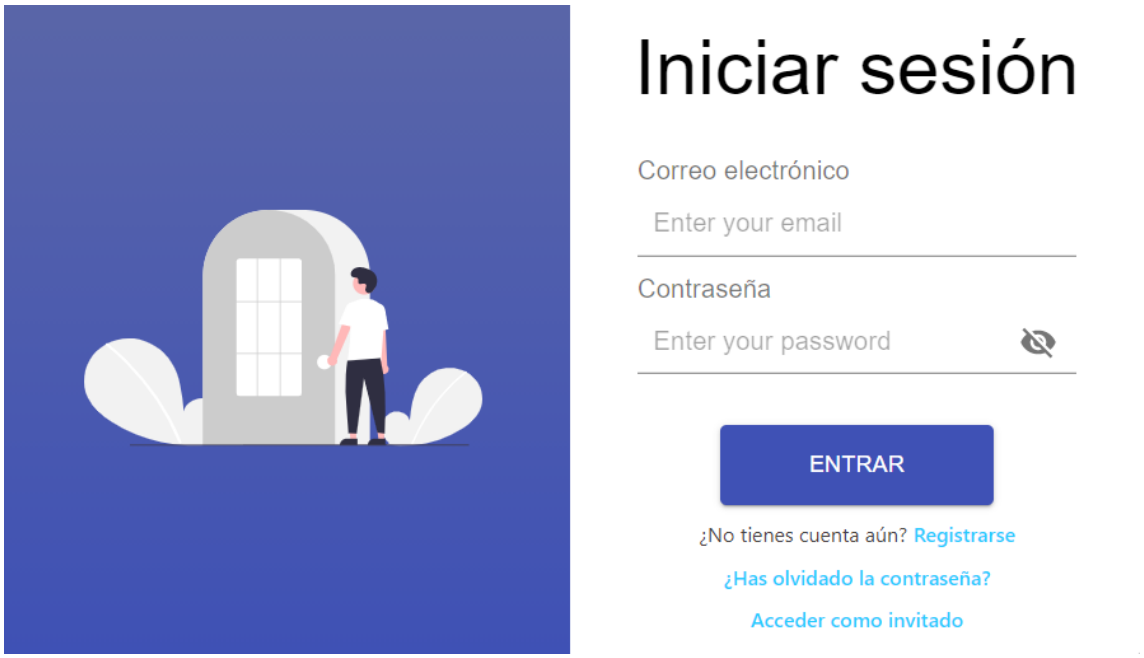

Figura 8.17: Pantalla login en pantallas grandes

En la figura 8.18 se puede observar el inicio de sesión que se muestra al usuario para una pantalla que cuenta con menos de 950 píxeles de ancho.

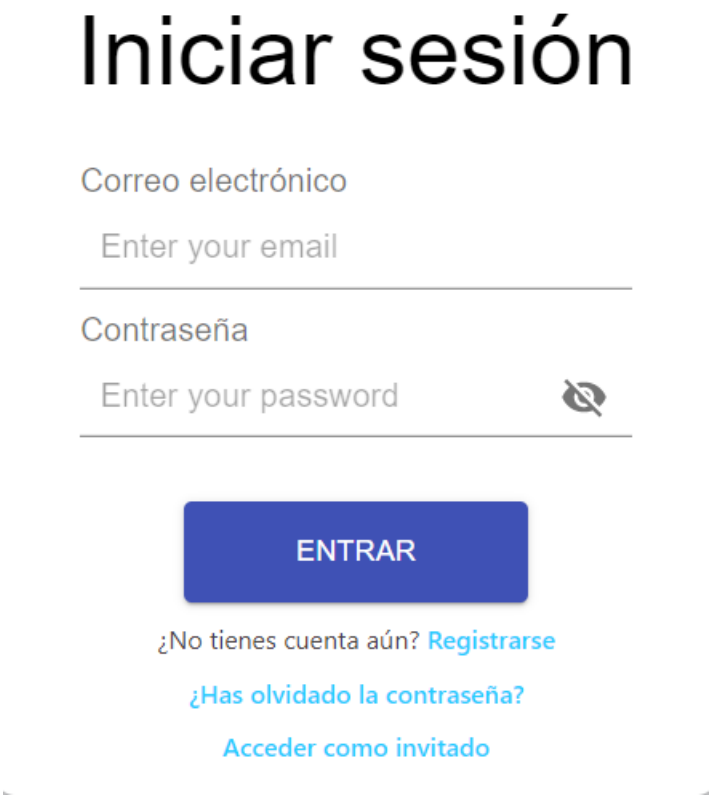

Figura 8.18: Pantalla login en pantallas <950px

Registrarse

En la figura 8.23 se puede observar el registro que se muestra al usuario para una pantalla que cuenta con más de 950 píxeles de ancho.

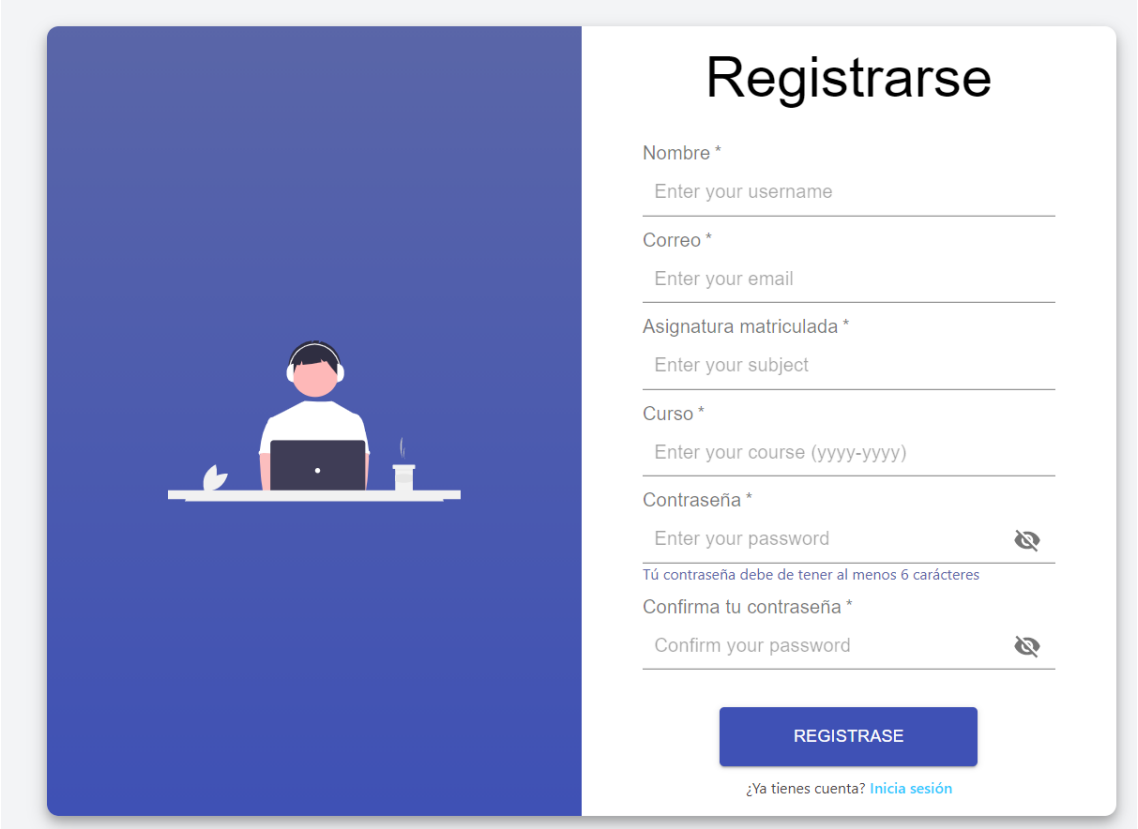

Figura 8.19: Pantalla login en pantallas grandes

En la figura 8.27 se puede observar el registro que se muestra al usuario para una pantalla que cuenta con menos de 950 píxeles de ancho.

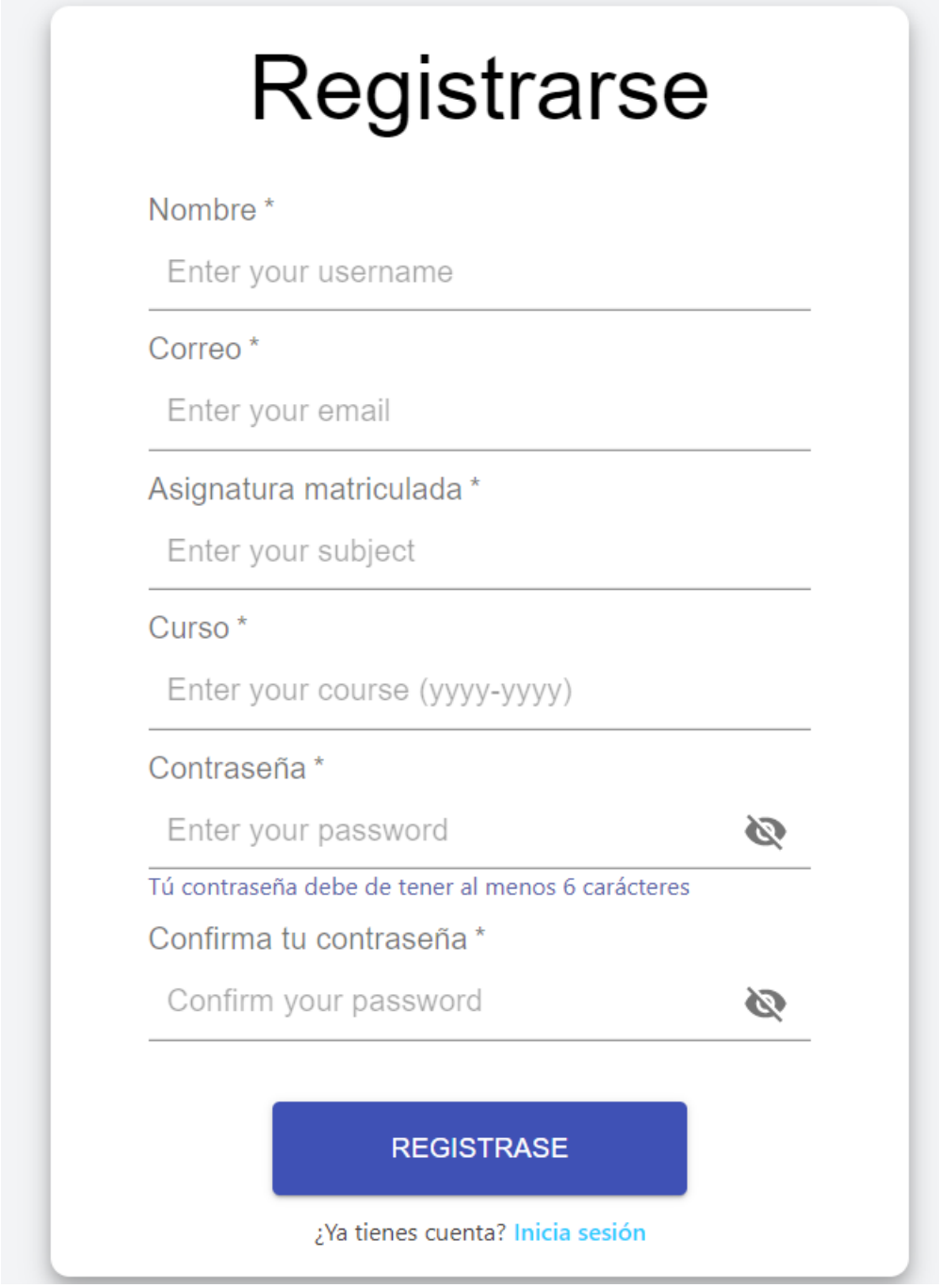

Figura 8.20: Pantalla login en pantallas <950px

Simulador

- Usuario registrado: En la figura 8.21 se puede observar el simulador para una pantalla que cuenta con más de 960 píxeles de ancho.

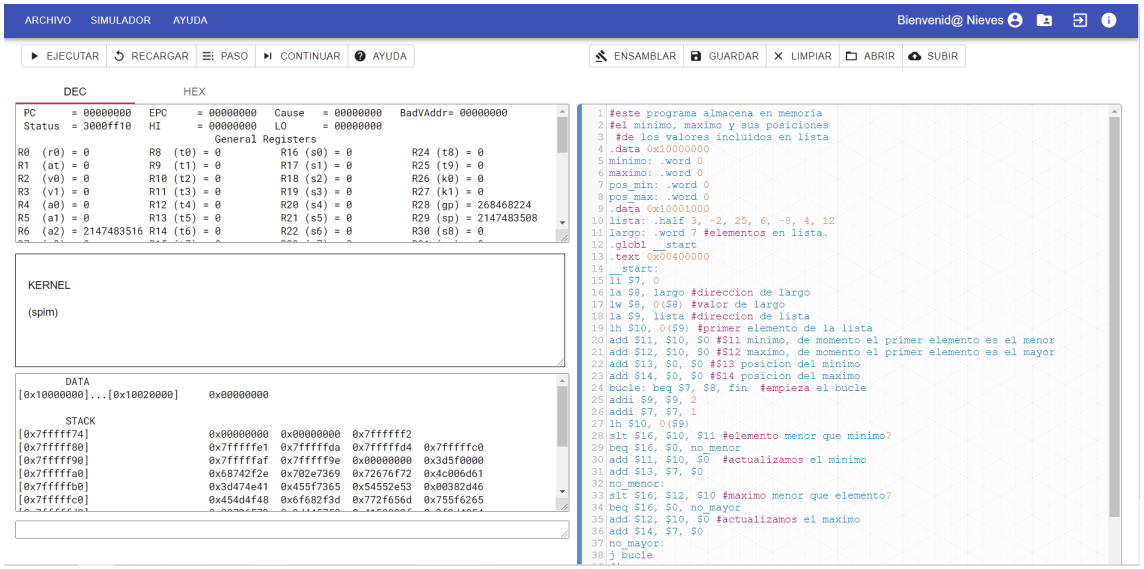

Figura 8.21: Página del simulador para el usuario registrado en pantallas >950px

En la figura 8.22 se puede observar el simulador para una pantalla que cuenta con menos de 960 píxeles de ancho.

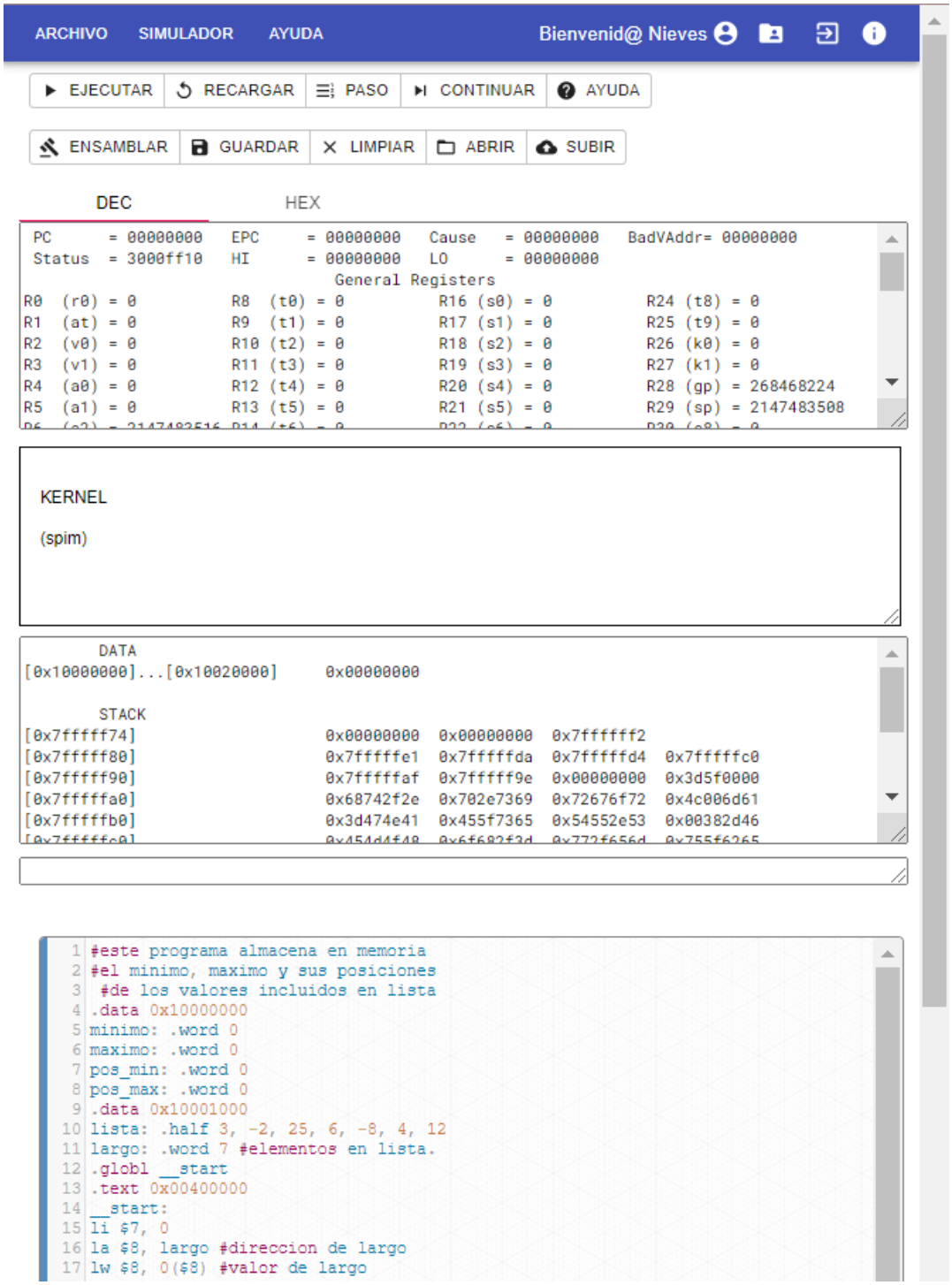

Figura 8.22: Página del simulador para el usuario registrado en pantallas <950px

- Usuario visitante: En la figura 8.23 se puede observar el simulador para una pantalla que cuenta con más de 950 píxeles de ancho.

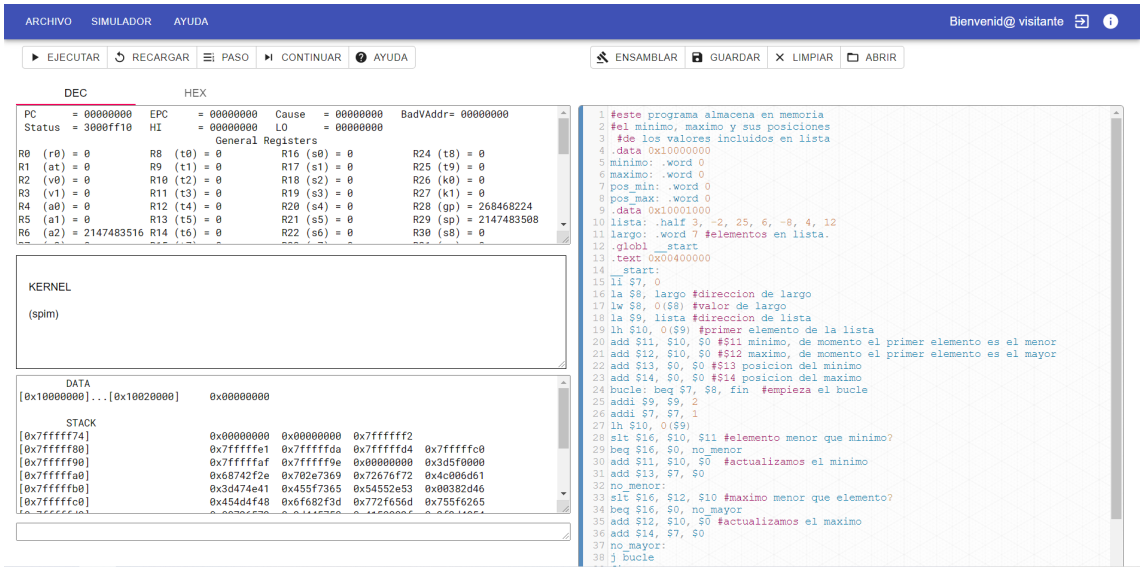

Figura 8.23: Página del simulador para el usuario visitante en pantallas >950px

En la figura 8.22 se puede observar el simulador para una pantalla que cuenta con menos de 950 píxeles de ancho.

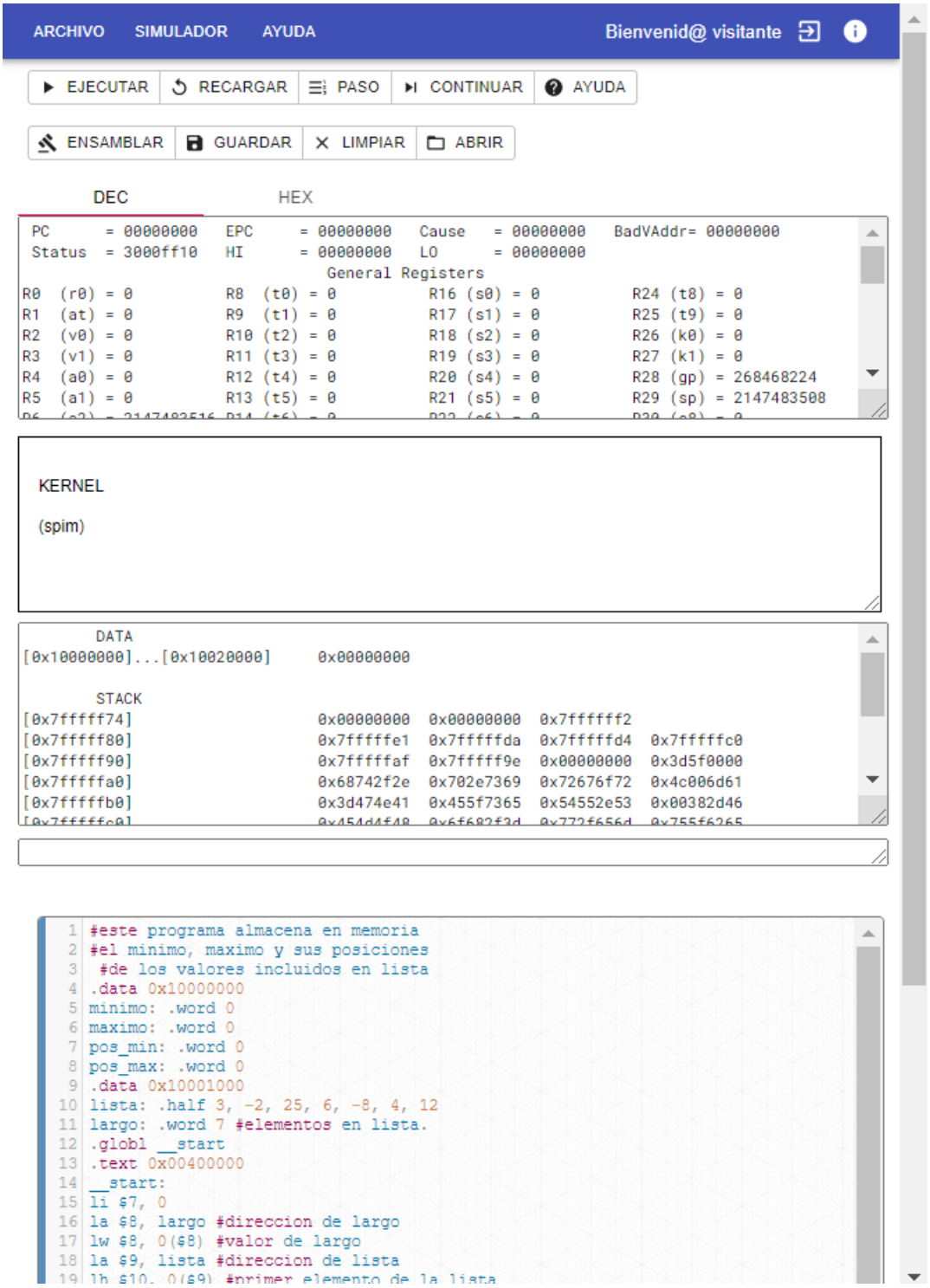

Figura 8.24: Página del simulador para el usuario visitante en pantallas <950px

8.1.3. Funcionalidades del simulador

Ejecutar

Si el usuario pulsa el botón de ejecutar, se le muestra un diálogo para que escriba que dirección de memoria quiere utilizar para el inicio de la ejecución del programa (figura 8.25).

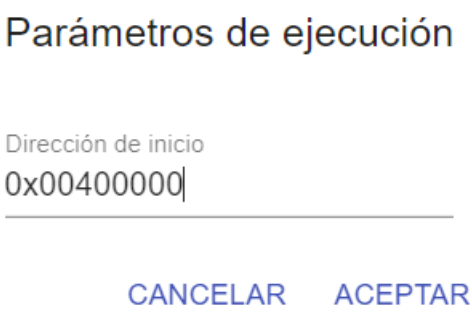

Figura 8.25: Prueba de validación: Diálogo para añadir la dirección de memoria inicial de la ejecución

De esta forma, si el usuario introduce una dirección de memoria que es inválida (que no tiene el formato "0x" seguido de 8 dígitos) se le muestra el campo de texto en rojo y se deshabilita la opción que le permite ejecutar el programa. En la figura 8.26 podemos ver representada esta situación.

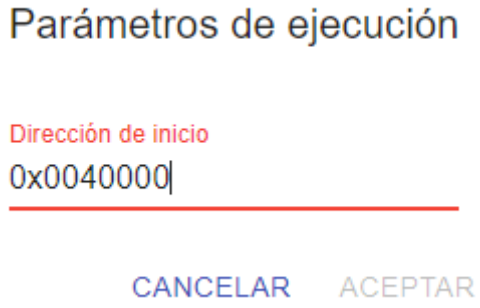

Figura 8.26: Prueba de validación: Dirección de inicio incorrecta

Si por el contrario introduce una dirección valida, el programa se ejecuta en el simulador, actualizándose el campo de registros, el de datos, el de texto y la consola. En la figura se puede observar esta situación, si observamos el PC ya no se encuentra en "00000000", si no que se ha desplazado a la ultima posición de memoria donde está la instrucción del programa.

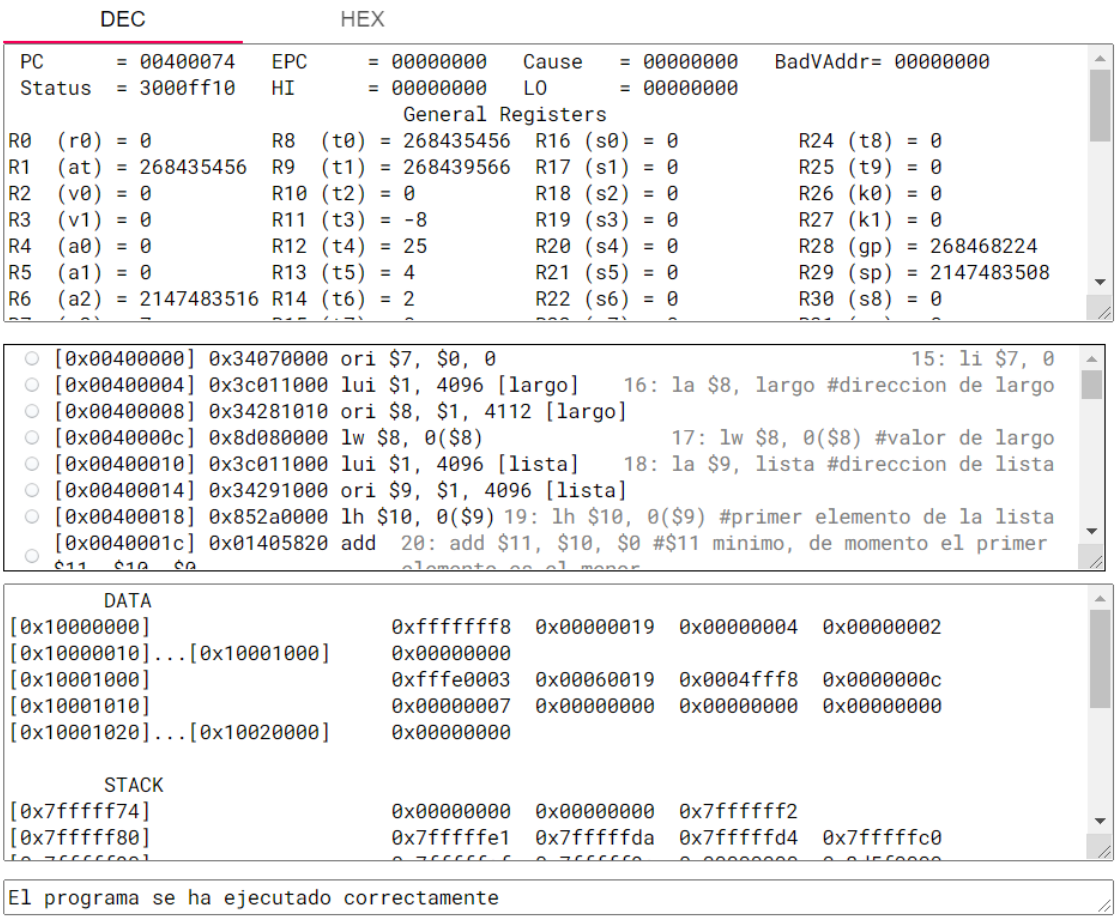

Figura 8.27: Prueba de validación: Ejecución correcta de un programa en el simulador

Si el programa que se ejecuta tarda más de tres segundos en acabar, el botón de ejecutar cambia al mostrado en la figura 8.28. Este botón permanece hasta que el programa se haya ejecutado.

Figura 8.28: Prueba de validación: Botón que muestra que un programa tarda en ejecutarse

Recargar

Si el usuario pulsa el botón de recargar, el programa se vuelve a ensamblar, actualizándose todos los campos del simulador.

Paso

Si el usuario pulsa el botón de paso, el programa avanza en su ejecución un paso, resaltando en un color amarillo fosforescente la linea de código en la que está la ejecución. En la figura 8.29 se puede observar esta situación.

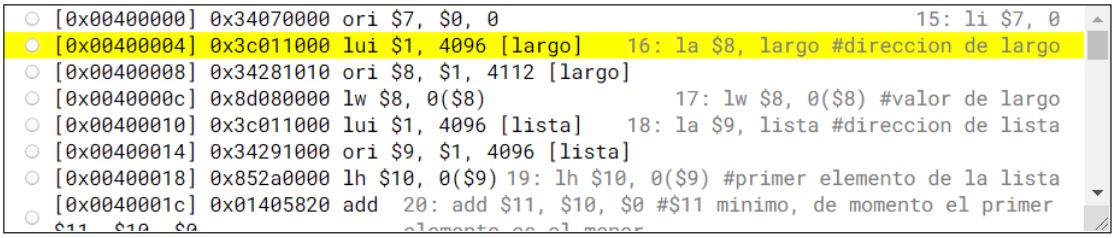

Figura 8.29: Prueba de validación: Segmento de código ejecutado por un paso

Si el usuario accede al menú del simulador y pulsa el botón de "Pasos", se le muestra un diálogo para que elija el número de pasos que desea, en este caso se han elegido 3 pasos como se puede observar en la figura 8.30.

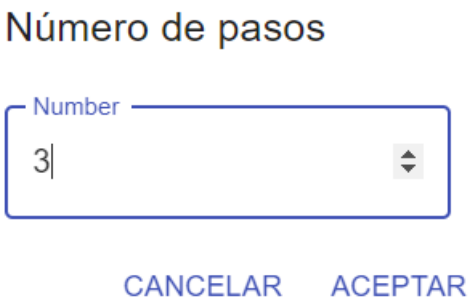

Figura 8.30: Prueba de validación: Pestaña de registros en decimal

Una vez que se acepte el diálogo, el resaltado, los campos de registros y datos y la ejecución del programa avanza tres pasos tal y como se le ha indicado. En la figura 8.31 se puede observar esta situación.

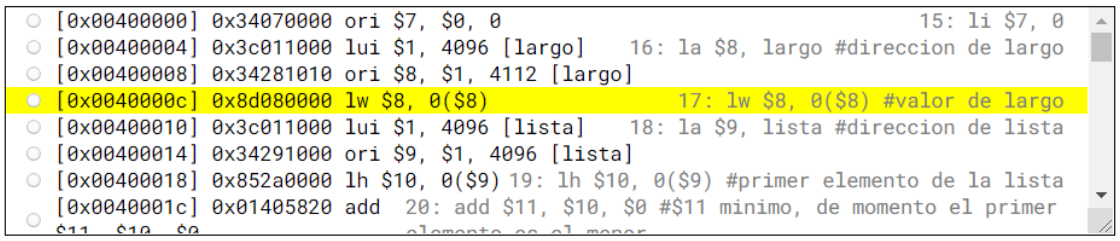

Figura 8.31: Prueba de validación: Segmento de código ejecutado por tres pasos

Continuar

Si el usuario quiere acabar con la ejecución, o pasar al siguiente punto de ruptura, debe pulsar al botón de continuar.

Registros

Si el usuario pulsa en pestaña "DEC", se muestran los campos correspondientes a los registros en decimal, tal y como se puede observar en la figura 8.32.

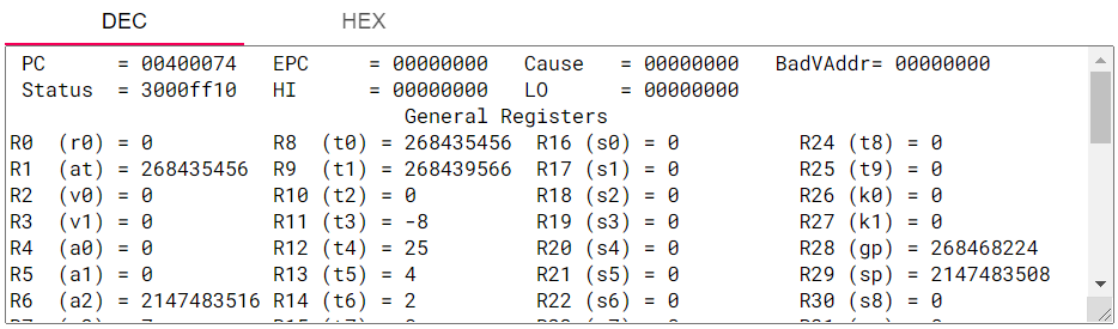

Figura 8.32: Prueba de validación: Pestaña de registros en decimal

Por otro lado, si el usuario pulsa en la pestaña "HEX", se muestran los campos correspondientes a los registros en hexadecimal, tal y como se puede observar en la figura 8.33.

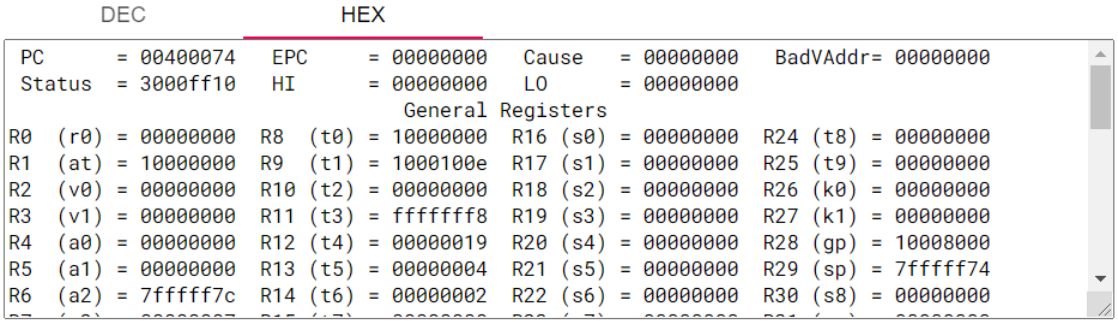

Figura 8.33: Prueba de validación: Pestaña de registros en hexadecimal

Puntos de ruptura

En este campo el usuario puede visualizar, una vez ensamblado el programa, las diferentes lineas de código. Si el alumno pulsa en los botones que hay al lado de cada linea puede crear un punto de ruptura completamente funcional, si lo vuelve a pulsar puede eliminarlo. En la figura 8.34 se puede observar un segmento de código de un programa, que cuenta con dos puntos de ruptura que el usuario ha establecido.

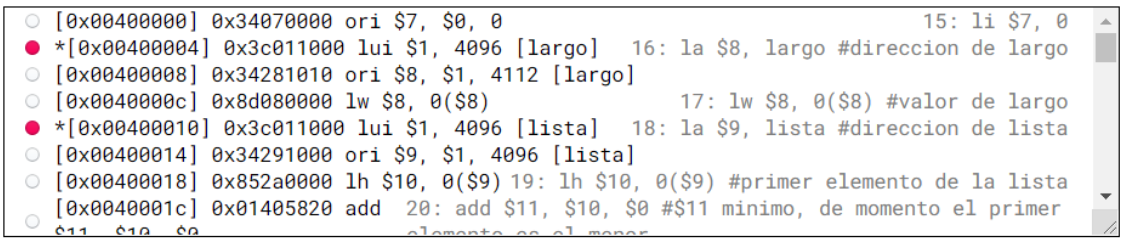

Figura 8.34: Prueba de validación: Segmento de código con *breakpoints*

Otra forma de establecer puntos de ruptura es a través del menú, pulsando la opción de "Puntos de ruptura", se muestra un diálogo para introducir la dirección de memoria donde, si se rellena con una dirección valida, se establece el *breakpoint*(figura 8.35). Si el usuario pulsa "Aceptar", el punto de ruptura se crea en la dirección introducida tal y como se puede observar en la figura 8.36.

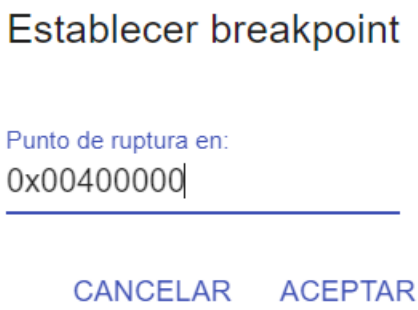

Figura 8.35: Prueba de validación:Introducir un archivo cuyo nombre ya existe en el servidor

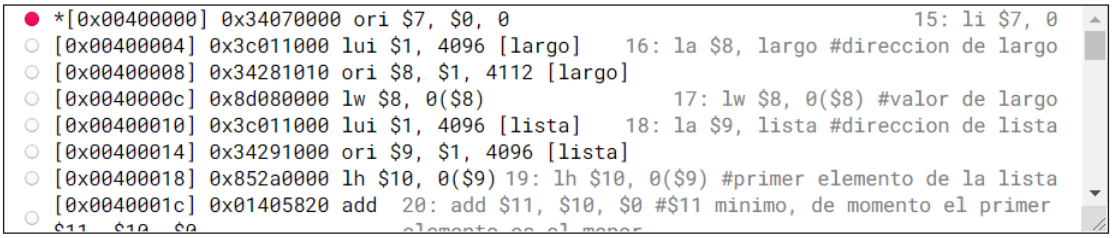

Figura 8.36: Prueba de validación: Segmento de código con *breakpoints*, establecido por medio del diálogo

8.1.4. Funcionalidades del editor de código

Ensamblar

Si el usuario ensambla en el simulador el programa correctamente, el campo de registros, el de datos, el de texto y la consola se actualizan. Si por el contrario no ha conseguido ensamblarse, muestra por medio de un diálogo un mensaje de error dando al usuario información sobre cual ha sido el problema.

Figura 8.37: Prueba de validación: Ensamblado correcto

Por el contrario, si el código contiene algún fallo, la aplicación muestra un diálogo con información sobre el error.

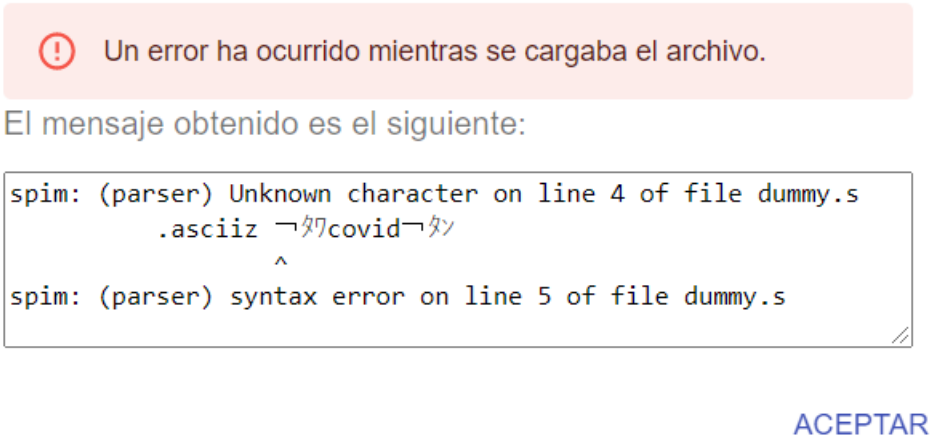

Figura 8.38: Prueba de validación: Diálogo de error para el ensamblaje

Limpiar editor de código

Si el usuario pulsa en el botón de "Limpiar", todo el código que está en el editor se elimina. En la figura 8.39 se puede observar esta situación.

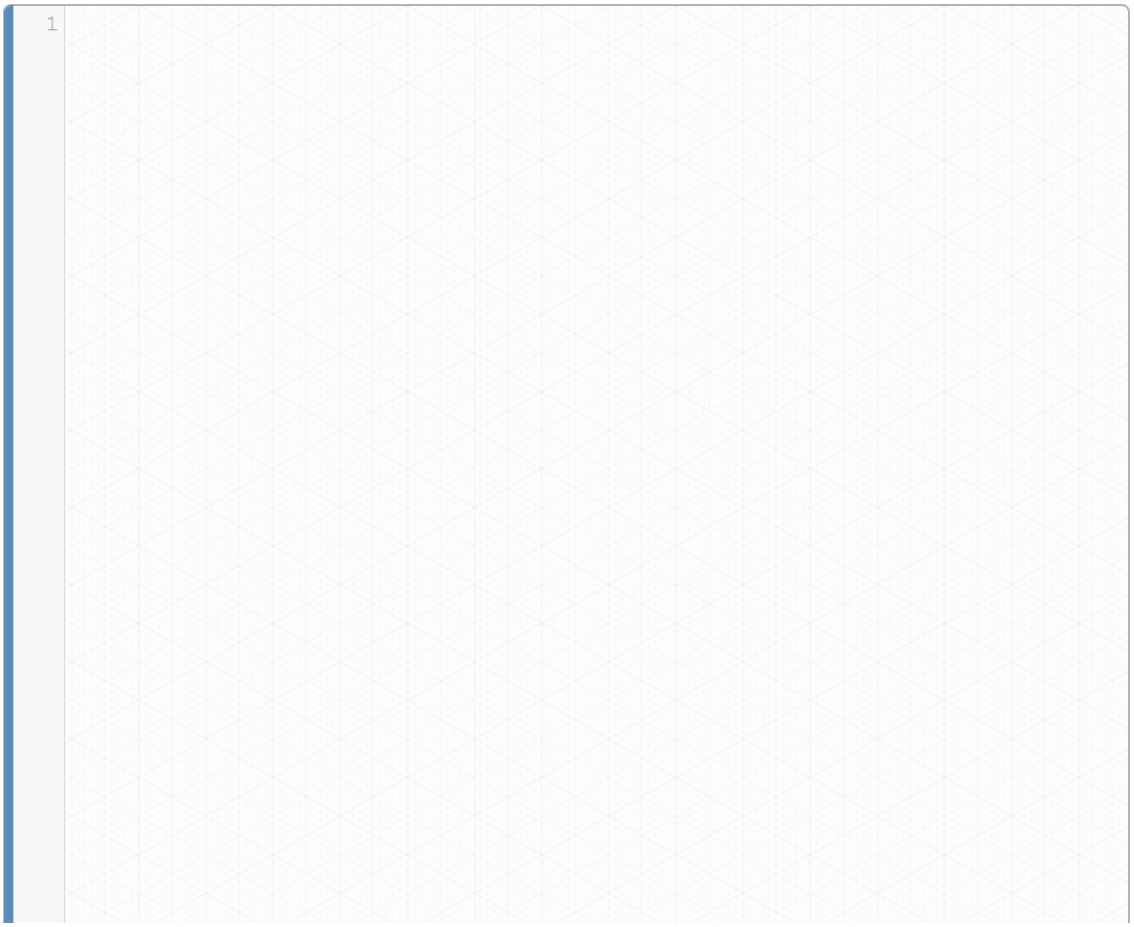

Figura 8.39: Prueba de validación: Limpiar editor de código

Guardar en disco

Si el usuario pulsa en el botón de "Guardar", se abre el sistema de archivos del usuario para poder guardar el programa que estaba almacenado en el editor en código en su sistema. En la figura 8.40 se puede observar la descarga en el navegador Chrome.

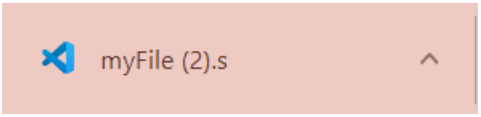

Figura 8.40: Prueba de validación: Guardar programa del editor en disco

Subir de disco

Si el usuario pulsa en el botón de "Subir", se abre el sistema de archivos con un filtro con la terminación que acepta el simulador (en este caso .s). El usuario puede elegir de su disco que programa que desea, tal y como se puede observar en la figura 8.41.

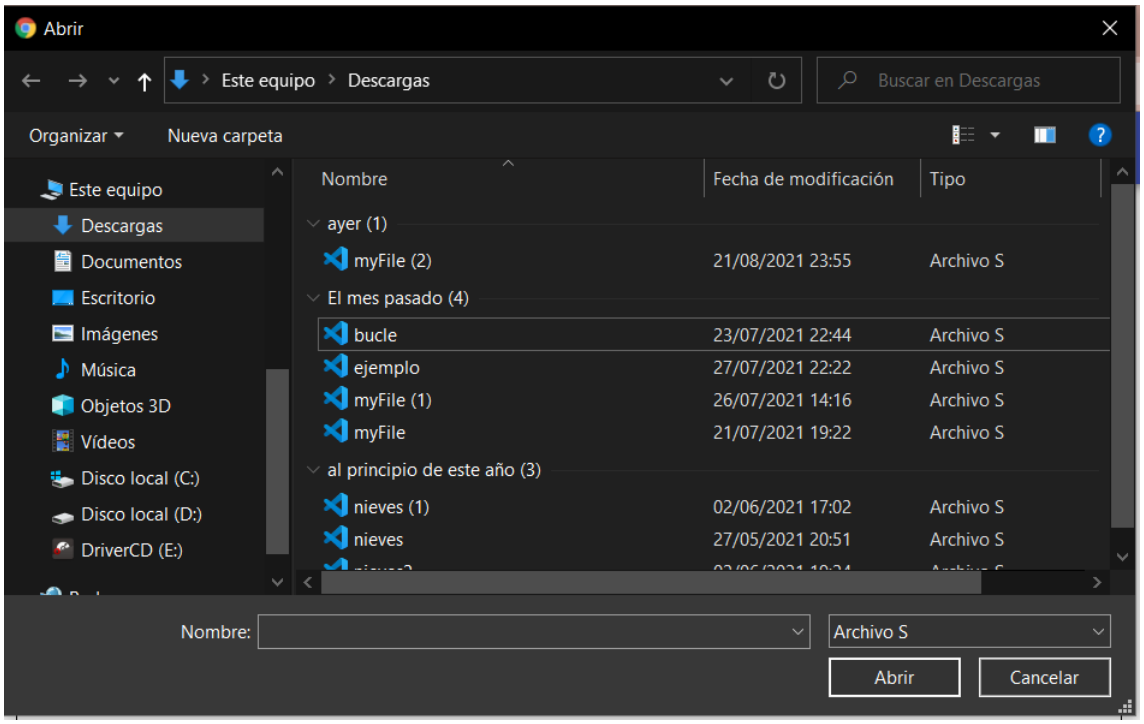

Figura 8.41: Prueba de validación: Sistema de archivos para subir un programa

Cuando el usuario haya seleccionado un archivo, este se carga en el editor de código tal y como se puede observar en la figura 8.42.

```
1  .globl __start
2
3
4
5  .data
6  .byte 1, 3, -1, 0xff
7  valor: .word 0x1ffffff
8  .asciiz "AABBA"
9
10
11
12 .text
13 __start:
14
15
16
17 addi $5, $0, 0
18 la $2, valor
19 lw $2, 0($2)
20 bucle: addi $5, $5, 1
21 beq $5, $2, fin
22 j bucle
23 fin:
24 .end
```

Figura 8.42: Prueba de validación: Editor de código con el programa del servidor cargado

8.1.5. Gestión de programas

Subir un programa del editor al servidor

Si el usuario pulsa el botón de "Subir", se le muestra un diálogo para que introduzca el nombre que quiere asignar al programa que se encuentra en el editor de código. Si introduce el mismo nombre para guardar en el servidor el programa almacenado en el editor de código, el sistema le muestra un mensaje de error especificándole este hecho. En la figura 8.44 podemos ver representada esta situación.

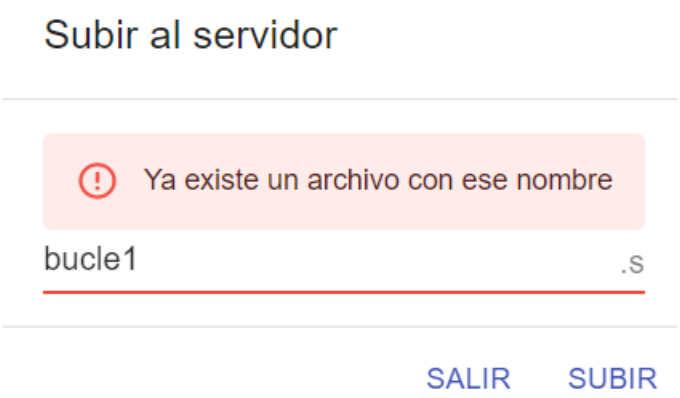

Figura 8.43: Prueba de validación: Introducir el mismo nombre de un programa que ya existe en el servidor

Si por el contrario, no hay ningún archivo propio del usuario que contenga ese nombre, entonces se sube el programa al servidor y se muestra un mensaje como se puede observar en la figura .

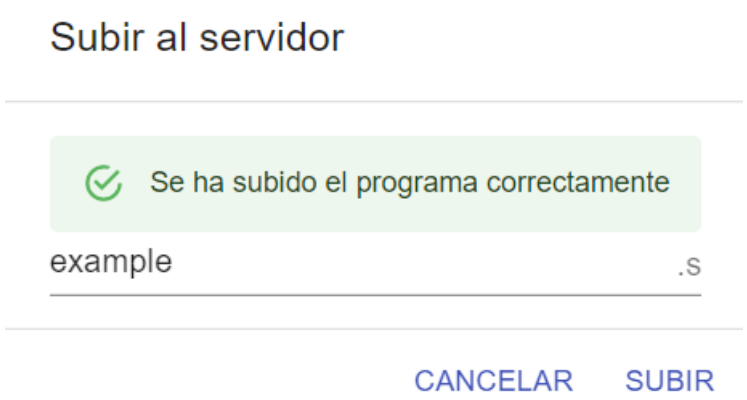

Figura 8.44: Prueba de validación: Subir un fichero que no existe en el servidor

Subir un archivo desde disco al servidor

Si el usuario pulsa en el botón "Seleccionar archivo" se le muestra su sistema de archivos (del mismo modo que ocurría en la figura 8.41). De este modo, y una vez elegido el programa (figura 8.45, el usuario pulsa el botón de "Subir" e inmediatamente se crea una nueva fila en la lista de ficheros del alumno con el mismo nombre del archivo subido.

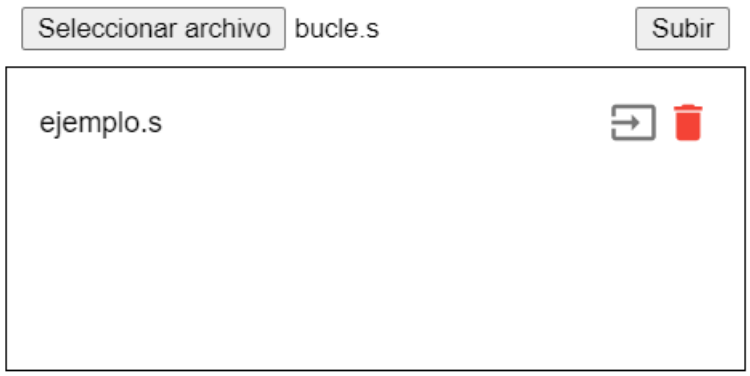

Figura 8.45: Prueba de validación: Añadir un fichero en el *input* de la ventana de gestión de ficheros

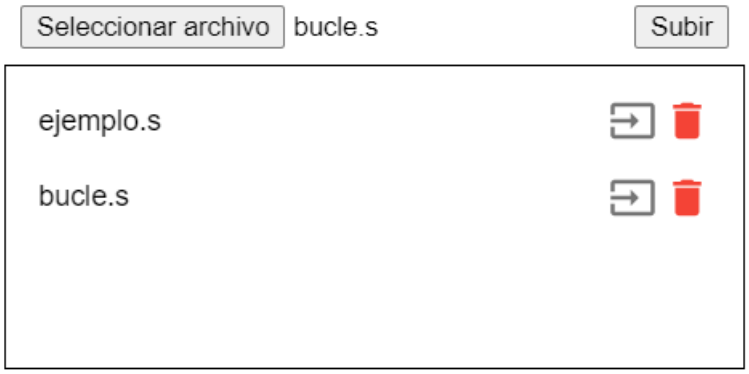

Figura 8.46: Prueba de validación: Programa de disco subido al servidor

Si el usuario intenta subir un archivo que supera los 1MB de tamaño, se muestra al usuario un mensaje de error (figura 8.47), no subiéndose al servidor el archivo seleccionado.

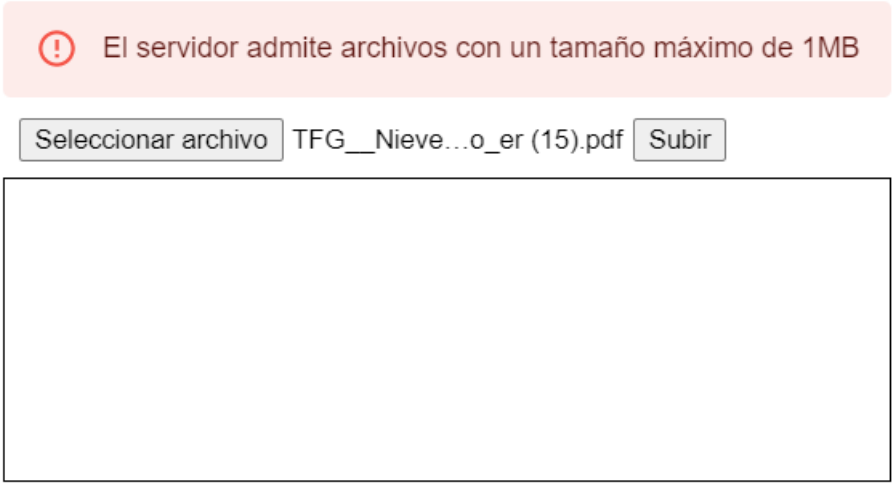

Figura 8.47: Prueba de validación: Añadir un archivo de más de 1MB

Por otro lado, si el archivo que el usuario ha elegido se encuentra ya en el servidor, el sistema lo notifica con un mensaje de error tal y como puede observarse en la figura 8.48.

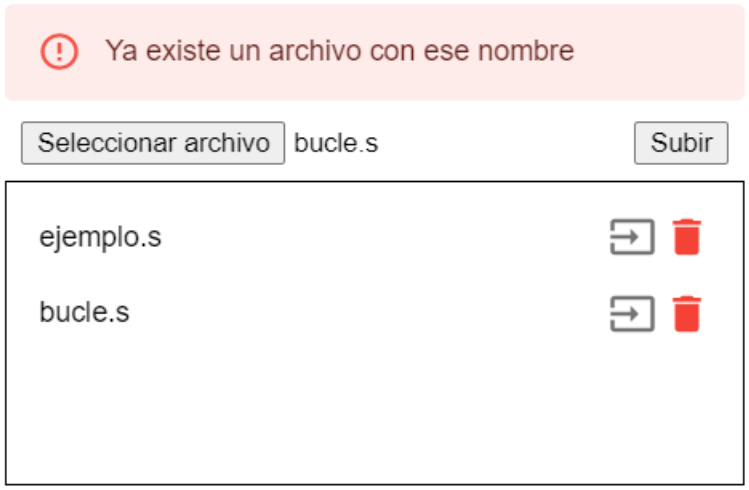

Figura 8.48: Prueba de validación: Introducir un archivo cuyo nombre ya existe en el servidor

Borrar un programa del servidor

Si el usuario desea borrar el fichero del servidor, tiene que pulsar el icono representado por una papelera. Tal y como se muestra en la figura 8.49, se ha procedido a borrar el archivo con nombre "ejemplo.s".

Figura 8.49: Prueba de validación: Borrar un programa del servidor

Cargar un programa del servidor al editor de código

El usuario puede cargar un fichero del servidor en el editor de código, para ello, debe pulsar en el botón que podemos ver representado en la figura 8.50. Una vez pulsado, en editor aparece el programa seleccionado anteriormente en el modal.

Figura 8.50: Prueba de validación: Botón para cargar un programa en el editor de código

8.1.6. Información de la aplicación

Si el usuario pulsa el botón de la interfaz "Ayuda" o el botón en el menú con el mismo nombre, se muestra en la pantalla un dialogo deslizable (figura 8.51) que cuenta con los diferentes iconos utilizados en la aplicación y un resumen de para que sirve cada uno.

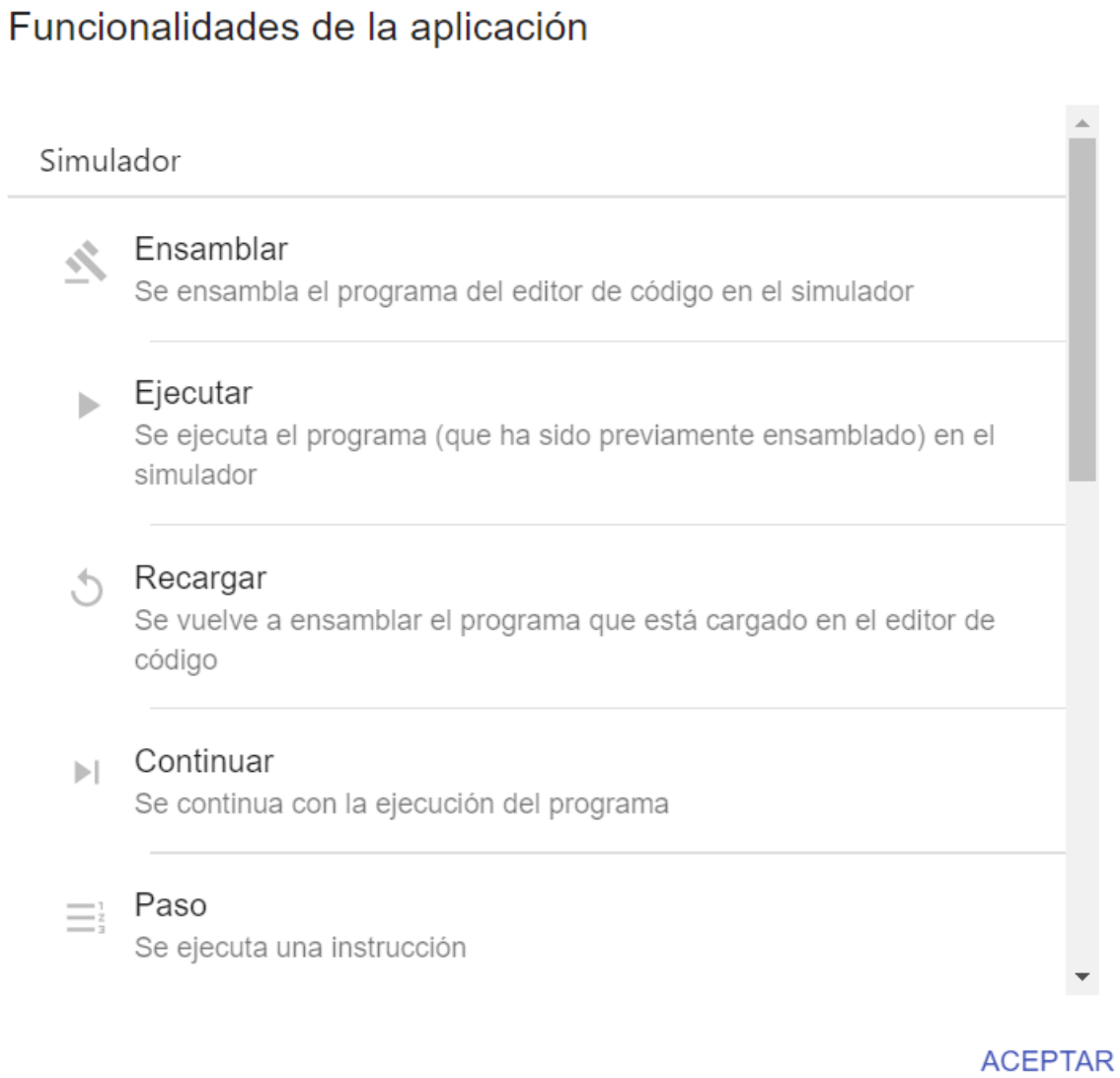

Figura 8.51: Prueba de validación: Diálogo de ayuda

Por otro lado, si el usuario pulsa el botón de la interfaz "Ayuda" o el botón en el menú con el mismo nombre, se muestra en la pantalla un dialogo (figura 8.52) que contiene la información sobre el TFG.

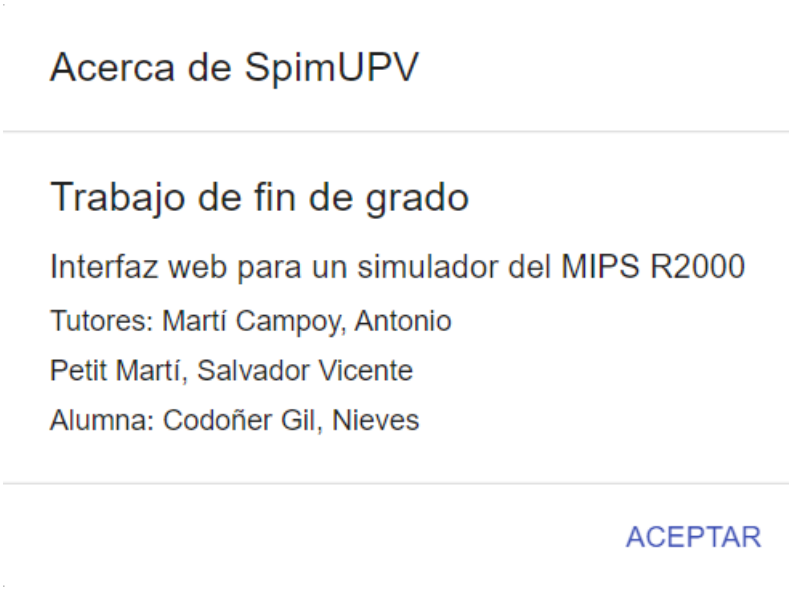

Figura 8.52: Prueba de validación: Diálogo de información sobre el TFG

8.1.7. Información del usuario y borrado de cuenta

Si el usuario quiere revisar su información de cuenta, tiene que acceder pulsando al icono de cuenta del usuario. De esta forma, se abre un dialogo en la que aparece su información (figura 8.53). Por otro lado, también existe la posibilidad que quiera borrar su cuenta de la aplicación. Si el usuario pulsa en "Borrar cuenta" tanto su cuenta como los programas que haya subido al servidor son eliminados, redireccionandole a la página de login y mostrando un mensaje sobre el borrado de cuenta (figura 8.54)

Figura 8.53: Prueba de validación: Diálogo de información sobre el usuario *logueado*

Figura 8.54: Prueba de validación: Mensaje de borrado de cuenta satisfactorio.

CAPÍTULO 9
Implantación

En el presente apartado se muestra el proceso de despliegue de la aplicación en servidores externos a la UPV.

Se ha optado por desplegar el servidor en la plataforma Heroku y el cliente en Netfily (que fueron comentadas en el apartado 6.3.4 de tecnologías). Estas dos plataformas cuentan con una versión gratuita que permite mantener alojada la aplicación y comprobar que el despliegue se puede realizar con éxito. El motivo que subyace de esta decisión recae en que desde un principio se ha realizado el desarrollo del cliente y del servidor por separado, y además, esta opción permite utilizar dos tecnologías que tienen procesos de despliegue diferentes.

9.1 Despliegue del backend

En este apartado se va a proceder a desplegar el servidor de la aplicación en la plataforma Heroku. En primer lugar, se accede a la página principal de Heroku `https://dashboard.heroku.com/` y una vez en ella se crea una cuenta gratuita pulsando el botón que se muestra en la figura 9.1.

Figura 9.1: Botón para la creación de una cuenta gratuita en Heroku

Una vez creada la cuenta, se procede a la creación de la aplicación pulsando el botón mostrado en la figura 9.2.

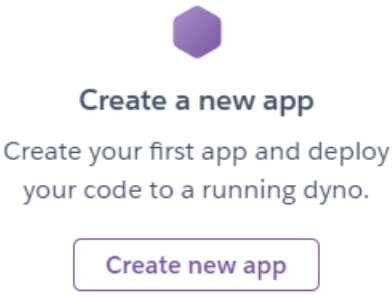

Figura 9.2: Botón para la creación de una nueva aplicación en Heroku

Este botón muestra una pantalla de configuración, dónde se tiene que introducir el nombre para la aplicación (que será el que se utilizará como subdominio de Heroku para acceder al servidor) y la región donde se quiera alojar la aplicación (figura 9.3).

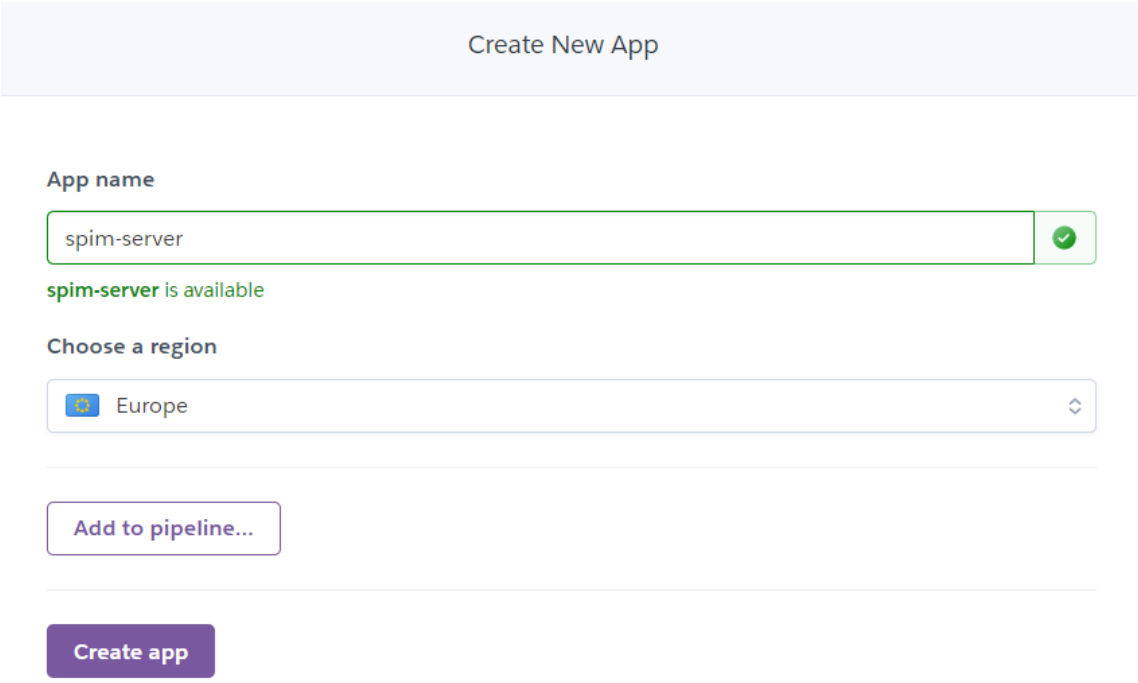

Figura 9.3: Página para introducir la información sobre los datos de la aplicación

Inmediatamente después de la creación de la aplicación, se comienza a instalar Heroku, se abre el proyecto (en este caso en el editor de código Visual Studio code) donde está alojado el servidor y se introduce el comando "heroku login" para iniciar sesión desde la terminal(incluyendo las credenciales en la ventana que mostrará una vez enviado).

Como el servidor ya está inicializado en un repositorio de GitHub (tal y como se mostró en el desarrollo del *backend* 7.3), se procederá a introducir el comando " heroku git:remote -a spim-server" para asignar el código en el repositorio remoto de Heroku y seguidamente se utiliza el comando "git push heroku master" para desplegar el servidor en la plataforma. Una vez realizados los pasos, se puede acceder a la aplicación por medio del botón "Open app" mostrado en la figura 9.4 o a través de URL anteriormente proporcionada.

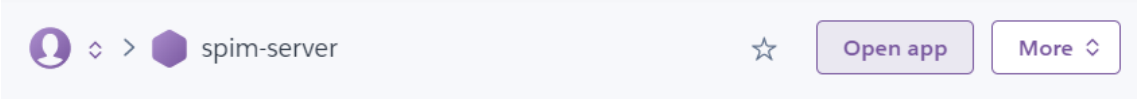

Figura 9.4: Botón para abrir la aplicación subida

Para comprobar que el despliegue del servidor funciona correctamente se ha utilizado Postman. De esta forma se realiza una petición HTTP (en este caso una petición POST con un objeto JSON que contiene los diferentes campos para registrar a un usuario en la aplicación). Como se puede observar en la figura 9.5, el servidor contesta que el usuario se ha guardado satisfactoriamente en la base de datos.

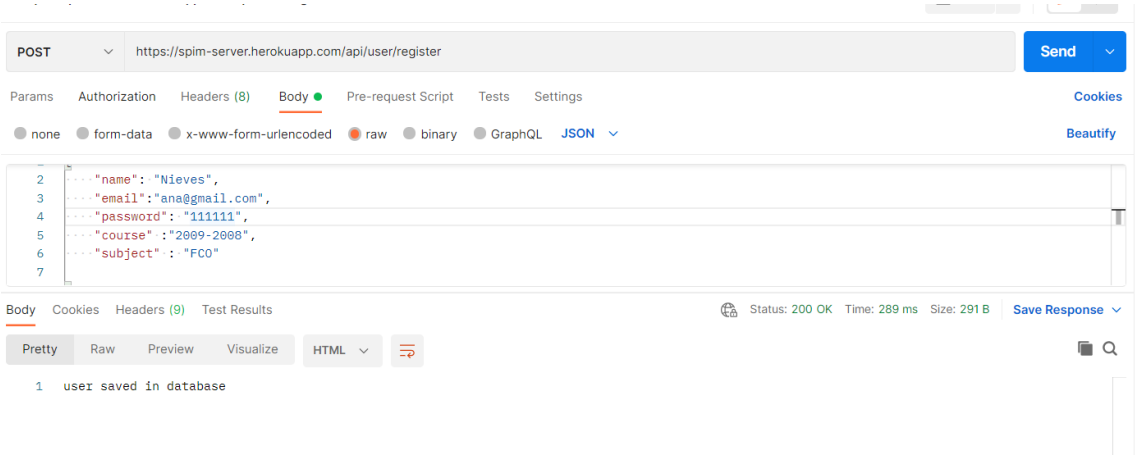

Figura 9.5: Prueba de registro en el servidor subido a Heroku

Además, si se accede a la base de datos de MongoDB se puede observar que el usuario que se ha introducido en la petición de Postman, está guardado en la colección usuario de la base de datos (figura 9.6)

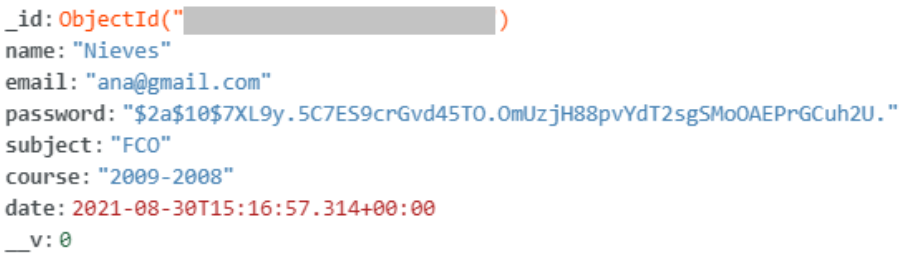

Figura 9.6: Prueba de registro en MongoDB

9.2 Despliegue del frontend

En este apartado se va a proceder a desplegar el cliente de la aplicación en la plataforma Netfily. Antes de empezar con el proceso, habrá que cambiar la URL de todas las peticiones del cliente a la nueva API del servidor (este proceso es facilitado gracias al uso de una variable global donde se almacena la URL del servidor).

En primer lugar, se accede a la página principal de Netfily `https://www.netlify.com/` y una vez en ella se crea una cuenta gratuita pulsando el botón que se muestra en la figura 9.7.

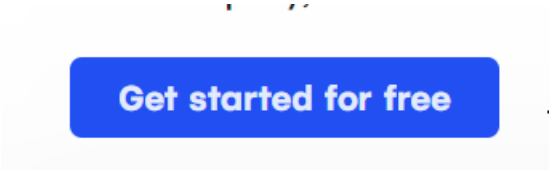

Figura 9.7: Botón para la creación de una cuenta gratuita en Netlify

En los siguientes puntos se expondrá el proceso de creación de un nuevo sitio web:

1. Se conecta la plataforma con la cuenta de GitHub donde está alojado el proyecto (figura 9.8).

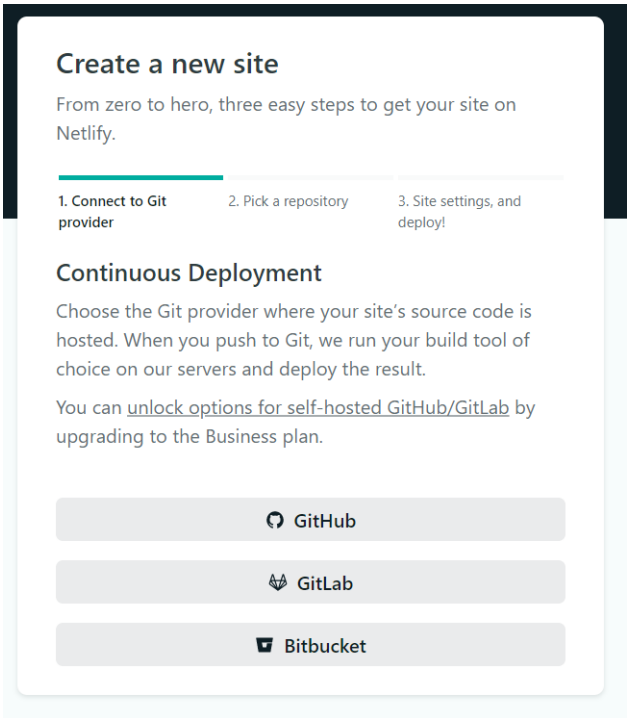

Figura 9.8: Página para la creación de un sitio web: Conectar con GitHub

2. Se elige el repositorio dónde está alojado el proyecto que contiene el cliente de la aplicación (figura 9.9).

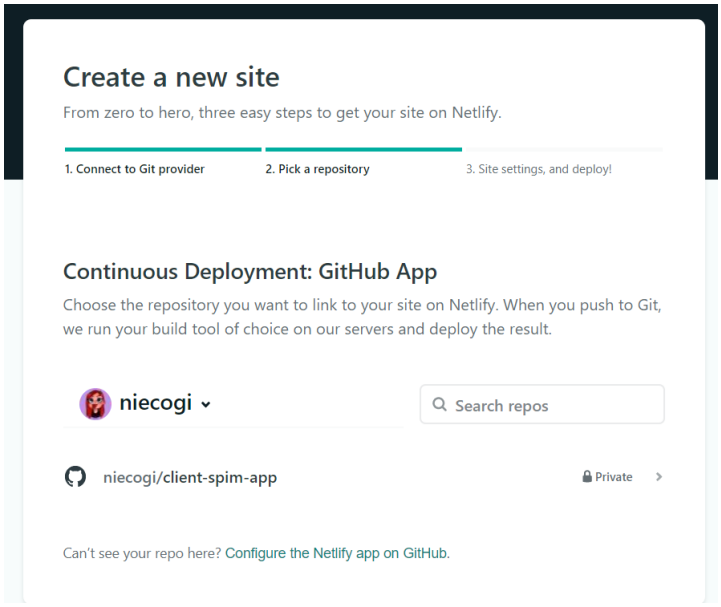

Figura 9.9: Página para la creación de un sitio web: Elegir un repositorio

3. Se detallan los ajustes para la "build"de la aplicación (figura 9.10).

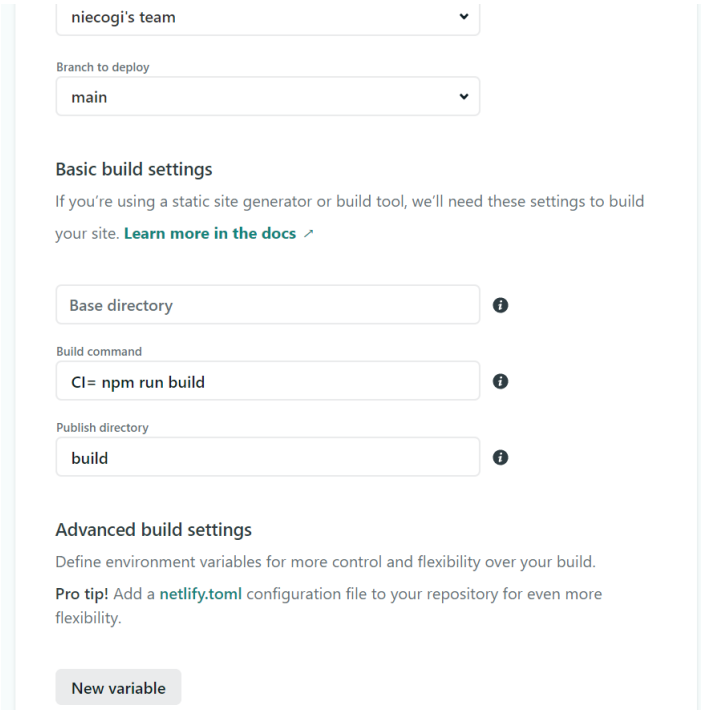

Figura 9.10: Página para la creación de un sitio web: Ajustes del sitio web

4. La aplicación se despliega en un dominio aleatorio proporcionado por Netfily (figura 9.11).

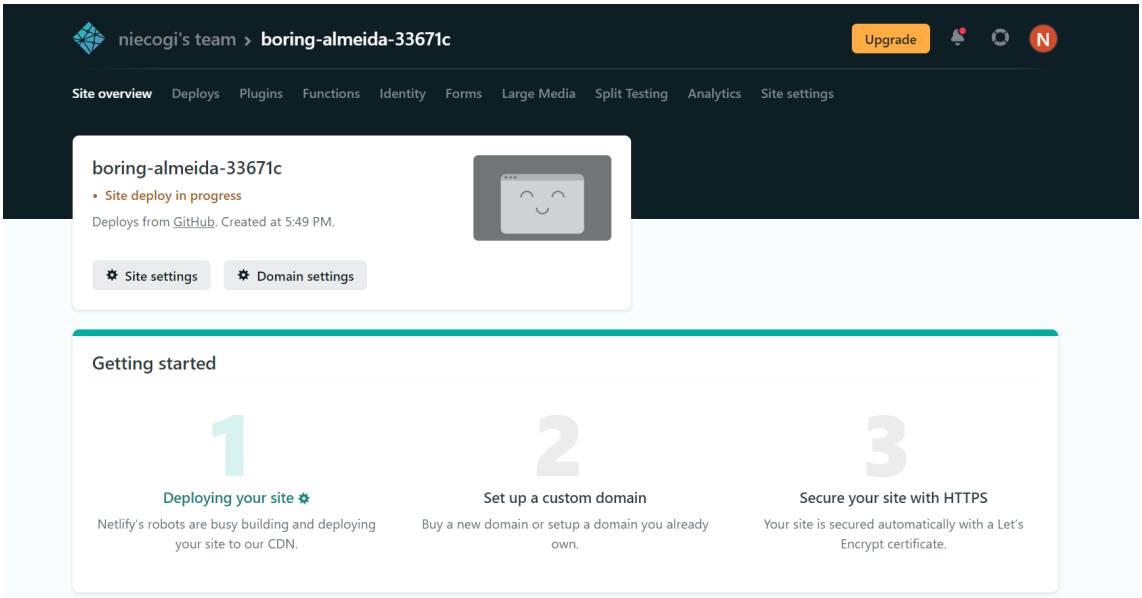

Figura 9.11: Desplegando en Netfily

Por otro lado, Netfily permite la creación de un nuevo subdominio. De esta forma, si se accede al apartado “Domain managament", se puede cambiar el nombre establecido por la plataforma. En la figura se muestra el campo para el cambio de nombre (figura 9.12).

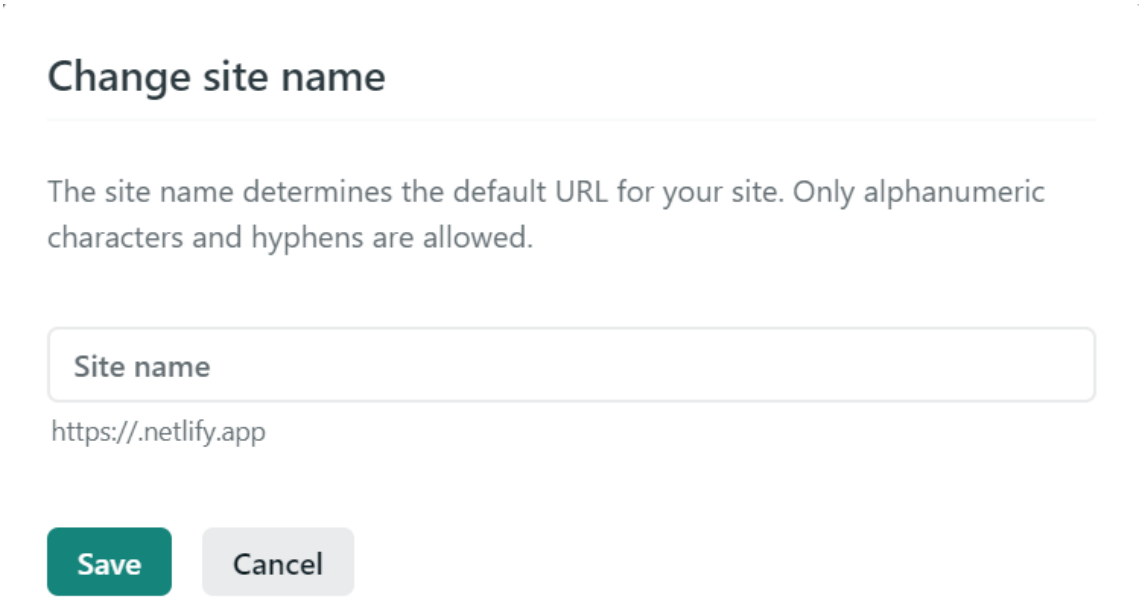

Figura 9.12: Página para el cambio URL de la aplicación en Netfily

Una vez modificado, se procede a entrar a la URL proporcionada (`https://spim.netlify.app/`), pudiéndose observar como el cliente está desplegado y se puede comunicar perfectamente con el servidor subido a Heroku (figura 9.13).

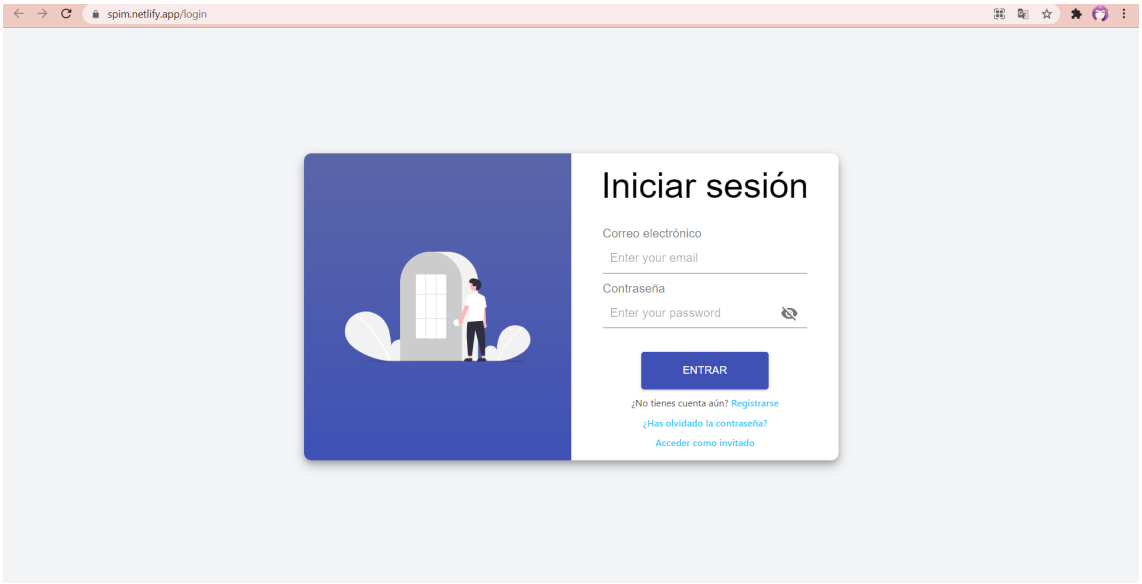

Figura 9.13: Página web desplegada en Netfily

CAPÍTULO 10
Presupuesto

En el presente apartado se procederá a exponer el presupuesto estimado para el proyecto.

En primer lugar, el salario de un graduado en ingeniería informática oscila entre los 12.000 y 30.000 euros anuales [41], contando con un sueldo por hora de unos 10 euros aproximadamente. Por otro lado, el sueldo de un profesor de la Universitat Politècnica de València ronda entre los 31.000 y 36.000 euros anuales [42], contando con un sueldo por hora de unos 20 euros aproximadamente.

Con referencia al coste derivado del equipo utilizado (en este caso un portátil) se tienen que tener en cuenta dos factores: precio de la luz consumida y precio de adquisición del equipo. El coste de adquisición ronda los 1000 euros, suponiendo para el proyecto unos 0,08 euros por cada hora dedicada. Por otro lado, el coste de electricidad que consume el equipo de trabajo es aproximadamente de 0,004875 euros la hora. Es decir, la suma entre el consumo eléctrico y el de adquisición rondará los 31 euros.

En cuanto el coste de la base de datos en MongoDB, se cuenta con un plan gratuito, pero si se quisiera llevar a producción se necesitaría aumentar el plan de pago. De esta forma, se podría optar por la opción "Serverless" [43] que cuenta con un coste de 0,30 euros por cada millón de lecturas, suponiendo de esta manera un coste total aproximado por mes de 0,60 euros.

Con respecto al coste del despliegue, se cuenta con un plan gratuito en cada plataforma, pero si se quisiera llevar a producción se necesaria aumentar el plan de pago. De esta forma, se cuenta con dos herramientas diferentes que cuentan con los planes mensuales:

- Heroku : Plan "Production" [33] con un coste mensual de 25 euros.

- Netlify : Plan "Pro" [35] con un coste mensual de 19 euros.

En la tabla 10.1 se puede observar el coste del desarrollo y el coste de explotación del proyecto.

Recursos	Tiempo (h)	Coste (€)
Alumno	360	10
Profesor	30	20
Ordenador	360	0,08
Consumo energético	360	0,004875
	Coste del desarrollo	4230,55 euros
Base de datos	Coste mensual	0,60
Despliegue Heroku	Coste mensual	25
Despliegue Netlify	Coste mensual	19
	Coste de explotación (mensual)	44,6 euros

Tabla 10.1: Estimación del presupuesto

CAPÍTULO 11
Conclusiones

Todos los objetivos que se plantearon en el apartado 1.2 han sido cumplidos satisfactoriamente. De este modo, se ha creado una aplicación web que proporciona a los usuarios una interfaz intuitiva y sencilla para la utilización del simulador, añadiendo funcionalidades nuevas tales como el registro e inicio de sesión, el editor de código o el sistema de archivos.

En lineas generales, la planificación que se planteó en la sección 1.4 ha sido acertada. No obstante, ha habido alguna actividad que consumió más tiempo del esperado, particularmente el desarrollo y las pruebas realizadas. Se invirtió más tiempo del estimado en el desarrollo de la propuesta, ya que en un principio no se contaba con los conocimientos necesarios para realizar el proyecto de una forma ágil. Una vez adquiridos, el proceso fue asemejándose a la estimación prevista. En cuanto a las pruebas implementadas, se fueron encontrando fallos o situaciones que no se esperaba que ocurriesen, por lo que se necesitó algunos días más para conseguir que el funcionamiento fuese correcto. En contraposición , la especificación de requisitos consumió menos tiempo del esperado, ya que se contaba con unos requisitos claros desde un primer momento y no cambiaron en exceso a partir de esa ocasión.

11.1 Valoración

Para concluir con este capitulo, me gustaría puntualizar una serie de aspectos sobre el proceso de desarrollo del proyecto.

En primer lugar, debo de decir que nunca había hecho ningún proyecto web. Aunque, si bien es cierto que contaba con algún conocimiento muy básico de JS, HTML y CSS, jamás me había lanzado a comenzar una aplicación, y menos con tal envergadura como esta. Gracias a su elaboración, he conseguido una cantidad ingente de conocimiento. He investigado sobre muchos temas diferentes: transcompiladores, frameworks, librerías, frontend, como crear código limpio, diseño de patrones, estructuras, backend... Todo ello me ha servido para crecer como futura ingeniera en informática, y darme cuenta que, aunque al principio me veía abrumada con los requisitos propuestos, he conseguido cumplir con todos ellos. Adicionalmente, he podido aplicar muchos de los conocimientos aprendidos en la carrera (en el apartado 13 se detallan aquellas asignaturas más influyentes en el proyecto)

Por otro lado, la realización del proyecto web me ha descubierto otra rama de la informática que siempre había despertado interés en mi, pero que por unos motivos u otros, no había podido probar hasta ahora. La parte del desarrollo que más me ha gustado sin duda ha sido el frontend. La libreria React me ha abierto un mundo de posibilidades, y,

aunque al principio fue bastante costoso entender como funcionaba, acabó siendo mi parte favorita del proceso de creación de software. Sin duda, seguiré investigando sobre esta librería, aprendiendo más sobre la misma e introduciéndola en mis proyectos personales.

Desde un punto de vista profesional, este trabajo me será de gran ayuda, ya que se han utilizado numerosas tecnologías que se emplean actualmente en el mundo laboral. Además, al haberme visto inmersa en un proyecto real, he tenido que saber gestionar el tiempo y conseguir cumplir con las planificaciones que se iban pidiendo, con el fin de ayudar también a controlar los niveles de estrés que estas podían producir.

En conclusión, creo que este proyecto me ha ayudado tanto personalmente como profesionalmente, creando de esta forma una mejor versión de mi persona y ayudándome tanto a aplicar los estudios cursados en el desarrollo de aplicaciones como de ampliar mis conocimientos en otras tecnologías. Todo ello, con el propósito de utilizarlos como una herramienta para en un futuro poder enfrentarme de una forma eficaz al mundo laboral.

CAPÍTULO 12
Trabajos futuros

En el presente capitulo se detallarán una serie de ideas que han ido surgiendo en el desarrollo y que resultan interesantes como posibles trabajos futuros. A continuación, se listan una serie de funcionalidades que se podrían añadir al proyecto:

- Implementación de las opciones del simulador en la aplicación. En el presente enlace se observan algunas de las opciones que pueden ser añadidas en el simulador http://pages.cs.wisc.edu/~larus/SPIM_command-line.pdf
- Mejoras en la gestión de archivos: añadiendo información como una descripción que pueda añadir el usuario, la posibilidad de editar esta información y filtros para organizar los programas según su nombre o fecha de subida.
- Mejoras en el sistema de registro: correo de verificación de cuenta.
- Implementación de un sistema de carpetas, para que los alumnos puedan organizar mejor sus proyectos.
- Verificación para garantizar que los usuarios registrados son alumnos matriculados en las asignaturas que utilizan la aplicación.
- Verificación para garantizar que los profesores que tengan acceso a los programas que suban como tareas los alumnos, sean tutores de los mismos e impartan dichas asignaturas.
- Implementación de un sistema que ofrezca a los profesores el acceso a las tareas o programas que suban sus alumnos con el objetivo de corregir posibles ejercicios propuestos.
- Añadir una nueva colección a la base de datos que contenga los datos de los diferentes profesores que utilicen en sus asignaturas la aplicación. Esta colección incluirá los campos: nombre, correo electrónico, contraseña, asignaturas impartidas e identificación del profesor.

CAPÍTULO 13

Relación del trabajo desarrollado con los estudios cursados

Para el desarrollo del proyecto se han utilizado conocimientos proporcionados por diferentes asignaturas del Grado en Ingeniería Informática. Cabe destacar, que aunque no se hace referencia a todas ellas, de una forma u otra todas las asignaturas han ayudado a consolidar una base de la cual partir. De este modo, las asignaturas más relacionadas con el TFG son las siguientes:

- **Tecnología de sistemas de información en la red (TSR):** Asignatura donde se da una introducción al lenguaje de programación JavaScript y el entorno de ejecución Node.js.
- **Fundamentos de computadores (FCO), Estructura de computadores (ETC), Arquitectura e ingeniería de computadores (AIC) y Arquitecturas avanzadas (AAV):** Asignaturas que dan a conocer conocimientos sobre el lenguaje ensamblador, la arquitectura R2000 y simuladores. Estas asignaturas han sido un punto de partida para entender que requisitos son necesarios en la aplicación. Además, la asignatura FCO será en la cual se hará uso de la aplicación web desarrollada.
- **Gestión de proyectos (GPR)**: Asignatura en la que se presentan diferentes conceptos relacionados con el ciclo de vida de un proyecto (planificación, los recursos empleados, seguimiento etc).
- **Bases de datos (BDA):** Aunque esta asignatura se centra en el modelo relacional, cabe destacar que se da alguna pincelada a los conceptos del lenguaje NoSQl, que es el utilizado en este proyecto.
- **Interfaces persona computador (IPC):** Asignatura sobre el análisis de requisitos, el diseño conceptual, visual, prototipado, evaluación de interfaces... Esta, junto a ISW han sido una de las asignaturas que más han ayudado para el desarrollo del software.
- **Ingeniería del Software (ISW):** Asignatura que nos da a conocer los principios fundamentales del desarrollo del software. Para este proyecto se han utilizado muchos de los conocimientos que se imparten en la misma, entre ellos: la metodología de desarrollo, el diseño de interfaz de usuario, el modelado de casos de uso y diagramas y el proceso de pruebas.

Bibliografía

[1] Salvador Petit. *WebSpim*. URL: http://www.disca.upv.es/spetit/spim.htm (visitado 20-07-2021).

[2] Spim. *Website to download spim*. URL: https://sourceforge.net/p/spimsimulator/code/HEAD/tree/spim/ (visitado 10-07-2021).

[3] James Larus. *Spim*. URL: http://spimsimulator.sourceforge.net (visitado 10-07-2021).

[4] David A. Patterson y John L. Henessy. *Estructura y diseño de computadores: la interfaz hardware/software*. spa. Segunda edición. Editorial Reverté, 2011.

[5] qtio. *QT*. URL: https://www.qt.io (visitado 20-07-2021).

[6] QtSpim. *Website to download QtSpim*. URL: https://sourceforge.net/projects/spimsimulator/files/ (visitado 10-07-2021).

[7] ShawnZhong. *JsSpim*. URL: https://github.com/ShawnZhong/JsSpim (visitado 25-07-2021).

[8] Mónica Morán y Rubén Barrado. *Especificación de Requisitos Software según el estándar IEEE 830*. URL: https://wikis.fdi.ucm.es/ELP/Especificaci%C3%B3n_de_Requisitos_Software_seg%C3%BAn_el_est%C3%A1ndar_IEEE_830 (visitado 19-07-2021).

[9] Michael Arias Chaves. «La ingeniería de requerimientos y su importancia en el desarrollo de proyectos de software». En: *InterSedes: Revista de las Sedes Regionales* 6.10 (2005), págs. 1-13.

[10] uml.org. *UML*. URL: https://www.uml.org/ (visitado 10-07-2021).

[11] Wikipedia. *Caso de uso — Wikipedia, La enciclopedia libre*. URL: https://es.wikipedia.org/w/index.php?title=Caso_de_uso&oldid=130350776 (visitado 21-07-2021).

[12] app.diagrams.net. *draw.io*. URL: https://app.diagrams.net/ (visitado 15-07-2021).

[13] Balsamiq.com. *Balsamiq*. URL: https://balsamiq.com/ (visitado 10-08-2021).

[14] visualctudio.com. *Visual Studio Code*. URL: https://code.visualstudio.com/ (visitado 04-08-2021).

[15] Git-scm.com. *About*. URL: https://git-scm.com/about (visitado 05-08-2021).

[16] GitHub.com. *About*. URL: https://github.com/about (visitado 05-08-2021).

[17] GitHub. *Choose the plan that's right for you.* URL: https://github.com/pricing#feature-comparison (visitado 05-08-2021).

[18] Desktop.github. *Github Desktop*. URL: https://desktop.github.com/ (visitado 21-08-2021).

[19] MDN contributors. *¿Qué es Javascript?* URL: https://developer.mozilla.org/es/docs/Web/JavaScript/About_JavaScript (visitado 29-07-2021).

[20] Javier Blanco. *JavaScript ¿Por qué es tan popular?* URL: https://agencia6.com/index.php/2019/09/01/javascript-por-que-es-tan-popular/ (visitado 29-07-2021).

[21] MongoDB. *What is the MERN Stack?* URL: https://www.mongodb.com/mern-stack (visitado 29-07-2021).

[22] Emscripten Contributors. *About Emscripten*. URL: https://emscripten.org/docs/introducing_emscripten/about_emscripten.html (visitado 04-08-2021).

[23] Reactjs.org. *React*. URL: https://es.reactjs.org/ (visitado 04-08-2021).

[24] React. *Introducción a React*. URL: https://es.reactjs.org/tutorial/tutorial.html (visitado 10-07-2021).

[25] Create React APP. *Crear una nueva aplicación React*. URL: https://es.reactjs.org/docs/create-a-new-react-app.html (visitado 04-08-2021).

[26] Angular.io. *Introduction to Angular concepts*. URL: https://angular.io/guide/architecture#:~:text=Angular%20is%20a%20platform%20and,Angular%20is%20written%20in%20TypeScript.&text=Components%20define%20views%2C%20which%20are,your%20program%20logic%20and%20data. (visitado 20-07-2021).

[27] Vuejs.org. *Introduction*. URL: https://vuejs.org/v2/guide/ (visitado 20-07-2021).

[28] material.com. *Material UI*. URL: https://material-ui.com/ (visitado 04-08-2021).

[29] React-Codemirror. *React-CodeMirror*. URL: https://uiwjs.github.io/react-codemirror/ (visitado 05-08-2021).

[30] Axios.com. *Axios*. URL: https://axios-http.com/ (visitado 05-08-2021).

[31] Ahooks.js.org. *Ahooks*. URL: https://ahooks.js.org/ (visitado 05-08-2021).

[32] Heroku.com. *About*. URL: https://www.heroku.com/about (visitado 31-08-2021).

[33] Heroku. *Heroku Pricing*. URL: https://www.heroku.com/pricing (visitado 31-08-2021).

[34] Netfily. *Netfily*. URL: https://www.netlify.com/ (visitado 31-08-2021).

[35] Netfily. *Netfily Pricing*. URL: https://www.netlify.com/pricing/ (visitado 31-08-2021).

[36] Postman.com. *Postman*. URL: https://www.postman.com/ (visitado 31-08-2021).

[37] LaTeX. *About*. URL: https://www.latex-project.org/about/ (visitado 15-08-2021).

[38] Alon Zakai. *Emscripten & asm.js: C++'s role in the modern web*. The C++ conference (CppCon). 2014. URL: http://kripken.github.io/mloc_emscripten_talk/cppcon.html#/27 (visitado 02-08-2021).

[39] The Hungry Brain. *Web workers in action: why they're helpful, and how you should use them*. URL: https://www.freecodecamp.org/news/web-workers-in-action-2c9ff33be266/ (visitado 20-07-2021).

[40] MDN contributors. *Web workers in action: why they're helpful, and how you should use them*. URL: https://developer.mozilla.org/es/docs/Web/API/Worker (visitado 20-07-2021).

[41] Consejo General de Colegios Profesionales de Ingeniería Informática (CCII). «Estudio Nacional sobre la Situación Laboral de los Profesionales del Sector TI». En: (2015). URL: https://www.coiicv.org/actualidad/descargas?task=download.send&id=470&catid=4&m=0.

[42] UPV Servei de Recursos Humans. *Tablas retributivas del PDI*. URL: http://www.upv.es/entidades/SRH/menu_urlv.html?/entidades/SRH/retribuciones/U0894376.pdf (visitado 01-09-2021).

[43] MongoDB.com. *MongoDB Pricing*. URL: https://www.mongodb.com/pricing (visitado 20-07-2021).

APÉNDICE A
Configuración de MongoDB Atlas

Para la creación de la cuenta y configuración de MongoDB se tienen que realizar los siguientes pasos:

1. Accedemos a la página oficial de MongoDB (https://www.mongodb.com/es)

2. Creamos una cuenta a través del botón "Prueba gratuita" (que se muestra en la figura A.1), añadiendo los campos del registro que se nos solicitan.

Figura A.1: Botón para crear una prueba gratuita.

3. Una vez nos hayamos registrado y conseguido iniciar sesión, pasaremos a crear el cluster que alojará la base de datos. De esta manera, accederemos a la pantalla de "Databases" y ahí pulsaremos al botón de "Create". En la figura A.3 se pueden ver señalado el recorrido que se debe de seguir para este paso.

Figura A.2: Pasos para crear un *cluster*

4. Seleccionamos el proveedor que queremos utilizar, en este caso se ha optado por el proveedor AWS[1] y elegimos la región donde se alojará la base de datos, en nuestro caso se ha elegido *Irlanda(eu-west-1)*. En la figura A.3 se puede ver señalado el recorrido que se debe seguir para este paso.

[1]Servicios proporcionados por la compañía Amazon, referentes a la computación de la nube.

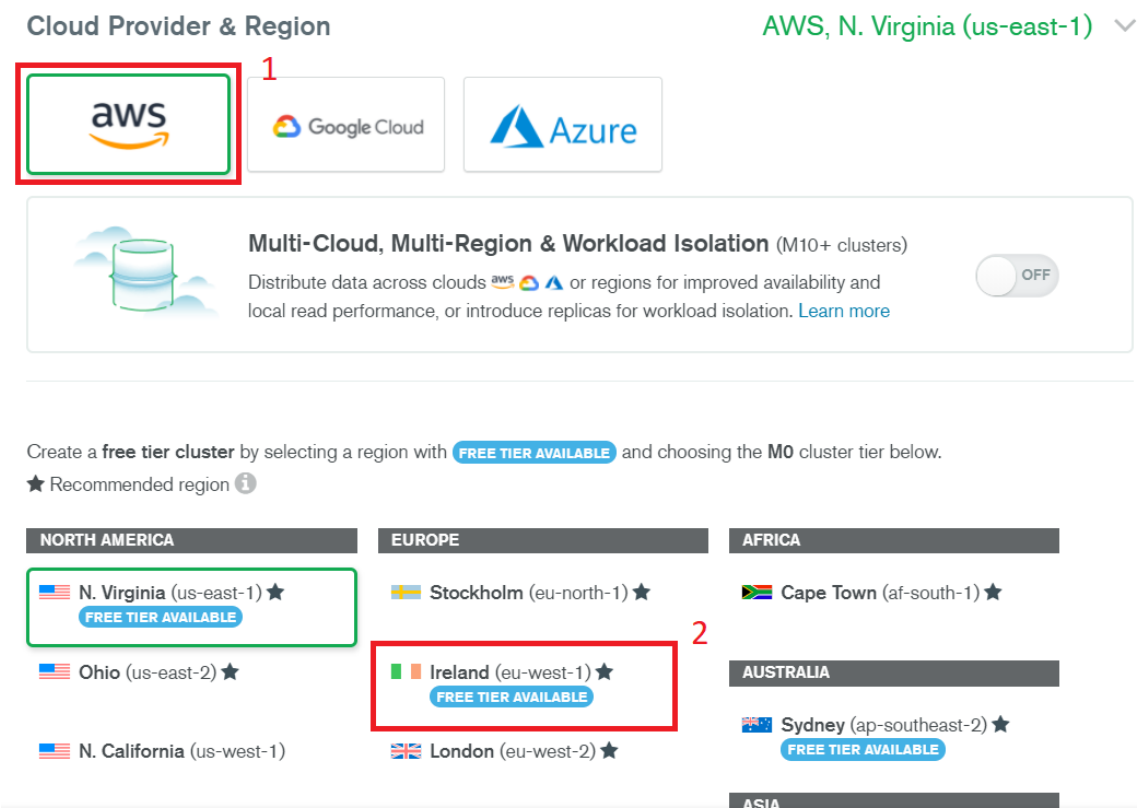

Figura A.3: Opciones elegidas para el *cluster*

5. Una vez creado el cluster pasaremos a añadir un usuario para que pueda tener permisos de escritura y lectura en la base de datos. De esta manera, accederemos a la página referente a "Security" concretamente al apartado de "Database Access". Una vez allí, pulsaremos el botón que se muestra en la figura A.4.

Figura A.4: Añadir un nuevo usuario en la base de datos

6. Aparecerá una pantalla para añadir un nuevo usuario, elegiremos un método de autenticación y rellenaremos lo que se nos solicite. En este caso, se ha optado por el método de contraseña, con lo que se tendrá que elegir tanto un nombre de usuario como una contraseña que luego será utilizada. En la figura A.5 se puede observar el proceso para realizar este paso.

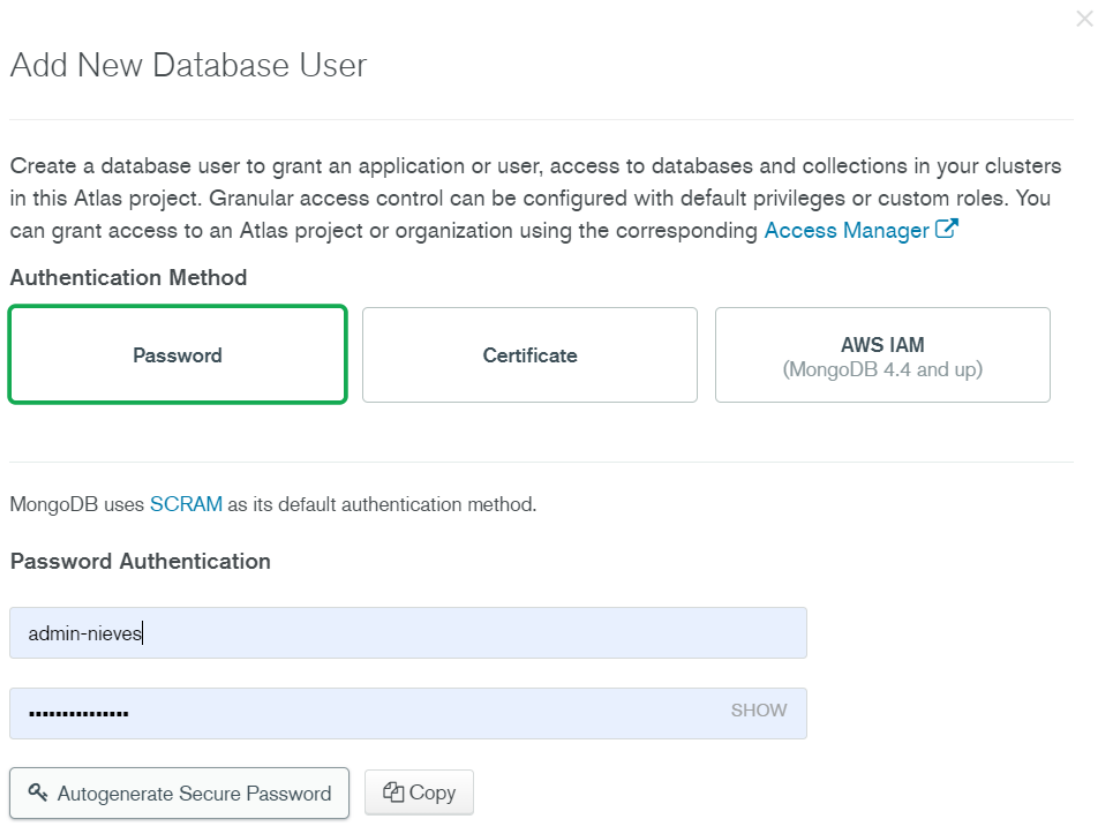

Figura A.5: Añadir un nuevo usuario en la base de datos

7. Una vez que se ha añadido un usuario se elegirán los privilegios que este tendrá. En la figura A.9 se puede observar que se le han otorgado permisos de escritura y lectura a cualquier base de datos que se encuentre en este cluster.

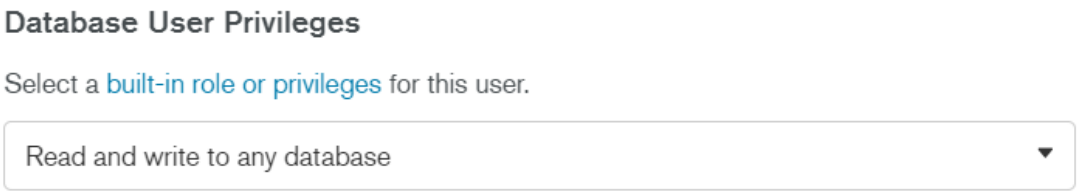

Figura A.6: Añadir privilegios al usuario creado

8. Inmediatamente después de la creación del usuario, accederemos al apartado "Network Access" y pulsaremos en el botón "ADD IP ADDRESS", donde añadiremos la IP donde estará el servidor para acceder a la base de datos. En la figura A.7 se pueden observar los campos que existen.

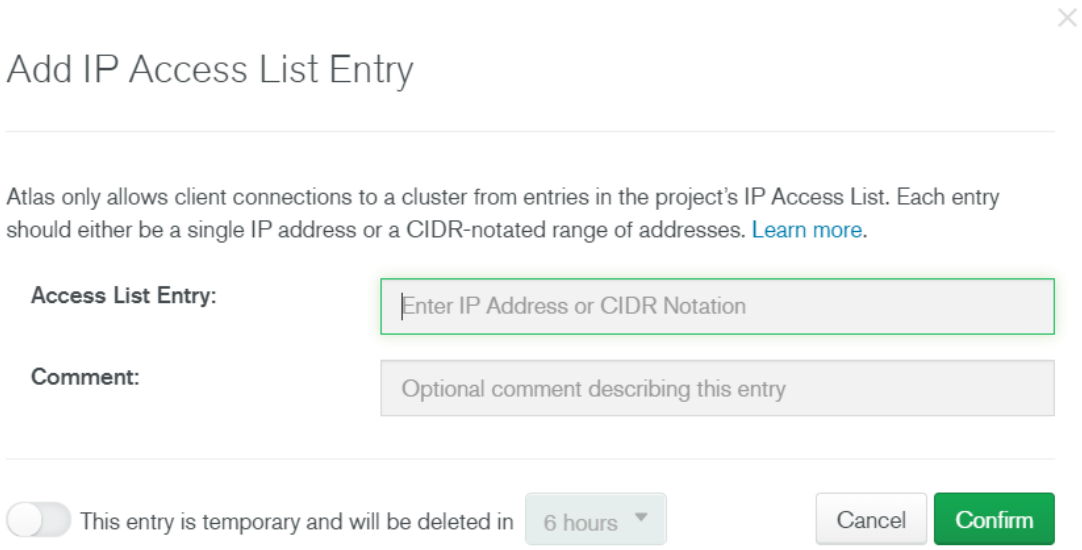

Figura A.7: Añadir las IPs que se permiten para acceder a la DB

9. Justo a continuación, volveremos a la pestaña de "Deployment" concretamente a la sección de "Database". Nos aparecerá el cluster y habrá un botón "Connect" el cual pulsaremos. Este *clic* nos dará la posibilidad de conectar la aplicación con el cluster, en nuestro caso utilizaremos la segunda opción representada con un cuadro rojo en la figura A.8.

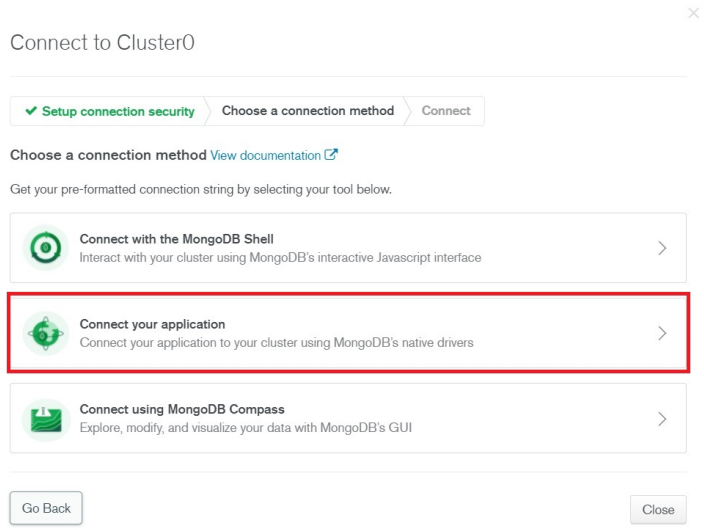

Figura A.8: Conectar la aplicación con el cluster

10. Por último, copiaremos el *string* que nos proporciona la aplicación, cambiaremos la parte de "<password>" por la contraseña utilizada para el usuario *admin* y lo añadiremos en la aplicación. En la figura se puede observar el *string*.

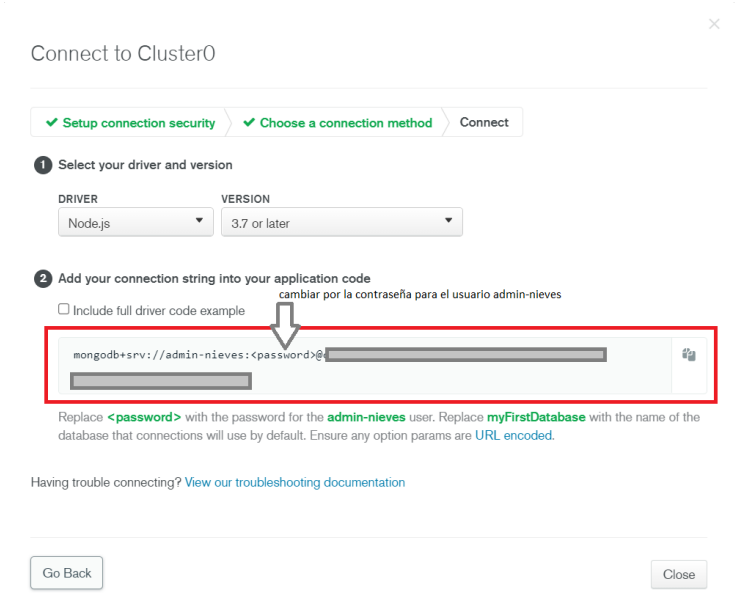

Figura A.9: String con la conexión a la base de datos