UNIVERSIDAD
REY JUAN CARLOS

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE LA TELECOMUNICACIÓN

Curso Académico 2015/2016

Trabajo Fin de Carrera/Grado/Máster

APLICACIÓN WEB DE ANÁLISIS DE PROYECTOS JAVASCRIPT

Autor : Adrián Viñuelas Pereda

Tutor : Dr. Gregorio Robles

Trabajo Fin de Grado

Aplicación Web de Análisis de Proyectos JavaScript

Autor : Adrián Viñuelas Pereda

Tutor : Dr. Gregorio Robles Martínez

La defensa del presente Proyecto Fin de Carrera se realizó el día de junio de 2016, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de junio de 2016

Dedicado a

mis abuelos Luis y Conchi.

Me encantaría que hubierais podido ver lo que he conseguido.

Os echo de menos. Os quiero y os querré siempre.

II

Agradecimientos

Quisiera darle las gracias a mi familia, tanto a los que están como a los que no están. Gracias a ellos hoy soy la persona que soy, en especial a mis padres. Mis padres, Rafa y Rosa, que siempre me lo han dado todo y sin los cuales nunca habría conseguido nada. Ellos siempre me han apoyado en todas mis decisiones y me han ayudado para que hoy en día pueda tener una profesión. Desde que acabé la E.S.O. y decidí seguir estudiando ellos siempre me han apoyado y han respetado mis decisiones. Siempre me han dicho que hay que tener paciencia, que todo llega. Que razón tienen. Sois los mejores. Os quiero.

Por otro lado, me gustaría agradecer de una forma especial a mi novia Melina. Apareciste en el momento justo. Desde el primer momento me has apoyado y animado a continuar esforzándome para conseguir lo que quiero. Sin ti tampoco lo habría logrado. Me has demostrado ser un ejemplo de superación. No he visto nunca a nadie ser tan fuerte como tu, y tener a alguien así a mi lado me ha hecho ser a mi más fuerte. Gracias por todo lo que haces por mi, por ser mi gran apoyo, y por hacerme feliz. Te quiero.

No quisiera olvidarme de mis compañeros de viaje en la universidad, Iván y Kevin, con los que tantos momentos buenos he compartido, al igual que tantas horas de estudio juntos. A ellos también tengo mucho que agradecerles, remando siempre juntos.

Gracias a todos, porque se que os alegráis siempre más que yo por todo lo bueno que me sucede.

Resumen

Este proyecto tiene como objetivo mejorar el desarrollo de habilidades de estudiantes a la hora de escribir programas usando código Javascript. Se ha creado una herramienta que permite obtener una visión del progreso durante el curso, tanto general de toda la clase como específica de cada alumno al profesor. Es decir, por un lado se intenta ayudar al profesor a tener una idea de la evolución de sus alumnos para poder afrontar las clases de la mejor manera posible, ayudando a estos a mejorar sus habilidades, y por otro lado se intenta que los alumnos se den cuenta de cuales son sus errores y cuales son las mejores técnicas para escribir código Javascript y continuar progresando en su formación.

Para ello se ha creado una aplicación web en la que tanto alumnos como profesores puedan incluir el ejercicio que se quiere analizar (en el caso del rol de profesor analiza los ejercicios de toda la clase) y posteriormente la aplicación realiza y muestra un análisis detallado de todos los errores que encuentra en el ejercicio. Además permite realizar un seguimiento a lo largo del tiempo para ver la progresión de los alumnos.

Se han empleado tecnologías como Bootstrap, jQuery y jQuery-UI para el desarrollo de la web. Por otro lado con Django se ha implementado el servidor (implementado en Python), y con AJAX se han realizado las comunicaciones entre servidor y cliente (implementado en Javascript). Con Highcharts se han podido realizar las gráficas, y con JSLint y JSHint se ha realizado el análisis del código Javascript.

Summary

This project aims to improve the development and skills of students when writing programs in Javascript code during his formative years at the university, and give a vision of the progess during the course, both general of the whole class as specific to each student to teacher. That is, on the one hand you try to help the teacher to get an idea of the evolution of their students, in order to meet classes in the best way possible to help their students improve their skills, and on the other hand is intended that students realize what are their mistakes and what are the best techniques to write Javascript code and continue to progress in their training.

For this reason I have created a web application in which both students and teachers to include exercise to be analyzed (for the role of teacher analyzes the exercises of the class) and then the application performs and shows a detailed analysis of all errors in the exercise. It also allows to track over time to see the progression of students.

Technologies have been used as Bootstrap, jQuery, jQuery-UI for web development. On the other hand with Django I implemented the server (implemented in Python), and I have made AJAX communications between server and client (implemented in Javascript). With Highcharts I could make graphs, and with JSHint and JSLint I have made the analysis of Javascript code.

Índice general

Índice de figuras

Capítulo 1

Introducción

Este proyecto está pensado para ayudar a los futuros desarrolladores, y que actualmente son estudiantes, a mejorar su formación y habilidades a la hora de desarrollar código Javascript. Su finalidad es totalmente educativa, y tiene que quedar claro que es una herramienta orientativa, puesto que los errores que muestra no hay que interpretarlos como que el código está desarrollado mal, ya que puede que funcione perfectamente.

Es una aplicación web, que sirve de herramienta tanto a alumnos como a profesores. La aplicación busca dar una orientación a los estudiantes para que desarrollen códigos que puedan ser legibles y entendibles por una persona que tenga que usarlo posteriormente. Es verdad que los alumnos solo hacen ejercicios semanales y prácticas finales, pero en la vida laboral normalmente se trabaja con códigos desarrollados por varias personas, y que por tanto tienen que estar claros y ser perfectamente legibles, independientemente de que funcionen correctamente, ya que el entender un código fácilmente puede ahorrar muchas horas de trabajo a la persona que lo lee por primera vez.

Al principio yo mismo no entendía muy bien el porqué de darle tanta importancia por parte de profesores que nos animaban a escribir nuestros códigos de una manera ordenada y clara. Cuando comencé el proyecto, y a medida que iba avanzándolo, me daba cuenta de que realizar ciertas acciones simples puede mejorar nuestra forma de desarrollar nuestro código. ¿A quién no le ha pasado escribir un código de un programa, dejarlo apartado durante un tiempo, volverlo a retomar y decir: “No entiendo que hice aquí”? Creo que no somos conscientes cuando escribimos nuestros programas de que con el tiempo, habremos hecho tantos programas que es muy difícil acordarse de todas las funcionalidades que hemos ido implementando. Por esta razón,

acciones tan sencillas como estructurar bien el código, dar buenos nombres a nuestras variables y funciones, e incluso usar correctamente espacios y tabulaciones, pueden facilitarnos mucho el trabajo el día de mañana.

Es una idea muy simple pero que la mayoría no implementamos correctamente. Y es importante que desde la base de la formación se cojan buenos hábitos.

Capítulo 2

Objetivos

2.1. Objetivo general

Llega un momento en el que no solo tienes que tratar con códigos desarrollados por ti mismo, si no que tienes que enfrentarte a códigos desarrollados por otras personas, los cuales tienes que analizar para poder trabajar con ellos. Y es en ese momento cuando uno se da cuenta de lo importante que es desarrollar un código limpio y legible, para que cuando otras personas tengan que utilizarlo, sepan como interpretarlo fácilmente dentro de la dificultad que tiene el que muy probablemente no haya dos personas que utilicen las mismas técnicas a la hora de desarrollar su código, ya sea en el lenguaje de programación que sea.

Por esta razón, el objetivo general del proyecto es transmitir a los usuarios (principalmente los estudiantes que son quienes están aprendiendo a desarrollar código) de la aplicación que tan importante es que funcione el código que estamos escribiendo, como que sea entendible por otros usuarios. Muchas veces, cuando nos preguntan que si podemos echar una mano a alguien, cuando nos encontramos ante un código que contiene muchas líneas, si el nombre de las variables, las tabulaciones, los espacios, o nombres de funciones por ejemplo no nos ayudan a entender que se está haciendo o que es lo que se quiere hacer, es muy difícil entender un programa. Y es peor aún cuando te pasan un código que no es legible, y que además no puedes contactar con la persona que lo ha desarrollado para que te explique el por qué de que el código esté así desarrollado.

Conseguir transmitir la idea de que un código limpio, legible, bien estructurado, y que ayude a entender lo que se está haciendo es tan importante como que funcione bien. Y si este trabajo

se comienza a hacer desde que los alumnos comienzan su formación, se conseguirá que en el futuro encontremos buenos códigos desarrollados correctamente dentro de unas ciertas pautas. La idea también es hacer entender que seguir las pautas que creamos correctas para desarrollar un buen código no tienen por qué ser las mismas para todos los usuarios, pero que si todos seguimos un mismo camino será más fácil que todos desarrollemos buenos códigos. Es como aprender a escribir, a todos nos enseñan desde pequeños pero no todos nos expresamos igual. La idea es expresarse lo mejor posible para que los demás nos entiendan. Con los códigos pasa exactamente lo mismo. En ningún caso debemos tomar las pautas que se establecen en este proyecto como reglas estrictas que haya que seguir pero si a modo de consejo. Simplemente como una ayuda para que nuestro código sea accesible para todos.

2.2. Objetivos específicos

Centrándome más en unos objetivos específicos que me gustaría conseguir, podemos diferenciar dos: uno a nivel de profesor y otro a nivel de alumno.

Empezando por la parte del profesor, lo que se ha intentado buscar es una forma de poder dar al profesor tanto una visión general de como toda la clase (los alumnos) desarrolla el código Javascript de los ejercicios que semanalmente entregan, como una visión específica de cada alumno en particular. El objetivo para la parte del profesor es que este pueda realizar un seguimiento de todos sus alumnos por cada ejercicio entregado y poder ver como estos desarrollan su código Javascript. Por cada ejercicio semanal, se realiza un análisis tanto a nivel colectivo como individual de los errores (que realmente no son errores pero que los trataremos como tales) que los alumnos han cometido, y se da una idea al profesor de que aspectos deberían mejorar a la hora de desarrollar el código, ya sea usar espacios en lugar de tabulaciones, declarar variables antes de usarlas (puesto que en Javascript no se tiene por qué declarar variables antes de usarlas) entre otros muchos aspectos que se podrían mejorar y en los que el profesor pueda hacer hincapié para intentar que sus alumnos mejoren. Creo que a los profesores les gusta ver que los ejercicios de sus alumnos, además de que funcionen, estén bien desarrollados y estructurados.

En segundo lugar, centrándonos en la parte del alumno, el objetivo es quizás más difícil, puesto que para los alumnos lo primero es ver que lo que escriben en su código funciona y lo demás es un poco más secundario para ellos, por lo que el reto es mucho mayor. Yo también

he sido alumno y esa era mi principal preocupación, pero en mi primer curso hice un programa enorme que funcionaba perfectamente y hacía todo lo que se me pedía, pero suspendí porque era totalmente ilegible (la inexperiencia del primer año). En su momento me pareció muy injusto, pero luego entendí que tan importante es que funcione como que se pueda leer y entender por otros usuarios. Por eso busco ayudar a los alumnos para que entiendan desde el primer momento que tienen que esforzarse también por desarrollar un código legible y limpio. Creo que si se empieza la formación por una buena base y se acostumbra a los futuros alumnos a escribir su código siguiendo una serie de pautas, el día de mañana cuando nos pasen un código que no es nuestro (y que es el pan de cada día en muchos trabajos) el esfuerzo por entender lo que hace el programa y como lo hace será mucho mas rápido y fácil.

2.3. Planificación temporal

La forma en la que me he organizado para realizar este proyecto ha sido la siguiente:

1. Reunión con mi tutor para aclarar todos los aspectos necesarios para poder arrancar el proyecto (tanto el tema del proyecto como qué tecnologías podía utilizar para ir arrancando la aplicación). Me documenté sobre la herramienta JSLint ya que nunca antes había trabajado con ella.

2. Realicé una primera versión, sin usar Bootstrap, en la que implementé la funcionalidad básica del profesor. Comprobaba los alumnos que habían entregado el ejercicio, y descargaba uno a uno (según el alumno que seleccionaba) los ejercicios, los analizaba (solamente usando JSLint) y por el momento no contemplaba los errores.

    - A su vez iba desarrollando un servidor muy básico que con el desarrollo de la aplicación fue cogiendo la forma.

3. Pasar la primera a versión a una segunda versión que incluía Bootstrap, para dar una forma más moderna y atractiva al proyecto. De momento sólo con la funcionalidad del profesor, aún por mejorar. Este paso fue adaptar lo que ya tenía a una versión más moderna visualmente.

4. Incluí la funcionalidad del alumno y mejoré la del profesor para que analizara todos los

ejercicios de sus alumnos de manera automática cuando indicara que ejercicio quería analizar.

   - Hasta este momento todavía no había incluido ni JSHint, ni Highcharts.

5. Introducción de una segunda herramienta que complementara el análisis realizado por JSLint, ya que para ciertos alumnos, al cometer errores que JSLint consideraba graves la propia herramienta paraba el análisis. Hablo de JSHint. Ambas herramientas no miden lo bueno o malo que es un código Javascript, simplemente ayudan a mejorar y orientar al desarrollador para mejorar su código y hacerlo más legible y eficaz.

6. Para que el análisis de los errores fuera más visual, introduje finalmente Highcharts que me permitía utilizar gráficas para mostrar el número de errores de cada tipo que había identificado la aplicación. También aproveché esta herramienta para facilitar el seguimiento de la evolución a lo largo del tiempo que muestro al usuario en su menú principal.

7. Una última mejora introducida al final, y aprovechando el uso de Highcharts, fue la implementación de una gráfica en la que se comparaban los errores del ejercicio actual, con los errores del último ejercicio analizado, y así ver en que errores se había mejorado y en cuales empeorado respecto al ejercicio anterior. Se busca con esto una forma de motivar a los usuarios (sobre todo a los alumnos) a mejorar sus códigos para reducir el número de errores y por otro lado que resulte más atractiva la aplicación para ellos.

8. Finalmente, y como un pequeño detalle, decidí dar la opción al usuario de introducir librerías que pueden ser usadas en los ejercicios (pero que no están implementadas por los alumnos, y que por tanto su análisis no es de interés) para que se descarte su análisis, y que por tanto el servidor identificará de que ficheros se trata ya que estarán en la base de datos guardados y la aplicación omitirá su análisis.

Capítulo 3

Estado del arte

Las tecnologías utilizadas son: Bootstrap, jQuery, jQuery-UI, Highcharts, JSLint, JSHint, Django, y AJAX.

3.1. Bootstrap

Bootstrap[1] es un framework o conjunto de herramientas de código abierto para diseño de sitios y aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS.

Para usar Bootstrap en una página HTML, el desarrollador solo debe descargar la hoja de estilo Bootstrap CSS y enlazarla en el archivo HTML. Permite diseñar las aplicaciones web tanto para móviles, ordenadores o *tablets* sin tener que modificar la hoja de estilo CSS de nuestro proyecto, puesto que el estilo viene definido en una serie de clases. Solo con darle una clase a nuestro objeto este ya aplica sus propiedades, lo cual facilita mucho el trabajo.

3.2. jQuery

jQuery[2] es una biblioteca de Javascript que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web. jQuery es software libre y de código

[1] `http://getbootstrap.com`

[2] `https://jquery.com/`

abierto permitiendo su uso en proyectos libres y privados. Ofrece una serie de funcionalidades basadas en Javascript que de otra manera requerirían de mucho más código, es decir, con las funciones propias de esta biblioteca se logran grandes resultados en menos tiempo y espacio.

jQuery consiste en un único fichero Javascript que contiene las funcionalidades comunes de DOM, eventos, efectos y AJAX. Su característica principal es que permite cambiar el contenido de una página web sin necesidad de recargarla, mediante la manipulación del árbol DOM y peticiones AJAX. Para ello utiliza las funciones de un API fácil de usar que funciona a través de una multitud de navegadores.

3.3. jQuery-UI

jQuery-UI[3] es una biblioteca de componentes para el framework jQuery que le añaden un conjunto de plug-ins, widgets y efectos visuales para la creación de aplicaciones web. Cada componente o módulo se desarrolla de acuerdo a la filosofía de jQuery.

jQuery-UI es un conjunto de interacciones de interfaz de usuario, efectos, widgets y temas creados por encima de la biblioteca de Javascript jQuery. El uso de los diferentes módulos de jQuery-UI te permite realizar una aplicación altamente interactiva.

3.4. Highcharts

Highcharts[4] es una librería escrita en Javascript que permite la creación de gráficas. La librería ofrece un método fácil e interactivo para insertar gráficas en un sitio web o aplicación web. La librería es compatible con todos los navegadores modernos incluyendo iPhone/iPad e Internet Explorer desde su versión 6.

Es abierto, todas las características pueden ser personalizadas permitiendo una gran flexibilidad. Además Highcharts está escrito solamente con código Javascript, sólo se requiere incluir el archivo highcharts.js y cualquiera de los tres frameworks más populares de Javascript (jQuery, MooTools o Prototype), que en nuestro caso es jQuery.

[3] `https://jqueryui.com`

[4] `http://www.highcharts.com`

3.5. JSLint

JSLint es una herramienta de análisis estático de código utilizada en el desarrollo de software para comprobar si el código fuente Javascript cumple con las normas de codificación. Se proporciona principalmente como una herramienta en línea, pero también hay adaptaciones de línea de comandos.

Realiza la lectura del código fuente y devuelve observaciones o puntos en los que el código puede mejorarse desde la percepción de buenas prácticas de programación y código limpio. Nos permitirá mostrar puntos en los que el código no cumpla unas determinadas reglas establecidas de "código limpio".

3.6. JSHint

JSLint no es una herramienta óptima ya que da muchos falsos positivos. Además los criterios evaluados son bastante subjetivos según el punto de vista del usuario. Por ello, algunos desarrolladores crearon un fork llamado JSHint. El objetivo de JSHint es mejorar las mediciones que son bastante arbitrarias en JSLint.

La utilización de estas herramientas debe ser prudente, a modo de consejo sobre nuestro código fuente y no como reglas estrictas. En todo caso, no hay que ofuscarse pensando que nuestro código está absolutamente bien o mal según los resultados obtenidos.

3.7. AJAX

AJAX[5] es una técnica de desarrollo web para crear aplicaciones interactivas. Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, mejorando la interactividad, velocidad y usabilidad en las aplicaciones.

AJAX es una tecnología asíncrona, en el sentido de que los datos adicionales se solicitan al servidor y se cargan en segundo plano sin interferir con la visualización ni el comportamiento

[5] `http://api.jquery.com/jquery.ajax/`

de la página, aunque existe la posibilidad de configurar las peticiones como síncronas de tal forma que la interactividad de la página se detiene hasta la espera de la respuesta por parte del servidor.

3.8. Django

Django[6] es un framework de desarrollo web de código abierto, escrito en Python. La meta fundamental de Django es facilitar la creación de sitios web complejos. En lugar de requerir que los desarrolladores escriban controladores y vistas para las áreas de administración de la página, Django proporciona una aplicación incorporada para administrar los contenidos, que puede incluirse como parte de cualquier página hecha con Django y que puede administrar varias páginas hechas con Django a partir de una misma instalación; la aplicación administrativa permite la creación, actualización y eliminación de objetos de contenido, llevando un registro de todas las acciones realizadas sobre cada uno, y proporciona una interfaz para administrar los usuarios y los grupos de usuarios (incluyendo una asignación detallada de permisos).

También permite administrar bases de datos. En mi caso he usado bases de datos MySQL, pero también permite usar otras bases de datos, como PostgreSQL, SQLite, Microsoft SQL Server y Oracle. Django proporciona una serie de funciones para el manejo de estas bases de datos, y permite crear objetos, guardarlos, modificarlos, leerlos y eliminarlos entre otras funciones.

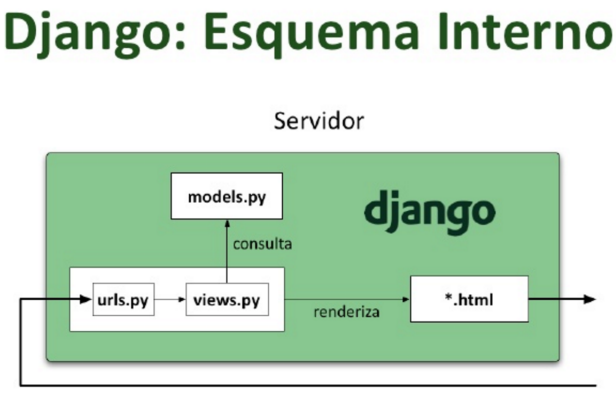

Figura 3.1: Esquema interno Django.

[6] https://www.djangoproject.com

Capítulo 4

Diseño e implementación

A continuación se va a proceder a explicar en detalle cómo se han desarrollado las distintas funcionalidades de este proyecto.

4.1. Arquitectura general

El proyecto está basado en una primera comunicación por parte del cliente que realiza una petición GET HTTP al servidor para cargar la página principal. Después toda la comunicación entre cliente y servidor se realiza mediante AJAX, usando POST en los que el cliente envía los datos que la acción requiera al servidor para que este los trate de acuerdo a la necesidad del POST del cliente. Posteriormente el servidor le devuelve la página renderizada incluyendo los datos necesarios, y es el cliente el que actualiza el HTML con la función "document.documentElement.innerHTML" de Javascript.

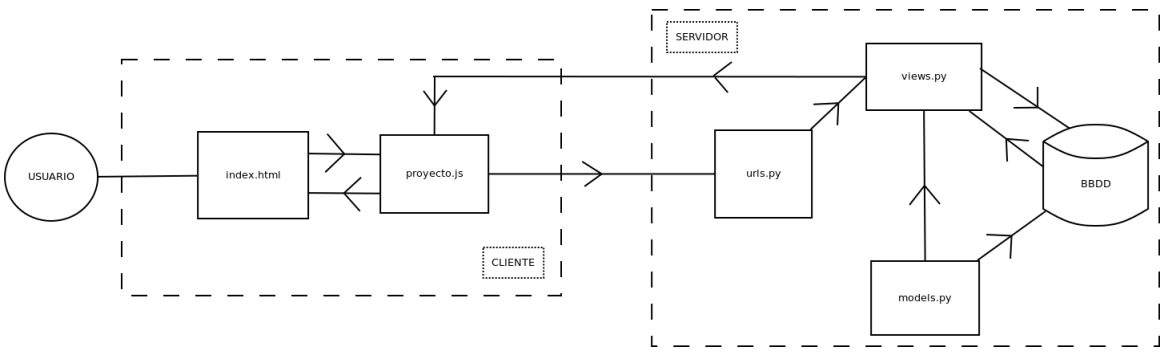

Figura 4.1: Esquema comunicación.

4.2. Base de Datos MySQL

Para entender el funcionamiento del proyecto, empezaré hablando de cómo está estructurada la base de datos. La base de datos está compuesta por: User, Usuario, TeacherExercise, StudentExercise, FichJs, Error, y Library.

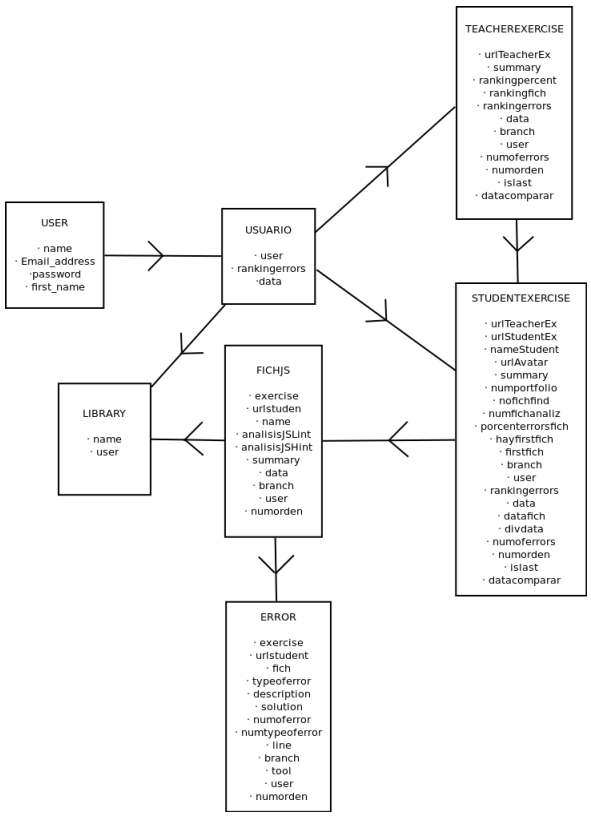

Figura 4.2: Diagrama con las tablas de la base de datos de la aplicación.

4.2.1. User

User es un objeto que Django proporciona por defecto. En él definimos el nombre (username), el correo electrónico (email address), y la contraseña (password) del usuario que se crea.

Por otro lado, los usuarios pueden tener el rol de profesor o de estudiante. Para ello aprovechamos este objeto y le incluimos el rol (first_name). Con esto tenemos identificado al usuario tanto por su nombre como por su rol. Cuando se loguea un usuario podemos comprobar la contraseña para ver si es correcta la autenticación.

4.2.2. Usuario

Este objeto puede parecer innecesario puesto que ya tenemos a un usuario definido por defecto en Django. Pero Django no permite guardar cierta información en ese objeto *User*. Por eso tengo definido el objeto Usuario que me permite almacenar la información relacionada con los propios usuarios de la aplicación. Almacena:

- el nombre de usuario (*user*, con el que puedo relacionarlo con el objeto *User*),
- el *ranking* de los 5 errores más comunes (rankingerrors),
- y la información necesaria para realizar la gráfica de la evolución (data).

4.2.3. TeacherExercise

Si el usuario de la aplicación tiene el rol de profesor, almacenará datos tanto de TeacherExercise como de StudentExercise (explicado más adelante en la memoria).

Un objeto TeacherExercise almacena toda la información relacionada con el ejercicio de entrega que el profesor propone (pero no de los ejercicios individuales de cada alumno). Es decir, guarda:

- la URL del ejercicio del profesor de GitHub (urlTeacherEx),
- el resumen de los errores que han tenido todos los alumnos (summary),
- el ranking de porcentaje de errores (rankingpercent), ordenando los alumnos de menor a mayor en función del número de errores y de los ficheros entregados. Por ejemplo, en cada fichero como máximo se pueden detectar 100 errores por cada herramienta que lo analiza, es decir, un total de 200 errores por fichero. Si un alumno solo tiene 1 fichero JavaScript analizado para ese ejercicio y ha tenido 40 errores, el porcentaje de errores para ese alumno es de un 20 %. Si por otro lado otro alumno ha entregado 2 ficheros y ha tenido 40 errores, el porcentaje de errores para este alumno es de un 10 % (más ficheros para el mismo número de errores, por tanto, menor porcentaje),
- el ranking de ficheros entregados (rankingfich), ordenando los alumnos de mayor a menor en función del número de ficheros entregados,

- el ranking de errores (rankingerrors), ordenando de mayor a menor los errores en función del número de errores totales de cada tipo que hayan tenido los alumnos para ese ejercicio,
- la información para crear mediante Highcharts la gráfica de los errores JSLint y JSHint (data),
- la rama a la que pertenece el ejercicio (branch), puesto que en GitHub hay dos ramas en las que se pueden subir los ejercicios, la rama máster (por defecto) o la rama gh-pages (que te crea un servidor en GitHub para ese ejercicio),
- el usuario al que pertenece el ejercicio (user), puesto que varios usuarios pueden pasar el mismo ejercicio por la aplicación,
- el número de errores totales que ha tenido el ejercicio (numoferrors), que luego se utilizará para la gráfica de evolución,
- el número identificativo para un mismo ejercicio (numorden), puesto que un usuario puede analizar un ejercicio las veces que quiera, por tanto hay que diferenciar cada análisis, ya que la información no se sobrescribe. Es decir, un usuario puede tener almacenados:
  - https://github.com/CursosWeb/X-Nav-APIs-Map-Flickr (1)
  - https://github.com/CursosWeb/X-Nav-APIs-Map-Flickr (2)
  - https://github.com/CursosWeb/X-Nav-APIs-Map-Flickr (numorden)
- un booleano que nos indica si es el último ejercicio analizado (islast) para saber con cual tenemos que comparar el ejercicio que estamos analizando en cada momento,
- la información necesaria para crear la gráfica de la comparación de los errores del ejercicio actual que se está analizando con el último ejercicio analizado (datacomparar).

4.2.4. StudentExercise

Este objeto guarda información relacionada con los ejercicios que los alumnos entregan en GitHub, por consiguiente, tanto el rol de profesor (los utiliza para realizar los rankings, resúmenes y gráficas de un ejercicio de profesor) como de estudiante manejan objetos de este tipo. Almacena los siguientes datos:

- en el caso del rol de profesor (y solo con este rol) se guarda la URL del ejercicio de profesor (urlTeacherEx) para luego poder relacionarlo con el mismo (objeto TeacherExercise),
- la URL del ejercicio del estudiante en GitHub (urlStudentEx),
- el nombre del alumno en GitHub (nameStudent),
- la URL del avatar del alumno en GitHub (urlAvatar),
- un resumen de los errores del alumno que solo utiliza el rol de profesor (summary),
- un número identificativo (numportfolio) para que cuando actualizo el HTML, pueda relacionar la foto del alumno con el contenido del ejercicio de este. Solo para rol de profesor,
- un booleano que indica si se han encontrado ficheros .js para analizar en el ejercicio (nofichfind),
- el número de ficheros analizados en el caso de que los haya (numfichanaliz),
- el porcentaje de errores que ha tenido el estudiante en el ejercicio (porcenterrorsfich),
- un booleano que nos indica si hay primer fichero analizado (hayfirstfich), que se utilizará a la hora de actualizar el HTML,
- el nombre del primer fichero analizado (firstfich), que se utilizará a la hora de actualizar el HTML,
- la rama a la que pertenece el ejercicio (branch),
- el usuario al que pertenece el ejercicio (user),
- el ranking de errores del ejercicio (rankingerrors), solo para el caso del rol de estudiante,
- la información necesaria para crear la gráfica de errores totales con Highcharts del ejercicio (data), solo para el caso del rol de estudiante,
- la información necesaria para crear la gráfica de errores con Highcharts del primer fichero analizado (datafich), solo para el caso del rol de estudiante,

- el nombre del div del HTML en el que se meterá la gráfica de los errores totales del ejercicio (divdata), solo para el caso del rol de estudiante,
- el número de errores totales que ha tenido el ejercicio (numoferrors), que luego se utilizará para la gráfica de evolución,
- el número identificativo para un mismo ejercicio (numorden), puesto que un usuario puede analizar un ejercicio las veces que quiera, por tanto hay que diferenciar cada análisis, ya que la información no se sobrescribe. Es decir, un usuario puede tener almacenados:
  - https://github.com/alumno/X-Nav-APIs-Map-Flickr (1)
  - https://github.com/alumno/X-Nav-APIs-Map-Flickr (2)
  - https://github.com/alumno/X-Nav-APIs-Map-Flickr (numorden)
- un booleano que nos indica si es el último ejercicio analizado (islast) para saber con cual tenemos que comparar el ejercicio que estamos analizando en cada momento,
- la información necesaria para crear la gráfica de la comparación de los errores del ejercicio actual que se está analizando con el último ejercicio analizado (datacomparar).

4.2.5. FichJs

Este objeto contiene la información relacionada con el análisis de un fichero .js. Almacena los siguientes datos:

- la URL del ejercicio del profesor, en el caso de que el usuario tenga rol de profesor (exercise),
- la URL del ejercicio del estudiante al que pertenece el fichero (urlstudent),
- el nombre del fichero (name),
- el análisis de JSLint (analisisJSLint),
- el análisis de JSHint (analisisJSHint),
- el resumen con el tipo de errores y el número de errores de cada tipo (summary),

- la información para crear la gráfica de los errores JSHint y JSLint con Highcharts (data),
- la rama a la que pertenece el fichero (branch), puesto que un mismo fichero puede estar en ambas ramas para un mismo ejercicio y entonces se considerarían ejercicios diferentes y, por tanto, ficheros diferentes,
- el usuario de la aplicación al que pertenece el fichero (user),
- el número identificativo para un mismo ejercicio (numorden), para saber a que ejercicio pertenece si se han analizado varios ejercicios iguales para un mismo estudiante, puesto que si se ha modificado el fichero, el nombre sigue siendo el mismo pero no su contenido y por tanto su análisis es diferente.

4.2.6. Error

Este objeto contiene la información relacionada con cada error que detecta la aplicación. Almacena los siguientes datos:

- la URL del ejercicio del profesor, en el caso de que el usuario tenga rol de profesor (exercise),
- la URL del ejercicio del estudiante al que pertenece el error (urlstudent),
- el nombre del fichero al que pertenece el error (fich),
- el tipo de error que es (typeoferror),
- la descripción del error (description),
- la URL donde se explica el error (solution),
- el número de error que ocupa en el análisis (numoferror),
- el número que identifica de que tipo de error se trata (numtypeoferror),
- la línea en la que está el error dentro del fichero (line),
- la rama a la que pertenece el error (branch), puesto que un mismo error puede estar en ambas ramas para un mismo ejercicio y fichero y entonces se considerarían ficheros diferentes y, por tanto, errores diferentes,

- la herramienta con la que se analizó el error, JSHint o JSLint (tool),
- el usuario al que pertenece (user),
- el número identificativo para un mismo ejercicio (numorden), para saber a que ejercicio pertenece si se han analizado varios ejercicios iguales para un mismo estudiante, puesto que si se ha modificado el fichero, el contenido del análisis es distinto y, por tanto, cada error solo puede pertenecer a un fichero y ejercicio distinto.

4.2.7. Library

Este objeto contiene información relacionada con cada librería que se quiere que no sea analizada. Aunque el nombre es Library, también guarda información de otros ficheros **.js** que no se quieran analizar. Almacena los siguientes datos:

- el nombre de la librería/fichero (name),
- el usuario al que pertenece (user).

4.3. Diseño e Implementación

4.3.1. Nuevos usuarios

El esquema del proceso de crear usuarios es el siguiente:

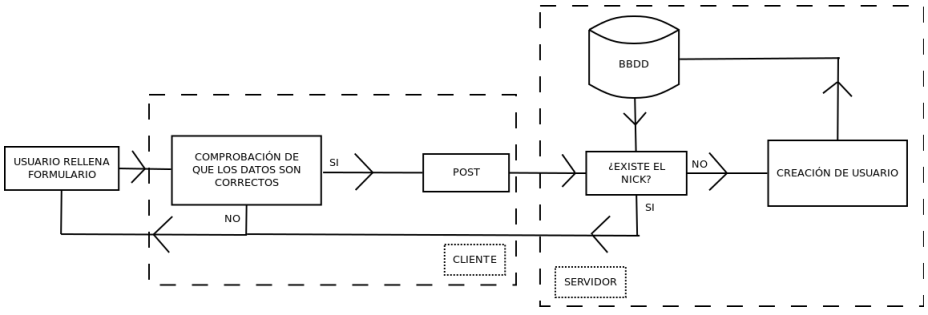

Figura 4.3: Diagrama proceso de crear usuario.

Lo primero que hay que hacer al usar la aplicación es crear un usuario. Los usuarios pueden tener el rol de profesor o el rol de estudiante. En la página de inicio creamos el usuario.

- **front-end:**

El usuario tiene que hacer click en New User y se le dará la opción de poder crear un usuario.

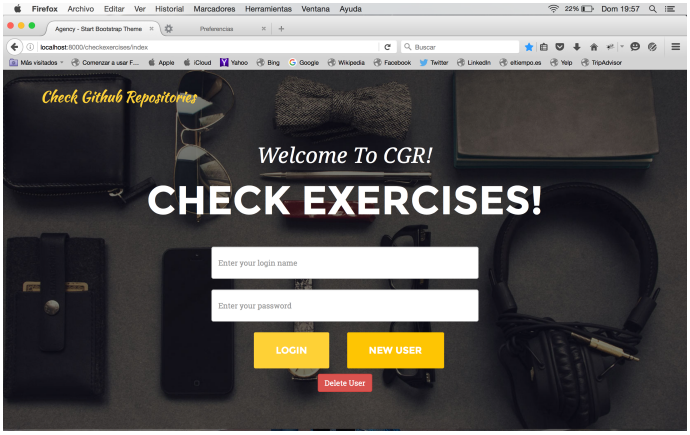

Figura 4.4: Formulario para loguearse, con opciones de crear y borrar usuario.

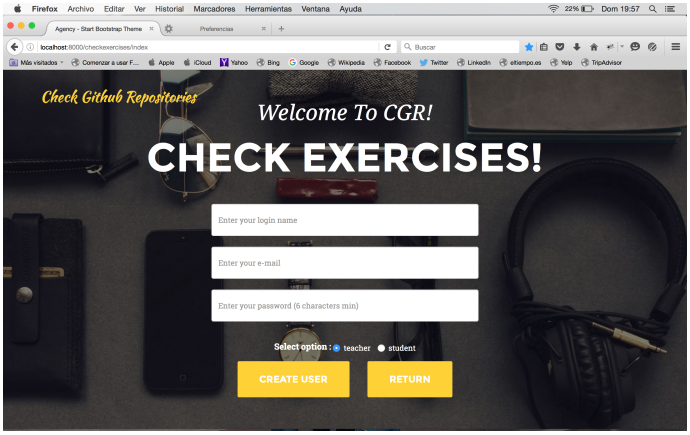

Figura 4.5: Formulario para crear un usuario.

Una vez ha rellenado todos los datos y hace click en Create User, el cliente comprueba que todos los datos son correctos y envía al servidor un POST con AJAX en el que indica:

- la acción de crear un usuario con el nombre del usuario a crear (createuser),
- el email (createmail),
- la contraseña (createpassword),
- el rol (createrol)

- **back-end:**

El servidor recibe el POST e identifica que uno de los campos es **createuser**, por lo tanto sabe que la acción que tiene que desempeñar es la de crear un usuario.

Lo primero que hace el servidor es comprobar que el nick de ese usuario no está usado previamente:

- Si está usado devolverá un mensaje al cliente de que el nick de usuario ya existe.

- Si no está usado comprueba primero si hay un usuario activo en la base de datos para desactivarlo y posteriormente crea el nuevo usuario que pasa a ser el usuario activo en ese momento. Que un usuario esté activo es importante para saber que usuario tiene que acceder a la base de datos, tanto para guardar informaciones como para extraerlas.

Después de crear el usuario, nos aparecerá el menú principal del usuario en el que podremos introducir un ejercicio a analizar

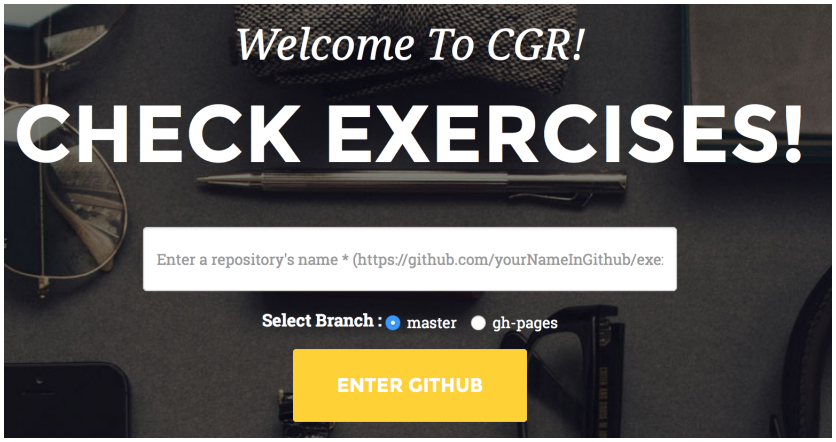

Figura 4.6: Formulario para introducir el ejercicio a analizar.

y ver tanto las librerías o ficheros que no se quieren analizar, y los ejercicios analizados previamente, que en este caso no tenemos ninguno puesto que acabamos de crear un usuario.

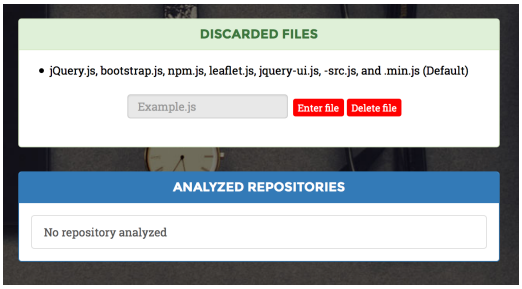

Figura 4.7: Paneles de Bootstrap que muestran los repositorios analizados y los ficheros descartados junto un formulario para añadir o eliminar ficheros de la base de datos.

Cuando tengamos ejercicios analizados, en este menú principal del usuario podremos ver un ranking con los errores más comunes y una gráfica con la evolución del usuario a medida que va entregando los ejercicios.

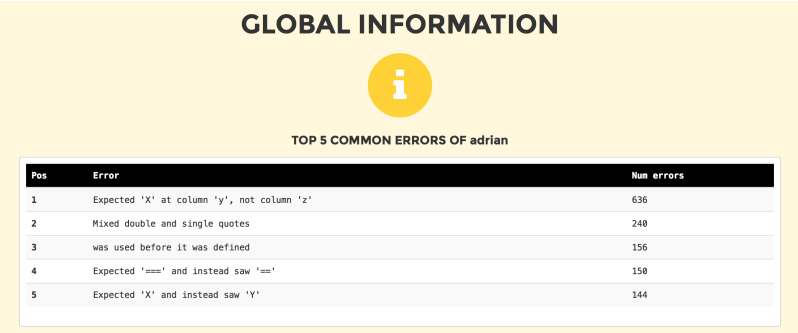

Figura 4.8: Ranking global de los errores más comunes de todos los ejercicios analizados del usuario.

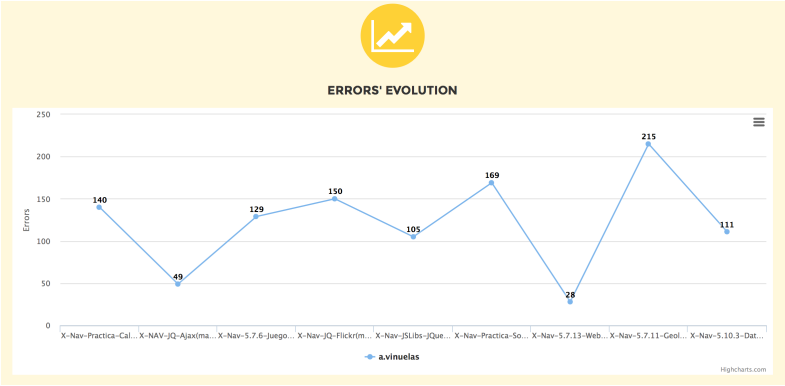

Figura 4.9: Gráfica que muestra la evolución del usuario a medida que introduce ejercicios.

4.3.2. Analizar ejercicio

Una vez tenemos creado el usuario, vamos a hacer uso de la aplicación. Para ello se introduce la URL de un ejercicio de GitHub. Dependiendo del rol no todas las URLs son válidas. Si el usuario es profesor, solo serán válidas las URLs que sean del tipo "https://github.com/CursosWeb/nombre-del-ejercicio". En el caso de que el usuario sea estudiante, solo serán válidas las URLs que sean del tipo "https://github.com/nombre-en-github/nombre-del-ejercicio".

Estudiante

El esquema del proceso de analizar un ejercicio de estudiante es el siguiente:

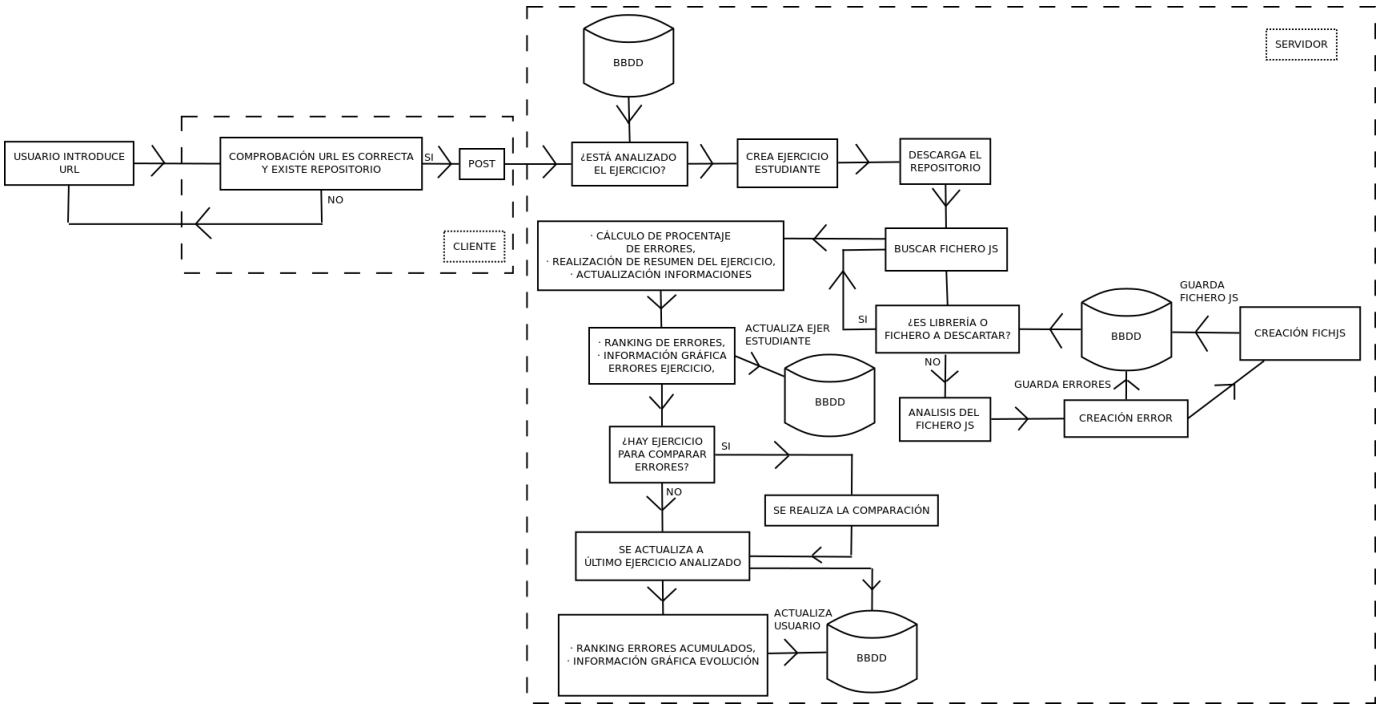

Figura 4.10: Diagrama proceso de analizar ejercicio de estudiante.

- **front-end:**

Si el usuario es estudiante, introduce la URL del ejercicio a analizar y el cliente comprueba que la URL es correcta y que el repositorio existe de verdad (puesto que la URL puede tener el formato correcto pero no existir en GitHub). Luego envía al servidor un POST con AJAX en el que indica:

- la acción de analizar un repositorio indicando la URL (urlanalizstudent),

- la rama de la que hay que descargar el repositorio (rama),
- la URL del avatar del estudiante al que pertenece el ejercicio (urlavatar)

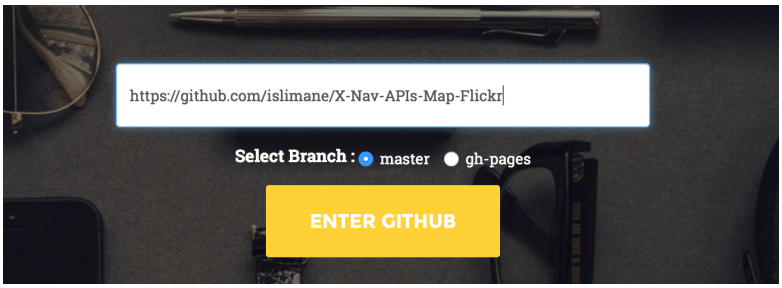

Figura 4.11: Ejemplo de cómo se introduce un ejercicio de estudiante.

- **back-end:**

El servidor recibe el POST e identifica que uno de los campos es **urlanalizstudent**, por lo tanto sabe que la acción que tiene que desempeñar es la de analizar un repositorio de un estudiante.

Lo primero que hace el servidor es coger al usuario activo y comprobar que tiene el rol de estudiante (si ha entrado por esta acción siempre se cumplirá, es simplemente por seguridad). Comprueba si el ejercicio que se va a analizar ya esta en la base de datos (es importante para ir incrementando el **numorden** explicado en la base de datos):

- si ya existe en la base de datos, mira cual es el **numorden** del último ejercicio analizado y lo incrementa en 1.
- si no existe en la base de datos, crea la variable **numorden** con valor 1.

A continuación guarda en la base de datos un nuevo ejercicio de estudiante (**StudentExercise**) con los valores iniciales por defecto, puesto que a medida que se vaya analizando se irán actualizando.

Posteriormente intenta descargar el repositorio, bien sea de la rama **master** o de la rama **gh-pages**. Si se produce algún fallo en la descarga del repositorio no se analiza nada. Si se descarga correctamente, se llama a una función que lo que hará será buscar ficheros con extensión **.js** y los analizará. Cuando encuentra un fichero **.js**, lo primero que hace es mirar a ver si es una librería o algún fichero que está en la base de datos y que no queremos analizar. Si el fichero

debe ser analizado se procede a ello aplicando primero la herramienta JSLint y después JSHint. El procedimiento es el mismo para ambas herramientas:

- se aplica la herramienta JSLint o JSHint,
- se analizan uno por uno todos los errores para ver de que tipo son,
- se crea un nuevo objeto Error (por cada error analizado) y se guarda en la base de datos,
- se devuelve el análisis que ha realizado la herramienta y el número de errores totales (en el caso del análisis de JSLint, para luego continuar por el mismo número en el análisis de JSHint, ya que no queremos diferenciar un análisis de otro en el resumen),
- se suman el número de errores JSLint y JSHint para luego hacer el resumen ,
- se hace el resumen del fichero,
- se da forma a la información con la que luego se podrá hacer la gráfica de los errores de ese fichero,
- se crea un nuevo fichero con los valores que le corresponden y se guarda en la base de datos

Al final la función que busca ficheros **.js** devuelve el número total de ficheros **.js** analizados. Hasta este momento, ya tenemos todos los ficheros analizados del repositorio que queremos analizar junto con todos los errores, todo ello guardado en la base de datos. Ahora falta completar la información para el ejercicio de estudiante (**StudentExercise**) que habíamos guardado con los valores por defecto. Para ello:

- se calcula el porcentaje de errores que ha obtenido dicho ejercicio,
- se realiza el resumen general de los errores del ejercicio,
- se actualizan las informaciones necesarias (si hay ficheros analizados, número de ficheros analizados, porcentaje de errores del ejercicio y número de errores totales del ejercicio) para el objeto (**StudentExercise**) y se guarda en la base de datos.

Al final se realiza el ranking de errores para el ejercicio y se da forma a la información con la que luego se podrá hacer la gráfica de los errores de ese ejercicio. Si hay ejercicios analizados anteriormente, se comparará el ejercicio actual con el último analizado, y se actualizará nuestro ejercicio actual al último ejercicio analizado. Por último, se realiza el ranking global de errores acumulados por el usuario durante todos sus análisis realizados a lo largo del tiempo, así como se da forma a la información con la que luego se hará la gráfica de la evolución en el tiempo. Se guarda nuestro objeto **StudentExercise** con los últimos cambios realizados y se acaba el proceso de análisis de un ejercicio para el rol de estudiante.

Profesor

El esquema del proceso de analizar un ejercicio de estudiante es el siguiente:

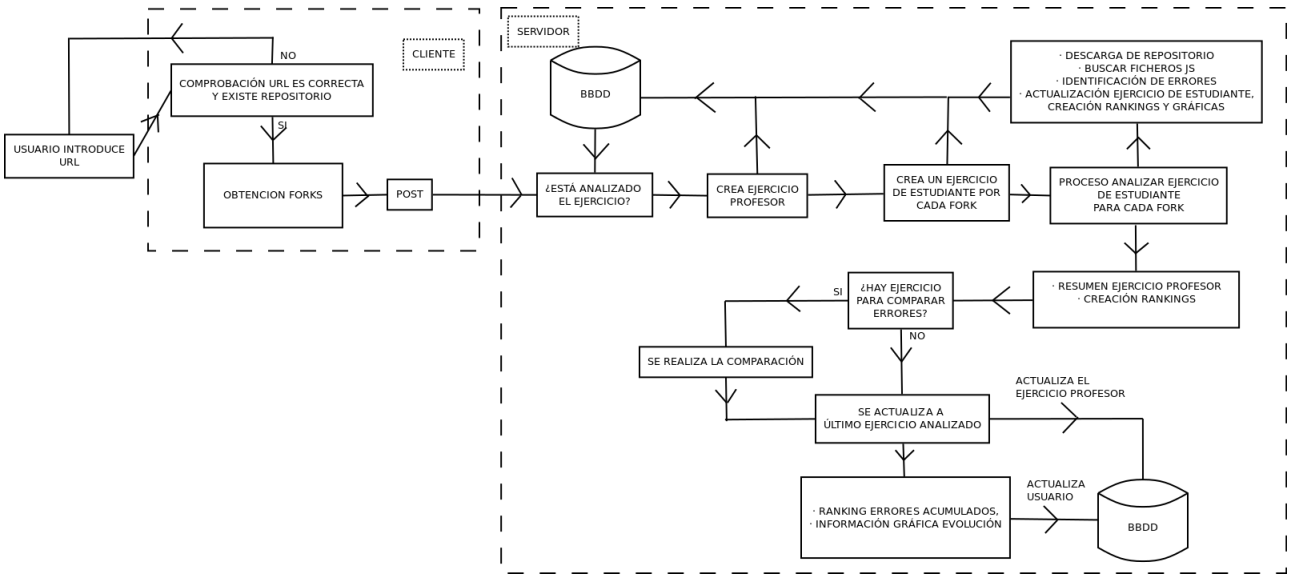

Figura 4.12: Diagrama proceso de analizar ejercicio de profesor.

- **front-end:**

Si el usuario es profesor, introduce la URL del ejercicio a analizar y el cliente comprueba que la URL es correcta y que el repositorio existe de verdad (puesto que la URL puede tener el formato correcto pero no existir en GitHub). Después busca todos los **fork** (cada fork es un ejercicio de estudiante en GitHub. Es una copia del repositorio del profesor que se genera en la cuenta del alumno, y es donde el alumno entrega los ficheros de su ejercicio). Luego envía al servidor un POST con AJAX en el que indica:

- la acción de analizar un repositorio de profesor indicando la URL (urlanalizall),
- la lista de todos los fork que tiene que analizar (listForks),
- la rama de la que hay que descargar el repositorio (rama)

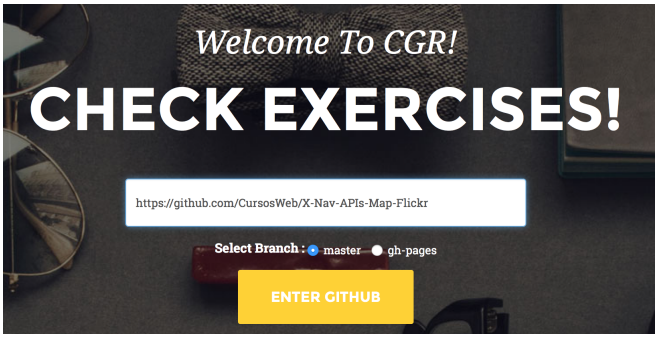

Figura 4.13: Ejemplo de cómo se introduce un ejercicio de profesor.

- **back-end:**

El procedimiento que realiza el servidor es muy parecido al procedimiento que realiza cuando analiza un repositorio de estudiante, puesto que ahora lo que hará será analizar todos los repositorios de estudiantes que haya en la lista de forks.

El servidor recibe el POST e identifica que uno de los campos es **urlanalizall**, por lo tanto sabe que la acción que tiene que desempeñar es la de analizar un repositorio de un profesor.

Lo primero que hace el servidor es coger al usuario activo y comprobar que tiene el rol de profesor (si ha entrado por esta acción siempre se cumplirá, es simplemente por seguridad). Comprueba si el ejercicio que se va a analizar ya esta en la base de datos (es importante para ir incrementando el **numorden** explicado en la base de datos):

- si ya existe en la base de datos, mira cual es el **numorden** del último ejercicio analizado y lo incrementa en 1.
- si no existe en la base de datos, crea la variable **numorden** con valor 1.

A continuación guarda en la base de datos un nuevo ejercicio de profesor (**TeacherExercise**) con los valores iniciales por defecto, puesto que a medida que se vaya analizando se irán actualizando. Después coge la lista de forks y crea un objeto de ejercicio de estudiante (**StudentExercise**)

por cada fork que encuentra (ya que lo que hará más adelante será analizar uno a uno todos los forks).

Una vez tenemos guardados todos los ejercicios de estudiantes que hay que analizar, los analiza uno a uno siguiendo el mismo procedimiento que ha realizado para analizar un repositorio dentro del ejercicio de estudiante explicado anteriormente:

- se descarga el repositorio del estudiante a analizar,
- busca ficheros **.js**, cuando encuentra uno lo analiza aplicando las herramientas JSLint y JSHint, identificando cada **error** y guardándolo en la base de datos,
- guarda toda la información relacionada con cada fichero **.js**,
- guarda toda la información relacionada con cada ejercicio de estudiante (**StudentExercise**) que analiza, es decir, toda la información analizada de cada fork que ha encontrado (ficheros, errores, y resúmenes)

Una vez analizados todos los repositorios de estudiantes, el servidor realiza el resumen de todos los errores que han tenido todos los estudiantes y todos los rankings que se corresponden al ejercicio de profesor (**TeacherExercise**), es decir, ranking de estudiantes según porcentaje de errores y según ficheros analizados, y el ranking de errores más comunes. Si hay ejercicios analizados anteriormente, se comparará el ejercicio actual con el último analizado, y se actualizará nuestro ejercicio actual al último ejercicio analizado. Por último, se realiza el ranking global de errores acumulados por el usuario durante todos sus análisis realizados a lo largo del tiempo, así como se da forma a la información con la que luego se hará la gráfica de la evolución en el tiempo. Se guarda nuestro objeto **TeacherExercise** con los últimos cambios realizados y se acaba el proceso de análisis de un ejercicio para el rol de profesor.

Como se puede observar, el procedimiento es prácticamente el mismo que al analizar un repositorio de estudiante, lo único que para el análisis de un ejercicio de profesor se analizan todos los repositorios de estudiante que hay dentro del ejercicio del profesor y se realizan los resúmenes y rankings que para el estudiante no se realizan.

4.3.3. Creación de gráficas

Una parte muy importante de este proyecto es interpretar los datos que nos devuelve el análisis de un ejercicio. Para facilitar su interpretación, se han implementado una serie de gráficas para facilitar la lectura de los datos. Hay dos tipos de gráficas, unas son de tipo **barras** y otras de tipo **líneas**. Como bien se explicó en la sección **Estado del arte**, la herramienta utilizada para la creación de las gráficas es **Highcharts**. Las gráficas las tiene que implementar el cliente. Por esta razón se tenía que guardar en la base de datos la información relacionada con las gráficas para cada ejercicio de profesor, ejercicio de estudiante, y fichero, así como la información de la evolución a lo largo del tiempo del usuario. Para crear las gráficas, estas informaciones se incluyen en el DOM, y mediante llamadas a funciones es el cliente quien implementa las gráficas mediante el uso de Highcharts.

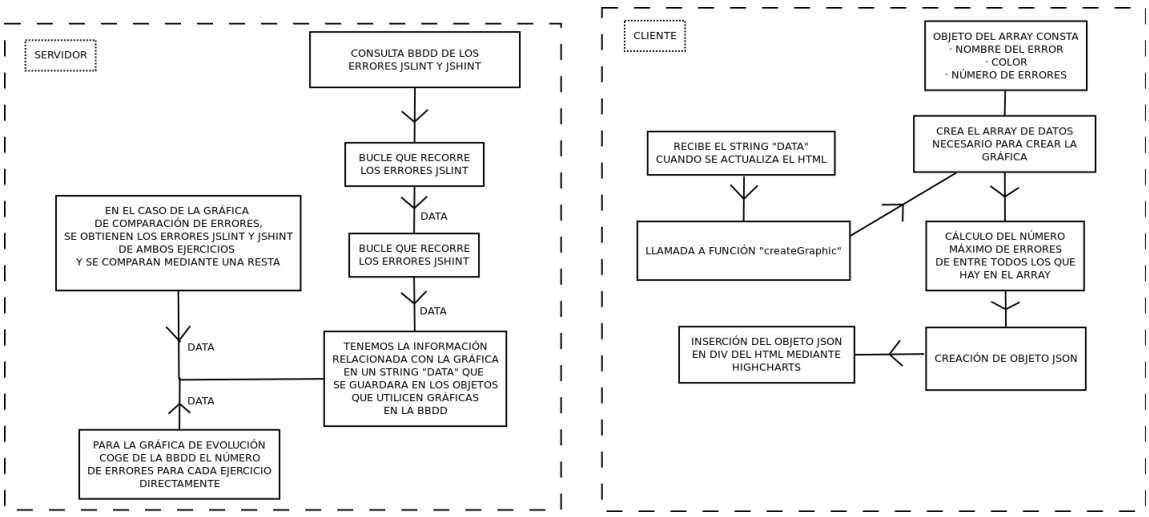

Figura 4.14: Diagrama proceso de creación de gráficas.

El resultado de las gráficas es el siguiente:

- Gráfica de resumen de errores totales del ejercicio:

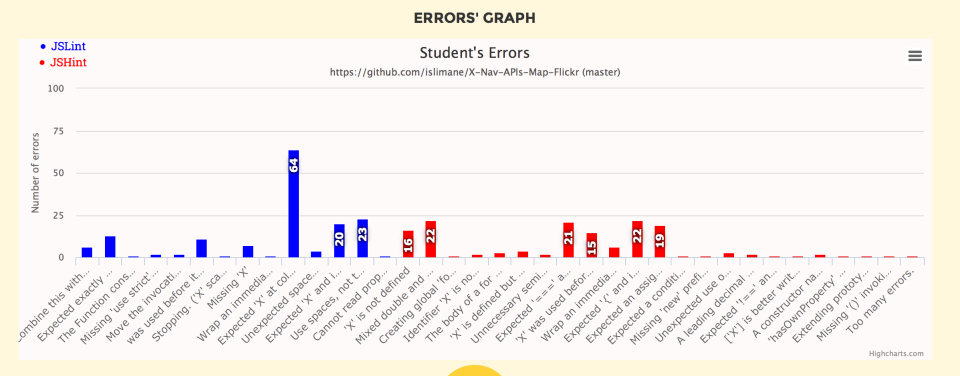

Figura 4.15: Gráfica de resumen de errores totales del ejercicio.

- Gráfica de resumen de errores de fichero:

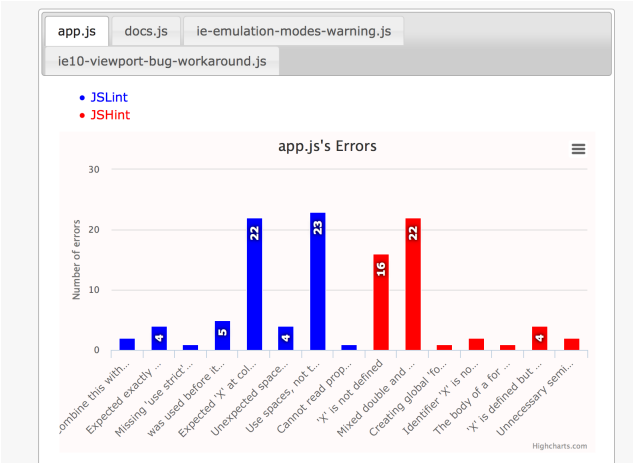

Figura 4.16: Gráfica de resumen de errores de fichero.

- Gráfica de comparación de errores con ejercicio anterior:

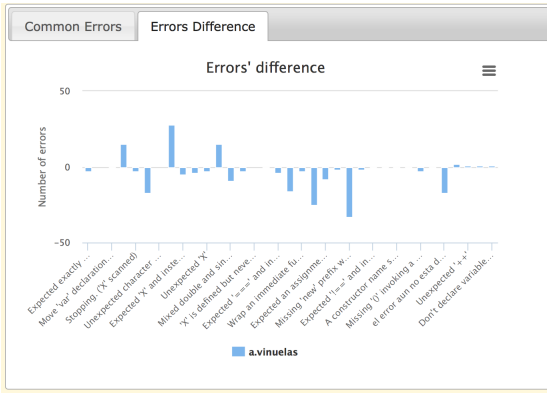

Figura 4.17: Gráfica de comparación de errores con ejercicio anterior.

- Gráfica de la evolución a lo largo del tiempo:

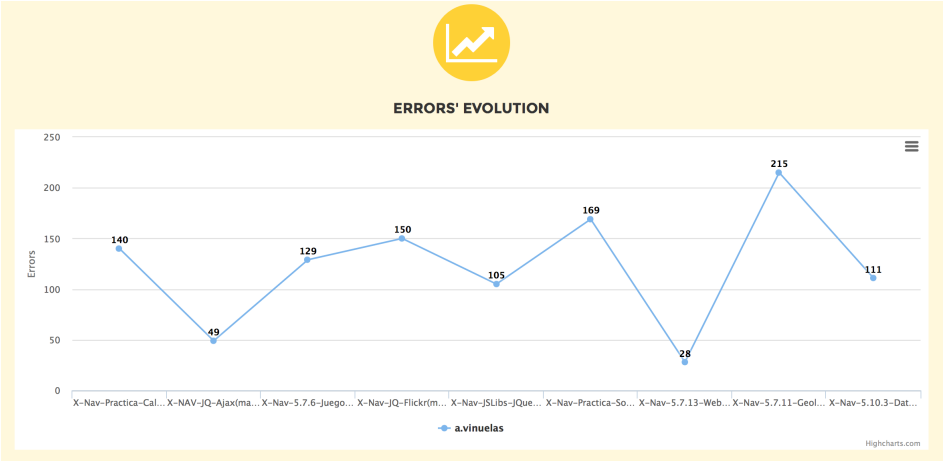

Figura 4.18: Gráfica de la evolución a lo largo del tiempo.

4.3.4. Crear y eliminar librerías/ficheros

Una de las opciones que permite la aplicación es evitar el análisis de ciertos ficheros que no son de interés, puesto que no los han desarrollado los alumnos y que, por tanto, no aportan ninguna información necesaria al análisis.

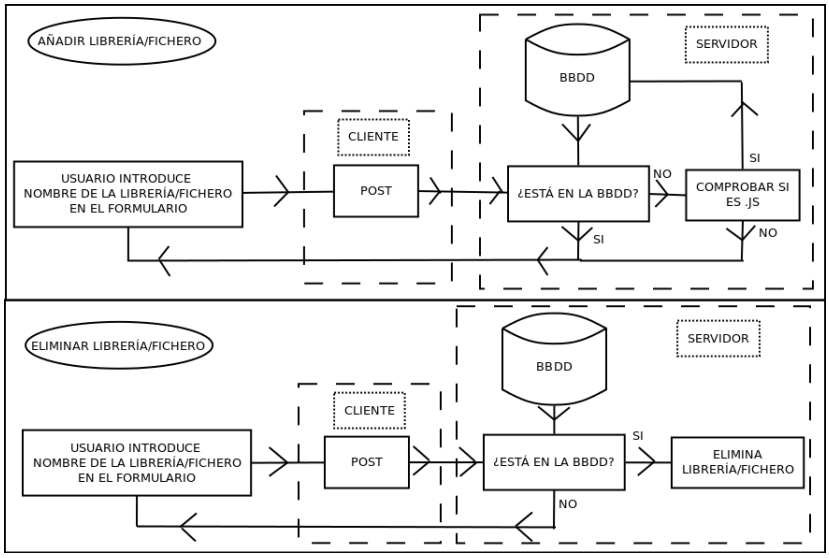

Figura 4.19: Diagrama proceso de creación/eliminación librerías/ficheros.

Por defecto, la aplicación ya incluye unas librerías, pero da la opción de poder añadir a la lista nuevos ficheros **.js**.

Añadir librería/fichero

- **front-end:**

El usuario incluye el nombre del fichero que quiere añadir a la lista de librerías que no se tendrán en cuenta a la hora de analizar un repositorio y hará click en "Enter file".

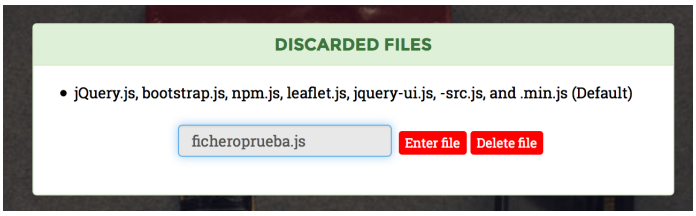

Figura 4.20: Ejemplo de introducción de fichero que será descartado en el análisis.

El cliente envía al servidor un POST con AJAX en el que indica:

- el nombre de la librería (o fichero) que tiene que añadir a la base de datos (savelibrary)

- **back-end:**

El servidor recibe el POST e identifica que uno de los campos es **savelibrary**, por lo tanto sabe que la acción que tiene que desempeñar es la de añadir una librería (o fichero) a la base de datos.

Para ello primero comprueba si esa librería (o fichero) ya está en la base de datos para ese usuario. Si está en la base de datos, no la vuelve a guardar y avisará al usuario con un *warning*.

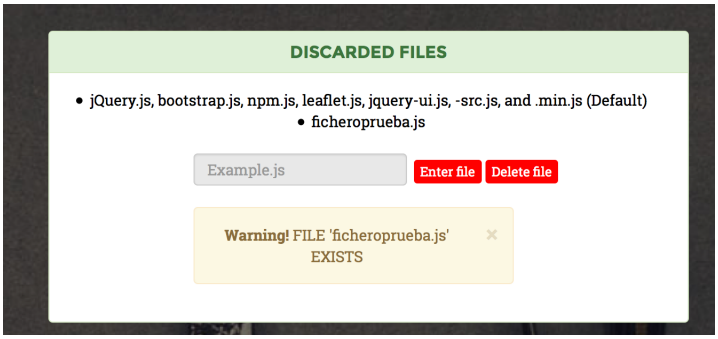

Figura 4.21: Warning de que el fichero ya existe en la base de datos.

Si no está en la base de datos, comprobará que la extensión de la librería (o fichero) introducido es **.js**, y si es así, lo guardará en la base de datos e informará al usuario de que todo ha ido bien.

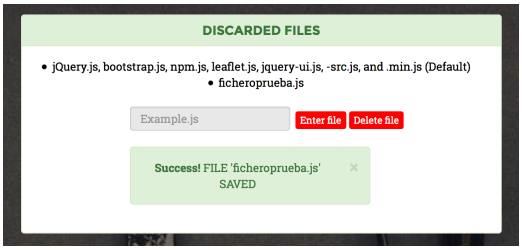

Figura 4.22: Aviso de que el fichero se ha guardado en la base de datos.

Si por otro lado la extensión de la librería (o fichero) no es **.js**, informará al usuario con un warning y no la guardará en la base de datos.

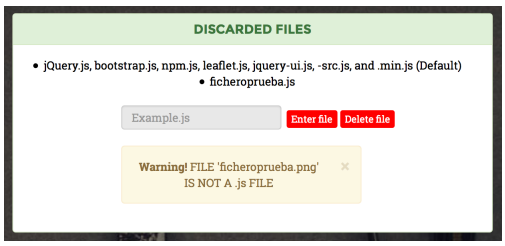

Figura 4.23: Warning de que el fichero no es de tipo .js .

Eliminar librería/fichero

- **front-end:**

El usuario incluye el nombre del fichero que quiere eliminar de la lista de librerías que no se tendrán en cuenta a la hora de analizar un repositorio y hará click en “Delete file”.

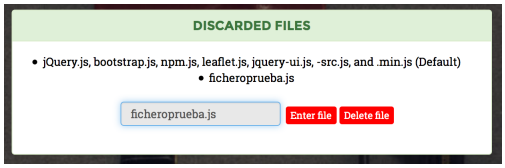

Figura 4.24: Ejemplo de introducción de fichero que será eliminado de la base de datos.

El cliente envía al servidor un POST con AJAX en el que indica:

- el nombre de la librería (o fichero) que tiene que eliminar de la base de datos (deletelibrary)

- **back-end:**

El servidor recibe el POST e identifica que uno de los campos es **deletelibrary**, por lo tanto sabe que la acción que tiene que desempeñar es la de eliminar una librería (o fichero) de la base de datos.

Lo primero que hace el servidor es comprobar que la librería (o fichero) está en la base de datos para ese usuario, y si es así, la elimina e informa al usuario de que todo ha ido bien.

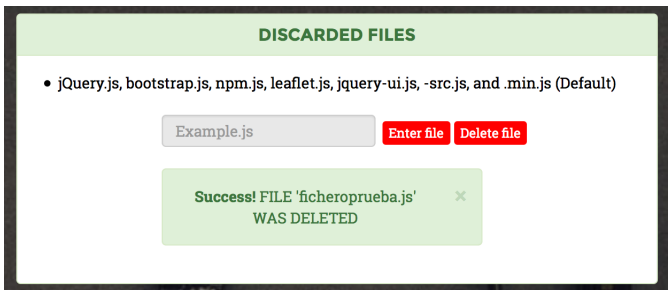

Figura 4.25: Aviso de que el fichero se ha eliminado de la base de datos.

Si la librería no está en la base de datos para ese usuario, el servidor informa al usuario de que la librería que desea eliminar no existe.

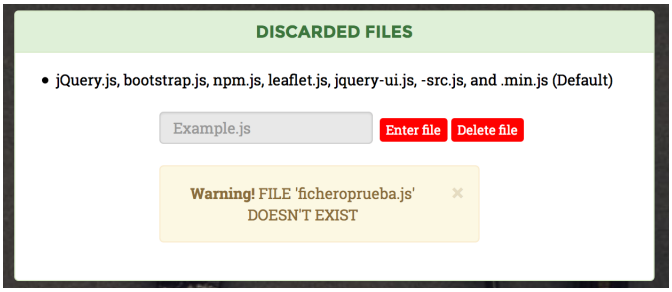

Figura 4.26: Warning de que el fichero no existe en la base de datos.

4.3.5. Acceder a la aplicación

Una vez hemos creado el usuario, podremos acceder a la aplicación y salir de ella en dos sencillos pasos. El primero es hacer un LOGIN y el segundo es hacer un LOGOUT.

Login

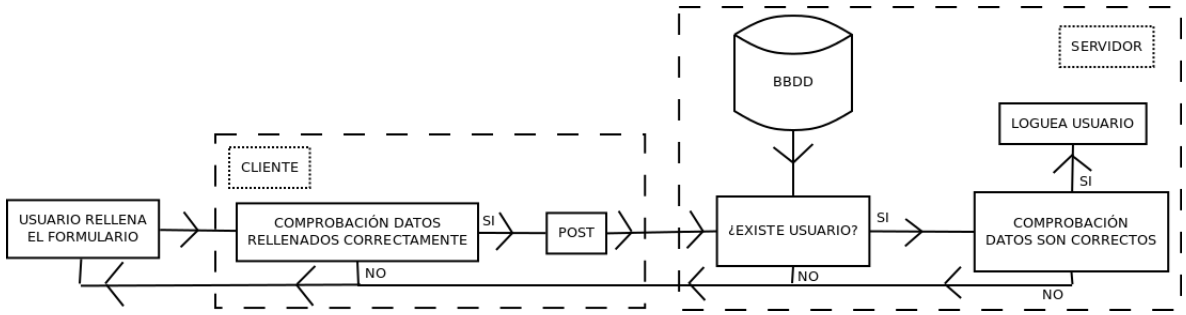

Figura 4.27: Diagrama proceso de loguear usuario.

- **front-end:**

Para acceder a la aplicación, lo que el usuario tiene que hacer es loguearse. Para ello hay que rellenar el formulario que aparece en la página principal cuando comenzamos a usar la aplicación con los datos correctos.

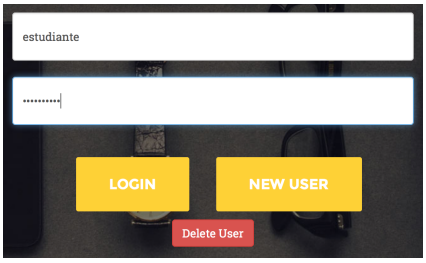

Figura 4.28: Ejemplo de introducción de datos para loguear un usuario.

Si falta algún dato por rellenar, es el propio cliente quien informa al usuario para que compruebe los datos.

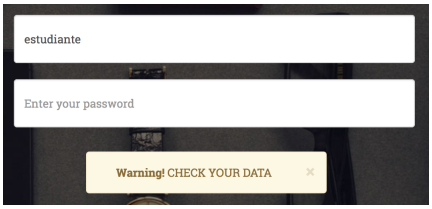

Figura 4.29: Warning de que faltan datos por introducir para poder loguear un usuario.

Si todo es correcto, el cliente envía al servidor un POST con AJAX en el que indica:

- el nombre del usuario que quiere loguearse (loginuser),
- la contraseña del usuario (loginpassword)
- **back-end:**

El servidor recibe el POST e identifica que uno de los campos es **loginuser**, por lo tanto sabe que la acción que tiene que desempeñar es la de loguear a un usuario.

Primero el servidor comprueba que el usuario existe. De no ser así, informa al usuario.

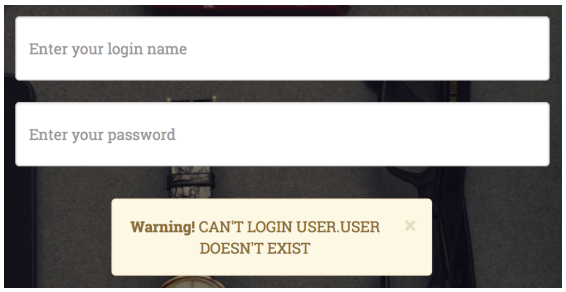

Figura 4.30: Warning de que el usuario a loguear no existe.

Si el usuario existe, procede a loguearlo. Si todo es correcto, tanto nombre de usuario como contraseña, el usuario se logueará y podremos comprobar que ya estamos dentro de la aplicación, ya que aparecerá nuestro nombre de usuario para informarnos de quien es el usuario que está utilizando la aplicación.

Figura 4.31: El nombre del usuario logueado aparece en la página.

Si el usuario existe pero la contraseña no es la correcta, este no será logueado y el servidor le informará de que la contraseña es errónea.

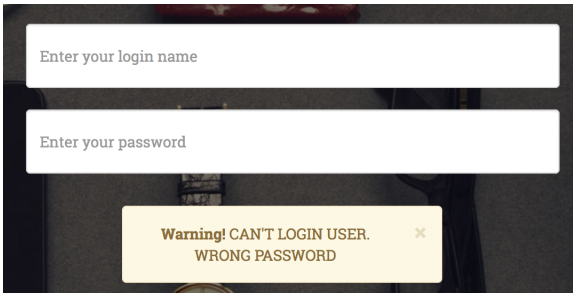

Figura 4.32: Warning de que la contraseña introducida es errónea.

Logout

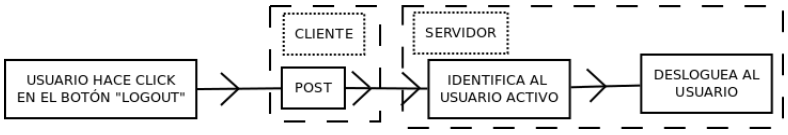

Figura 4.33: Diagrama proceso de desloguear usuario.

- **front-end:**

Una vez queremos salir de la aplicación solo hace falta que el usuario haga click en el botón "Logout".

Figura 4.34: Botón Logout.

El cliente envía al servidor un POST con AJAX en el que indica:

- que hay que hacer un logout del usuario (logoutuser)

- **back-end:**

El servidor recibe el POST e identifica que uno de los campos es **logoutuser**, por lo tanto sabe que la acción que tiene que desempeñar es la de desloguear a un usuario.

El servidor coge al usuario activo (que es el que está logueado) y lo desloguea poniéndolo a inactivo.

4.3.6. Eliminar usuario

Otra de las opciones que permite la aplicación es la de borrar la cuenta de un usuario, y con ello toda la información relacionada con el usuario.

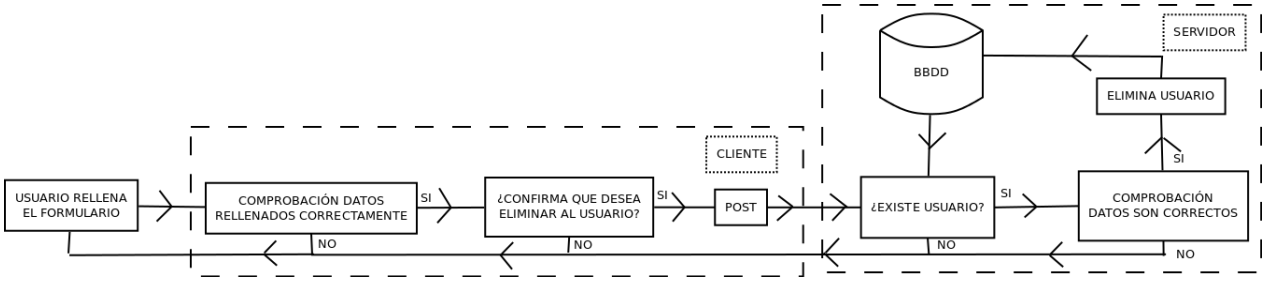

Figura 4.35: Diagrama proceso de eliminar usuario.

- **front-end:**

Para ello el usuario solo tiene que pulsar en el botón “Delete User”

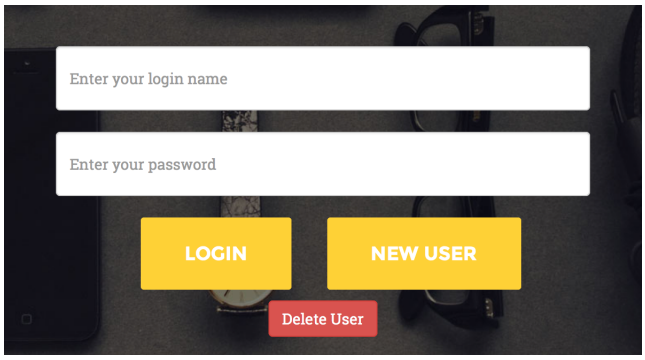

Figura 4.36: Formulario en el que aparece el botón “Delete User”.

Y rellenar correctamente el formulario.

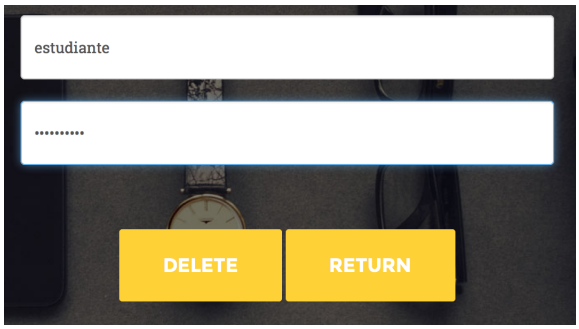

Figura 4.37: Ejemplo de introducción de datos para borrar un usuario.

Si falta algún dato, el cliente avisará al usuario.

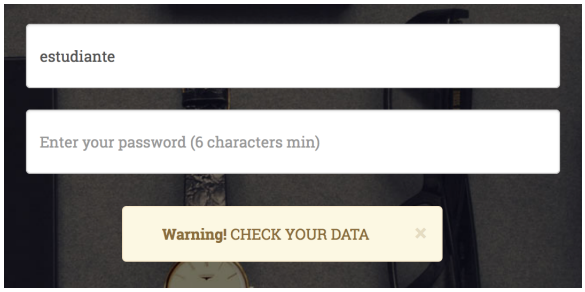

Figura 4.38: Warning de que faltan datos por introducir para poder borrar al usuario.

Si todo es correcto, la aplicación preguntará al usuario que si está seguro de querer borrar al usuario (ya que es un paso importante porque se perderán todos sus datos).

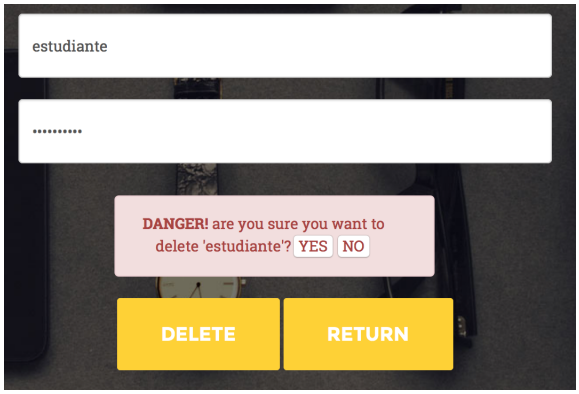

Figura 4.39: Aviso de que se va a proceder a eliminar al usuario introducido.

De querer continuar borrando el usuario, el cliente envía al servidor un POST con AJAX en el que indica:

- el nombre de usuario a borrar (deluser),
- la contraseña del usuario a borrar (delpassword), porque si no cualquiera podría borrar los usuarios de los demás, y hay que estar seguro de que el usuario que quiere eliminarlo tiene permiso.
- **back-end:**

El servidor recibe el POST e identifica que uno de los campos es **deluser**, por lo tanto sabe que la acción que tiene que desempeñar es la de eliminar a un usuario de la base de datos junto a toda la información que hay almacenada de él.

El servidor comprueba en primer lugar que el usuario a eliminar existe. De no ser así avisará al usuario.

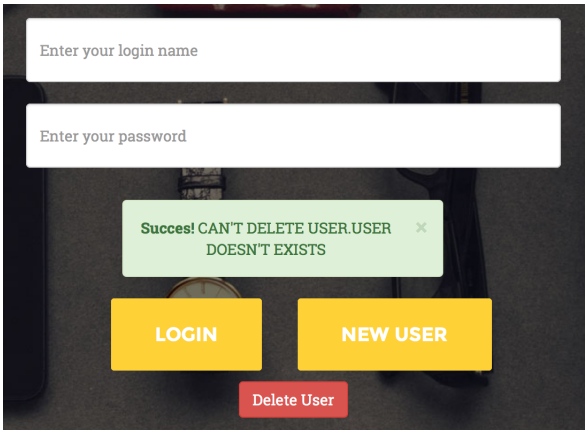

Figura 4.40: Aviso de que no se puede borrar al usuario porque no existe.

Si el usuario existe, lo primero que hace es comprobar que la contraseña es correcta, puesto que tenemos que asegurarnos de que el usuario que quiere borrar una cuenta tiene permiso para ello. Si la contraseña es incorrecta, el servidor avisará al usuario de ello.

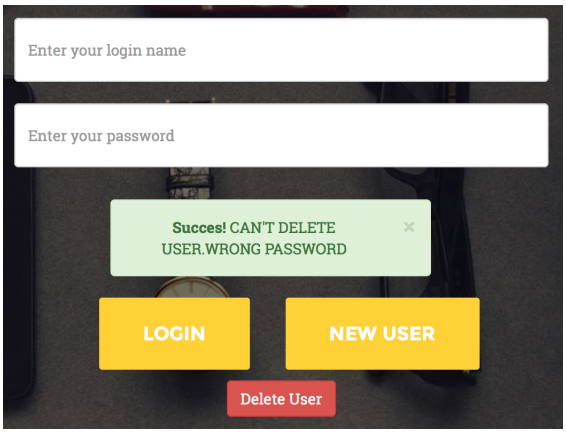

Figura 4.41: Aviso de que la contraseña introducida para borrar el usuario es errónea.

Si la contraseña es correcta, el servidor borra todos los datos relacionados con el usuario y le informará de que todo ha ido bien.

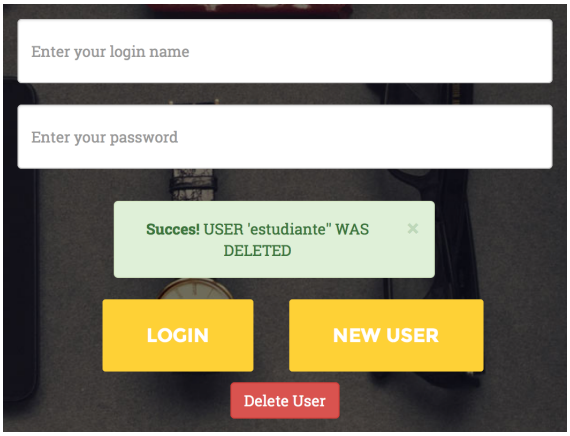

Figura 4.42: Aviso de que el usuario ha sido eliminado.

4.3.7. Información errores

Una de las funcionalidades de la aplicación, además de analizar los ejercicios y mostrar su resultado, es dar la opción al usuario de que pueda ver en que parte de su código está el error que el análisis ha encontrado, así como de que pueda ver una explicación de por qué sale ese error y como solucionarlo.

- **front-end:**

Cuando antes explicaba en el análisis de un ejercicio que guardábamos el resumen de los errores de un fichero era porque lo íbamos a necesitar ahora para poder ver en que parte del código está localizado el error. El resumen (en el que se agrupan todos los errores según el tipo de error) de un fichero es el siguiente:

Figura 4.43: Resumen de los errores de un fichero.

Si se pulsa sobre uno de los errores en ese resumen, la aplicación nos abrirá una nueva pestaña que nos llevará a la página de GitHub donde se encuentra el fichero, y nos indicará la línea en la que se localiza el error. Si pulsamos en el error 51 del ejemplo, nos mostrará la línea en el fichero:

Figura 4.44: Ejemplo de selección de un error para ver la línea exacta en el fichero.

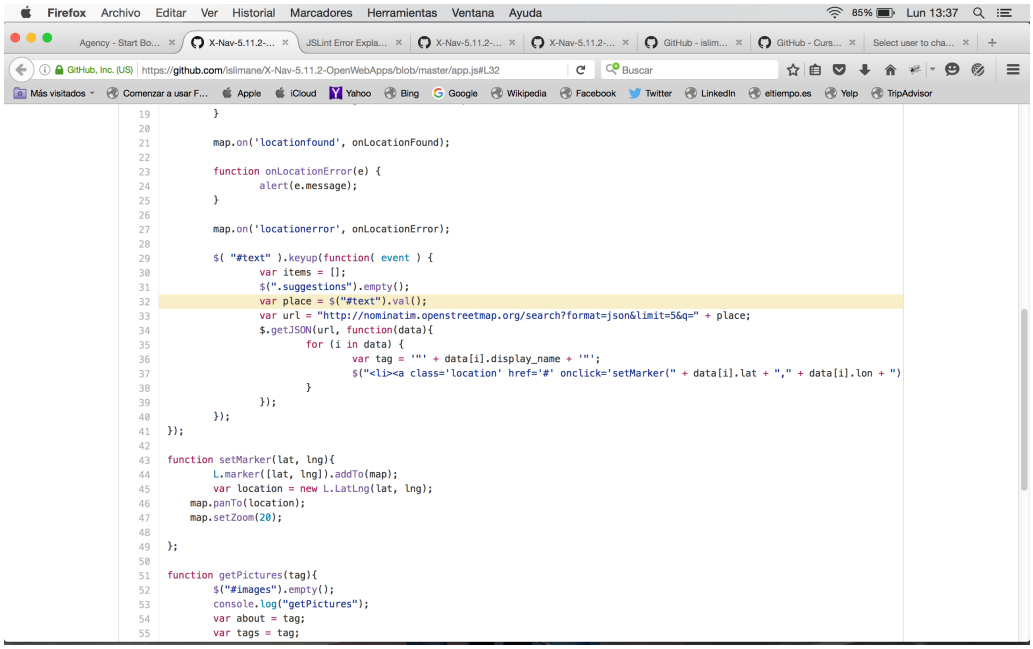

Figura 4.45: Página que se nos muestra al hacer click en el error para verlo en el fichero desarrollado por el estudiante.

Además se puede ver la información sobre el error en concreto pulsando en la solución (Sol):

Figura 4.46: Ejemplo de la selección de una solución.

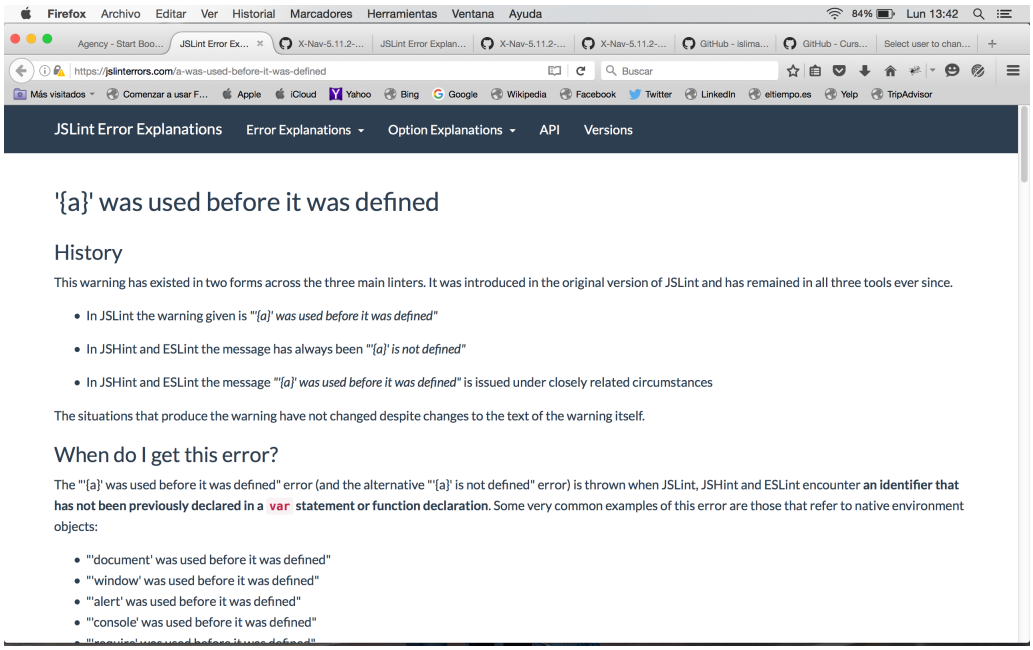

Figura 4.47: Página que se nos muestra al hacer click en la solución para ver la explicación.

- **back-end:**

Es el servidor quien cuando crea los errores almacena en la base de datos para cada error la URL en la que se nos mostrará donde está el error en el fichero en GitHub. Por otra parte almacena también la solución a ese error. Posteriormente cuando crea el resumen del fichero, incluye unos **href** en este, que al pulsarlos son los que nos redirigen tanto a la solución como a la visualización del error. La solución de los errores está definida en una función que tiene el servidor en la que están definidos todos los tipo de error así como sus soluciones.

Capítulo 5

Resultados

Una vez terminado el proyecto, tenemos ante nosotros una aplicación que permite analizar el código JavaScript desarrollado por estudiantes de la universidad. Analiza los errores, los agrupa por tipos y muestra el resultado empleando gráficas y rankings que ayudan al usuario de la aplicación a tener una visión rápida del análisis del ejercicio en tan solo un minuto.

Puede ayudar a los alumnos a mejorar su código al mismo tiempo que lo están escribiendo, puesto que un mismo ejercicio se puede pasar por la aplicación tantas veces como se quiera y compara un ejercicio con el anterior, por tanto, los alumnos pueden corregir los errores que les muestre la aplicación en un primer análisis, y luego volver a analizar de nuevo el ejercicio para comprobar que la segunda versión entregada tiene menos errores que la primera vez que se analizó el ejercicio, y así sucesivamente.

Por otro lado, no se necesita la figura del profesor en un primer momento para que te de una explicación de por qué salen los errores ni tener que ir a buscar en internet una explicación, ya que se ha conseguido que la aplicación sea didáctica en este sentido y lo hace ella por sí misma. Muestra el motivo por el que se identifican los errores así como su solución (sacada de internet), para que el alumno no tenga que esperar a ver a su profesor para preguntarle ni pierda tiempo en buscar la explicación por su cuenta.

Por supuesto, no pretende sustituir la figura del profesor pero sí ayudar en su labor de enseñar a los alumnos.

Simulación Profesor

Se ha simulado que un profesor ha utilizado la aplicación (una vez a finalizado el cuatrimestre). En total se han analizado trece ejercicios.

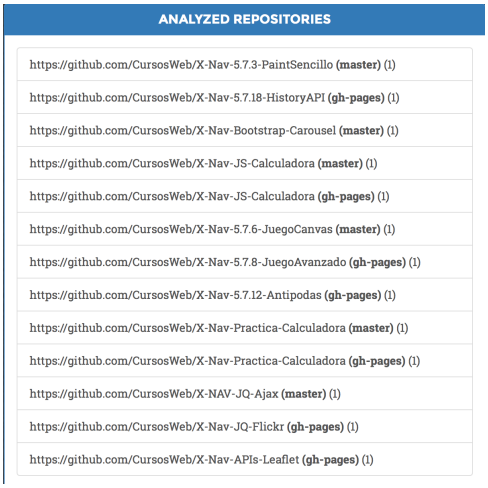

Figura 5.1: Lista de ejercicios analizados durante el curso.

Por un lado podemos ver que la evolución a lo largo del curso es la siguiente:

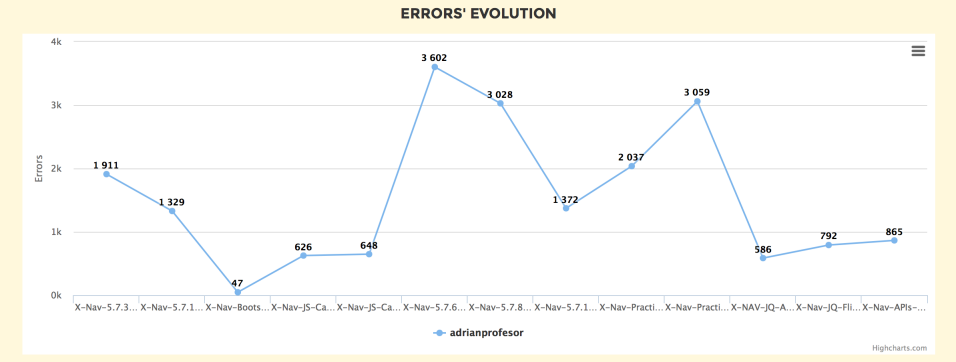

Figura 5.2: Evolución a lo largo del curso.

Como resultados podemos observar una serie de subidas y bajadas en lo referido al número de errores a lo largo del tiempo. Los resultados que podemos sacar de esta gráfica son:

- Partimos de que al principio se parte de un número de 1911 errores. El primer ejercicio lo entregaron un total de 30 alumnos. El error más común es “Identifier “X” is not in camel case” (436 errores).

- En el segundo ejercicio vemos que el número de errores ha disminuido considerablemente en 582 errores.

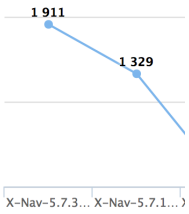

Figura 5.3: Ejercicios 1 y 2 del curso.

Esto se debe a que 8 alumnos no entregaron el ejercicio. La aplicación nos informa de ello. Por tanto, es lógico que el número de errores haya disminuido. El error más común en este segundo ejercicio es “Expected “X” at column “y”, not column “z” ” mientras que el error “Identifier “X” is not in camel case” ha disminuido de forma notable (32 errores).

Undelivered	Student
9	alexeal90
10	dpayo
11	kthristov
12	nachomanriqueperez
13	jesmoreno
14	SGjorge
15	arubiopa
16	islimane

Figura 5.4: Alumnos que no han entregado el segundo ejercicio.

- El tercer ejercicio puede extrañarnos puesto que el número de errores es de 47. Esto se debe a que hay ejercicios que se realizan en clase. Este es un ejemplo de ello. Solo 5 alumnos lo entregaron, lo cual da una pista al profesor de que alumnos van a clase (aunque no es una herramienta que mida la asistencia) y que alumnos están participando activamente durante las clases.

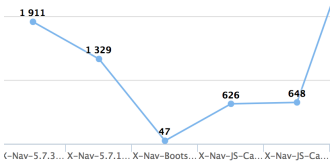

Figura 5.5: Número de errores ejercicio realizado en clase.

- Los ejercicios cuarto y quinto son el mismo ejercicio de entrega, pero uno en la rama **master** y otro en la rama **gh-pages**. Como se observa la diferencia de errores es mínima. Esto se debe a que el cuarto ejercicio lo entregaron 11 alumnos y en el quinto hubo 2 de los 11 que no lo entregaron (no lo pasaron a la rama gh-pages, se los considera no entregado por ese motivo). Sigue predominando el error "Expected "X" at column "y", not column "z" (94 errores), y por otro lado dos errores que tienen mucho que ver entre sí: ""X" is not defined" y "was used before it was defined".

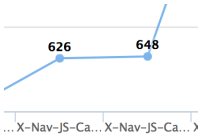

Figura 5.6: Mismo ejercicio, ramas diferentes.

Este mismo caso se cumple en los ejercicios noveno y décimo analizados.

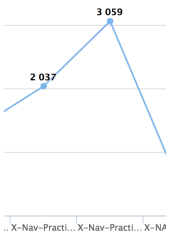

Figura 5.7: Mismo ejercicio, ramas diferentes. Es una práctica importante del curso.

En este caso es al revés. En el noveno, que es la rama **master**, lo entregaron 17 alumnos y en el décimo, que es la rama **gh-pages**, lo entregaron 24. Esto se debe a que hay ejercicios que se piden que sean entregados directamente en la rama gh-pages y por eso hay alumnos que directamente no entregan nada en la rama master y si en la rama gh-pages. Como se

puede ver la diferencia de errores entre una rama y otra es de 1022 errores. Por tanto, no solo vale con analizar una rama en muchos casos, si no que analizar las dos nos puede decir con más exactitud que alumnos entregaron un ejercicio y que alumnos no. Además, estos dos ejercicios (que en realidad son uno pero separado en dos ramas) son una práctica importante del curso (y no un ejercicio de entrega normal), con lo cual el número de errores vuelve a subir porque lo han entregado la mayoría de los alumnos, y los ficheros suelen tener mucho más código.

- En el ejercicio sexto predomina el error “Use spaces, not tabs” que está relacionado con el error “Expected “X” at column “y”, not column “z” que sigue predominando (nos vamos dando cuenta de que es muy frecuente entre las diferentes entregas).

- En los siguientes análisis siguen predominando los errores “Use spaces,not tabs”, “Expected “X” at column “y”, not column “z”, ““X” is not defined” y “was used before it was defined”.

- Y para terminar la simulación podemos observar el top 5 de los errores más comunes a lo largo del curso. Estos errores pueden dar una idea de que puntos debe comentar el profesor en clase a sus alumnos. En nuestro caso, vemos que los errores más comunes son (y que habiendo visto la evolución podemos intuir ya cuales son) :

TOP 5 COMMON STUDENTS' ERRORS OF adrianprofesor

Pos	Error	Num errors
1	Expected 'X' at column 'y', not column 'z'	3674
2	Use spaces, not tabs	2517
3	was used before it was defined	1960
4	'X' is not defined	1803
5	Expected exactly one space between 'X' and 'Y'	1203

Figura 5.8: Errores más comunes entre los alumnos a lo largo del curso.

- Que se espera un elemento ’X’ en una posición ’Y’ cuando en realidad esta en la posición ’Z’. Este error tiene mucho que ver con el segundo error más común, ya que en muchos casos el uso de tabulaciones hace que el código quede peor estructurado aunque pueda parecer lo contrario.

- Uso de espacios en lugar de tabulaciones. Como bien se ha explicado antes, puede parecer que el código queda mejor estructurado si se usan tabulaciones, pero cuando tienes que usar 7 u 8 tabulaciones nos quedamos sin pantalla y el código queda muy separado. Si se usan espacios, el código queda más ordenado.
- Los errores más comunes 3 y 4 están totalmente relacionados. Bien se sabe que en Javascript no hace falta definir variables para usarlas pero aunque no salgan fallos a la hora de compilar y ejecutar, no es una buena metodología. Si se definen las variables, el código se entenderá mucho más rápido. Los alumnos deben coger una buena metodología a la hora de aprender a programar.

El haber hecho uso de la aplicación ha permitido al profesor por un lado darse cuenta de que alumnos han entregado los ejercicios, que alumnos no han entregado los ejercicios, que estudiantes han hecho los ejercicios que se realizan en clase (y por tanto han asistido a las clases), ahora sabe cuales son los errores más comunes entre sus alumnos con lo que sabe cuales son los aspectos en los que tiene que insistirles más, además de intentar explicárselo como también a los alumnos que pueda tener en un futuro, puesto que le da una pista de en que puntos suelen estar más flojos los estudiantes desde el principio y por tanto tiene que intentar insistir a sus futuros alumnos desde un primer momento en tomar una buena metodología de programación.

Simulación Alumno

Por otro lado se ha realizado una simulación de como un alumno utiliza la aplicación para intentar mejorar el desarrollo de su código. El alumno ha pasado el ejercicio 5 veces por la aplicación hasta conseguir dejar un código mucho más legible y mejor desarrollado y estructurado.

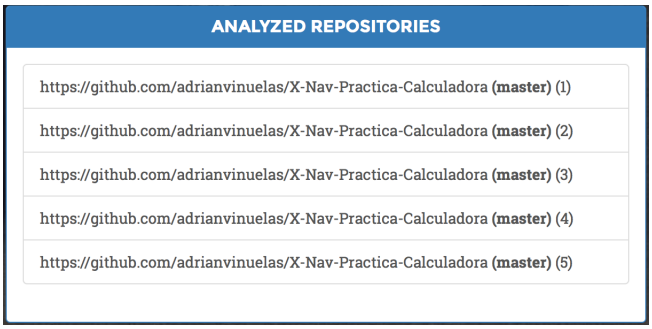

Figura 5.9: Panel que muestra los ejercicios analizados por parte del alumno.

La evolución del progreso del ejercicio ha sido la siguiente.

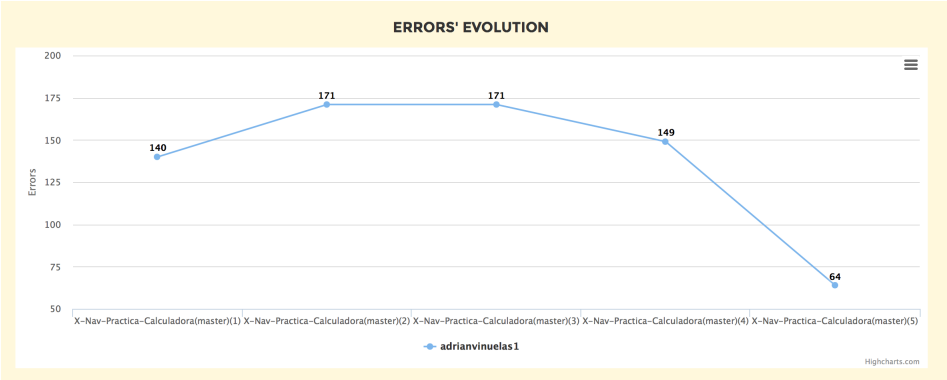

Figura 5.10: Progreso del ejercicio a medida que se ha ido analizando varias veces.

El alumno parte de un primer análisis del ejercicio en el que obtiene 140 errores.

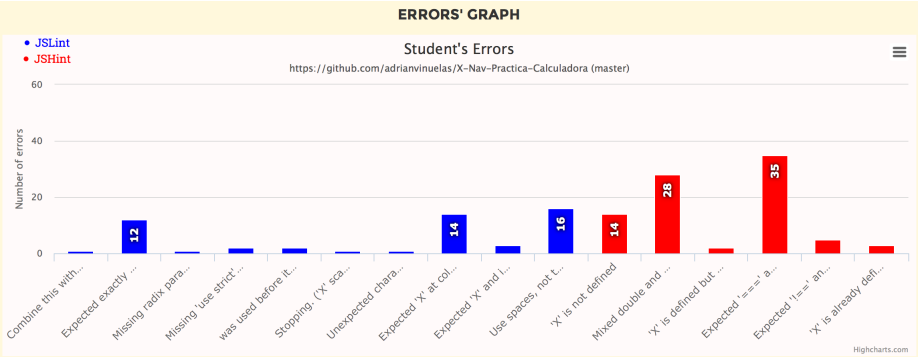

Figura 5.11: Gráfica que muestra los errores del ejercicio la primera vez que es analizado.

Como se puede apreciar, los errores más comunes son “Expected “X” at column “y”, not column “z” ”, “Use spaces, not tabs”, “Mixed double and single quotes”, y “Expected === and instead saw ==”. El alumno ha identificado los errores y los ha corregido antes de pasar el ejercicio de nuevo por la aplicación.

En un segundo análisis el alumno ha obtenido 171 errores. El número de errores ha aumentado debido a que en el primer análisis JSLint se paró cuando había analizado un 15 % del fichero (y JSLint solamente detectó 52 errores), y al haber arreglado el error que hizo que el análisis se detuviera ha permitido que en el segundo análisis JSLint llegue hasta un 34 % antes de detenerse de nuevo (esta vez por haber alcanzado el número máximo de errores permitido).

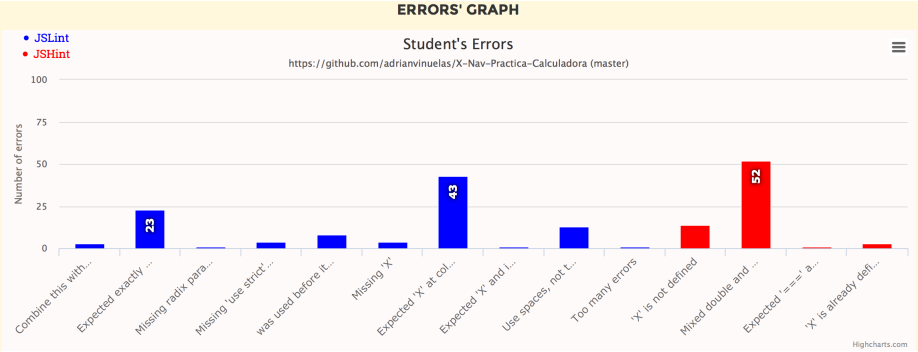

Figura 5.12: Gráfica que muestra los errores del ejercicio la segunda vez que es analizado.

Esta vez observamos que los errores más comunes siguen siendo “Expected “X” at column “y”, not column “z” ” y “Mixed double and single quotes”. Esto indica que de momento son los errores más comunes que el alumno ha tenido, y por tanto la aplicación le está obligando a corregirlos, lo que hace que el alumno asimile como se deben solucionar estos errores y repitiendo el proceso de corregir estos errores (43 y 52 veces por cada tipo de error) lo que se busca es que el alumno vaya cogiendo la metodología de programación adecuada (y que es uno de los objetivos del proyecto).

En el tercer análisis se observa que el alumno ha corregido la mayoría de los errores del tipo “Expected “X” at column “y”, not column “z” ” pero vemos que el número total de errores sigue siendo de 171. Esto se debe a que JSLint ha continuado analizando el fichero (hasta un 56 %) pero ha vuelto a llegar al número máximo de errores permitido.

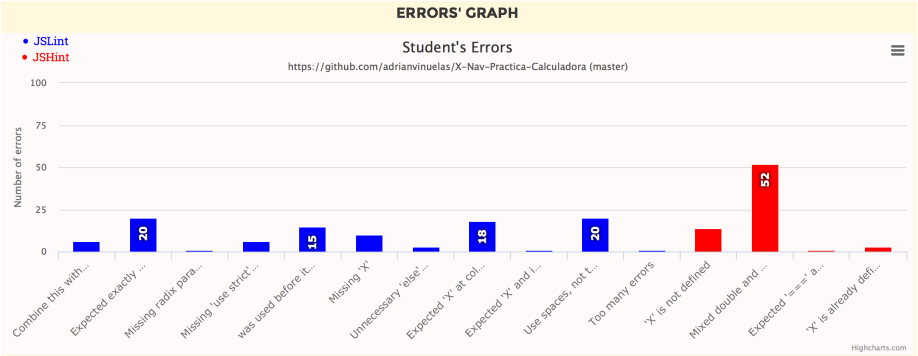

Figura 5.13: Gráfica que muestra los errores del ejercicio la tercera vez que es analizado.

Ahora el alumno ha corregido de nuevo los errores que tenía en el anterior análisis. Se observa que el número de errores ya comienza a bajar (ahora ha obtenido 149) al mismo tiempo que el fichero está casi analizado por completo (95 % escaneado). El número de errores no ha

bajado de forma considerable debido a que, aunque el alumno ha corregido gran parte de los errores, ahora se ha analizado un 39 % del fichero en el que se han seguido identificando más errores.

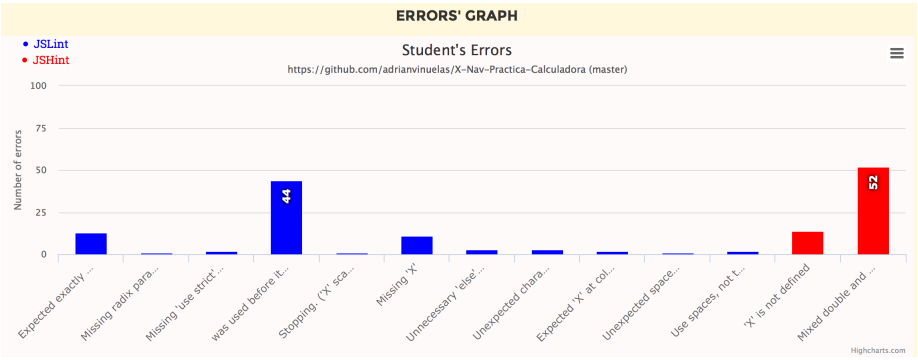

Figura 5.14: Gráfica que muestra los errores del ejercicio la cuarta vez que es analizado.

Como se puede observar el alumno aún no ha corregido los errores del tipo “Mixed double and single quotes”.

Ya casi con todo el fichero escaneado, el alumno vuelve a arreglar los errores identificados y el resultado obtenido es que con un 100 % del fichero analizado (y, por tanto, del ejercicio puesto que solo contiene un fichero .js para analizar) el alumno ha obtenido 64 errores.

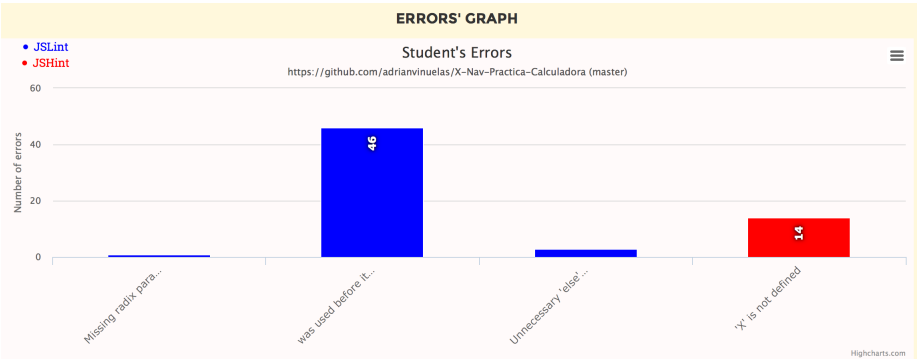

Figura 5.15: Gráfica que muestra los errores del ejercicio la quinta vez que es analizado.

Ahora podemos observar que el alumno ha corregido la mayoría de los errores que le ha ido mostrando la aplicación. Como ya se explicó en el Capítulo 3 el uso de JSLint y JSHint es a modo de consejo y no como reglas estrictas, de ahí que aún queden errores por corregir. Una vez finalizado el proceso vemos cómo ha sido su evolución.

```
#53 Stopping. (15% scanned).
     // Line 30, Pos 3
#101 Too many errors. (34% scanned).
     // Line 68, Pos 31
#101 Too many errors. (56% scanned).
     // Line 110, Pos 1
#83 Stopping. (95% scanned).
     // Line 192, Pos 10
```

Figura 5.16: Porcentajes escaneados del fichero por parte de JSLint.

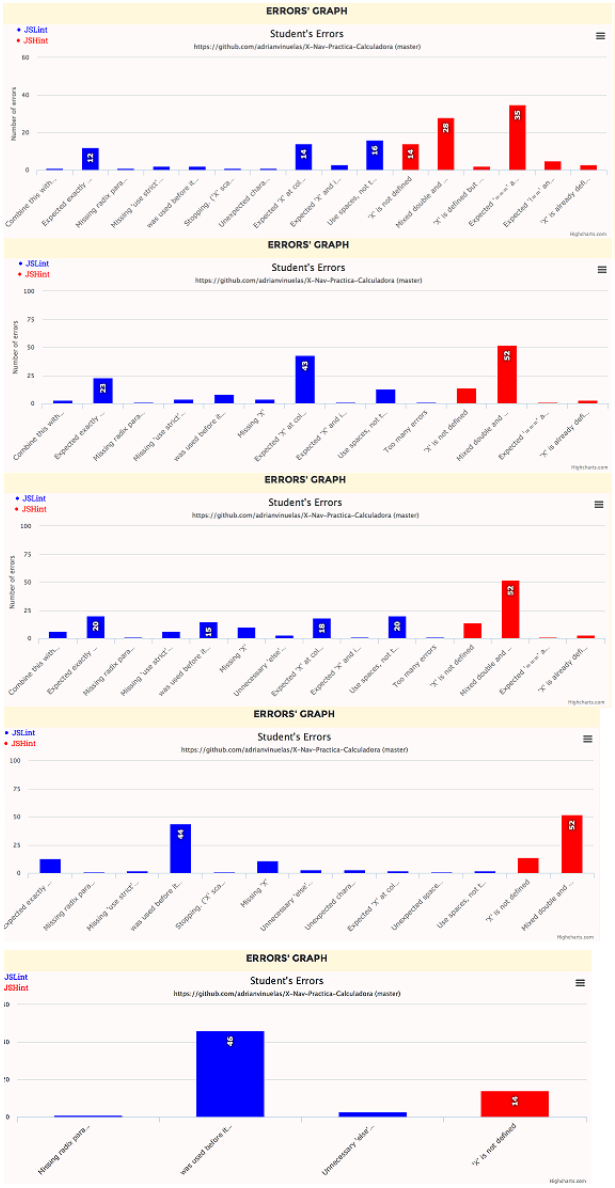

Figura 5.17: Comparativa de los cinco análisis realizados.

Una vez realizada toda la simulación podemos decir que el alumno ha ido identificando

cuáles son los errores más frecuentes que ha cometido, los ha ido solucionando poco a poco y asimilando los conceptos de por qué se identifican esos errores y cómo corregirlos. Repitiendo este proceso con todos los ejercicios que el alumno vaya desarrollando se conseguirá que el alumno vaya obteniendo una metodología de trabajo a la hora de programar en Javascript y que sus programas sean más legibles. Vemos un ejemplo de una función del código del alumno antes y después de analizar el ejercicio.

```
function verTecla(key){
    if(key>47 && key<58){//es un numero
        var numero = key - 48; //48 es por el caracter del 0 en el
        asignarnum(numero);
    }else if(((key > 42 && key < 48) && key != 44) || key == 120){
        if(key == 46){
            numAcumulado = $('#disp span').html() + ".";
            $('#disp span').html(numAcumulado);
        }else{
            var op = verOper(key);
            opera(op);
        }
    }else if(key == 13 || key == 61){
        igual();
    }else if(key == 99 || key == 67){
        $('#disp span').html("0");
        num1=0;
        num2=0;
        numAcumulado=0;
    }else if(key == 77 || key == 109){
        if(operacion != ""){
            asignarnum(numMemo);
        }else{
            $('#disp span').html("");
            asignarnum(numMemo);
        }
    }
}
```

Figura 5.18: Función desarrollada antes del primer análisis del ejercicio.

```
function verTecla(key) {
    'use strict';
    var numero, op;
    if (key > 47 && key < 58) {//es un numero
        numero = key - 48; //48 es por el caracter del 0 en el teclado,
        asignarnum(numero);
    } else if (((key > 42 && key < 48) && key !== 44) || key === 120) {
        if (key === 46) {
            numAcumulado = $('#disp span').html() + '.';
            $('#disp span').html(numAcumulado);
        } else {
            op = verOper(key);
            opera(op);
        }
    } else if (key === 13 || key === 61) {
        igual();
    } else if (key === 99 || key === 67) {
        $('#disp span').html('0');
        num1 = 0;
        num2 = 0;
        numAcumulado = 0;
    } else if (key === 77 || key === 109) {
        if (operacion !== '') {
            asignarnum(numMemo);
        } else {
            $('#disp span').html('');
            asignarnum(numMemo);
        }
    }
}
```

Figura 5.19: Función desarrollada después del último análisis del ejercicio.

Capítulo 6

Conclusiones

6.1. Consecución de objetivos

En la parte de ayudar al estudiante a entender la importancia de escribir un código legible y limpio se ha conseguido mediante:

- El análisis de los ejercicios, haciendo un seguimiento del alumno durante el curso de todos los ejercicios entregados y mostrando los errores más comunes que ha ido teniendo, para que mejore sus puntos débiles.
- La aplicación permite ver la explicación de por qué salen los errores, es decir, unicamente no ves que identifica errores, si no que da la posibilidad al alumno de que entienda el porqué surgen los errores y como solucionarlos.
- Se muestra al alumno la línea exacta en la que se identifica el error, para que no tenga que usar otra herramienta para abrir el fichero y luego buscar manualmente dónde está el error. Simplificar el trabajo ayuda a que la aplicación sea más atractiva para el usuario.

La parte de ayudar al profesor ha realizar un seguimiento de sus alumnos se ha conseguido mediante:

- Permite saber que alumnos no entregaron el ejercicio semanal y que alumnos sí (siempre que se trate de un ejercicio de Javascript) .
- Puede reconocer cuáles son los puntos en los que sus alumnos presentan debilidades a la hora de desarrollar código (No definir variables, tabulaciones ...) .

- Para cada ejercicio, permite ver el análisis de todos los alumnos de forma individual. Esto le permite hacer un seguimiento tanto colectivo como individual.

Puntos débiles

- Si se analizan librerías u otros ficheros necesarios en el ejercicio pero que no son del alumno, pierde eficacia puesto que se detectan errores que no son del alumno. La solución a este problema ha sido dar la posibilidad al usuario de introducir nombres de ficheros **.js** para evitar su análisis.

- El ranking de los alumnos no es 100 % preciso puesto que para aquellos alumnos en los que el análisis de JSLint encuentre un error grave y se detenga, tendrán menor porcentaje de errores. La solución a este problema ha sido introducir un segundo análisis con JSHint (que no lo detiene) para intentar ser más preciso.

- Solamente analiza un lenguaje en concreto, por lo que si hay ejercicios durante el curso en los que no se usa JavaScript, no se podrán analizar todos los ejercicios. En una versión futura de esta herramienta esto se podría mejorar introduciendo nuevas herramientas de análisis de otros lenguajes.

6.2. Aplicación de lo aprendido

1. A lo largo de mis años de estudio en el grado he podido aprender una serie de lenguajes de programación, de los cuales he usado Javascript para programar la parte del cliente y Python para programar la parte del servidor.

2. En la asignatura ISI (Ingeniería de Sistemas de Información) he aprendido todo lo relacionado con el lenguaje de programación Javascript.

3. En la asignatura DAT (Desarrollo de Aplicaciones Telemáticas) he aprendido todo lo relacionado con la parte del cliente. En la parte del cliente he aplicado mis conocimientos adquiridos en tecnologías como jQuery mediante el uso de las funciones que proporciona, al igual que jQuery-ui, y Bootstrap para el diseño visual de la aplicación (la interfaz de usuario) mediante el uso de sus componentes (glyphicons, panels, list groups, progress bars, buttons..) y herramientas de estilo CSS que nos permite usar Bootstrap.

4. En la asignatura SAT (Servicios y Aplicaciones Telemáticas) he aprendido todo lo relacionado con la parte del servidor, y por tanto de Django. En la parte del servidor he aplicado mis conocimientos adquiridos en Django, creando el servidor usando bases de datos MySQL para tratar todos los datos que se manejan en la aplicación (errores, ejercicios de estudiantes, ejercicios de profesor, ficheros, usuarios, y librerías). Por otro lado, para comunicar la parte del cliente con la parte del servidor he aplicado mis conocimientos sobre AJAX, que mediante llamadas POST Y GET he realizado el intercambio de información del cliente con el servidor, y luego en el servidor he tratado los datos acorde a las necesidades que el cliente mandaba en la petición AJAX. En la asignatura SAT también comencé a estudiar Python, un lenguaje con el que no había trabajado hasta ese momento.

6.3. Lecciones aprendidas

Durante la realización del Trabajo Fin de Grado he tenido que aprender el uso de nuevas tecnologías que en mi grado no he tenido la oportunidad de utilizar, ya que son tecnologías específicas para realizar ciertas tareas. Estoy hablando de Highcharts, JSLint y JSHint.

Estamos hablando de un Trabajo Fin de Grado en el que el objetivo principal es analizar código Javascript y por tanto necesitaba usar herramientas que me facilitasen esa tarea. Entonces descubrí lo que eran las herramientas de análisis estático de código (static analysis tools), JSLint y JSHint. Hasta el momento en el que comencé con el proyecto nunca antes había oído hablar de lo que eran este tipo de herramientas. Una vez tuve el conocimiento de su existencia, estuve realizando diferentes pruebas para entender su funcionamiento y entender por qué al pasarle un fichero Javascript muestra los errores que detecta. Al principio solo empecé usando JSLint, pero me di cuenta de que necesitaba llegar un paso más lejos al ver que en algunas ocasiones JSLint dejaba de analizar porque detectaba errores que consideraba muy graves y, como consecuencia, decidía parar el análisis de un fichero. Como mi objetivo era analizar todo un fichero entero, al ver que JSLint podía dejar de analizar un fichero y no completar el análisis, decidí complementar el proyecto con JSHint, que sí analizaba un fichero entero y complementaba la información que JSLint no proporcionaba en ciertas ocasiones.

JSLint y JSHint me parecen unas herramientas muy útiles y que pueden ayudar a los alum-

nos a desarrollar mejores códigos, que sean más legibles.

Por otro lado, una vez había conseguido analizar los ficheros, me di cuenta de que un simple análisis del fichero no era suficientemente atractivo aunque la información proporcionada era la deseada. Por esta razón busqué darle un toque más estadístico al proyecto, y para ello necesitaba usar gráficas en las que pudiera mostrar el análisis de todos los errores, así como las diferencias de errores entre unos ejercicios y otros, y la progresión a lo largo del tiempo en función del número de errores de los ejercicios analizados. Entonces descubrí Highcharts, que era una herramienta que se ajustaba perfectamente a mis requerimientos, y que al igual que JSLint y JSHint no conocía anteriormente. Highcharts me ha proporcionado las herramientas necesarias para cumplir mi objetivo.

Creo que Highcharts es una herramienta muy útil y muy dinámica que te permite realizar gráficas de distintos tipos y mostrar la información de múltiples maneras en función de lo que el usuario busca. He usado dos tipos de gráficas que creía las más adecuadas al proyecto, pero la herramienta te permite hacer infinidad de gráficas y gráficos muy interesantes. Además creo que con Highcharts he conseguido que el proyecto sea mucho más atractivo visualmente, y he conseguido que la información se pueda leer de manera rápida y sencilla.

6.4. Trabajos futuros

1. De cara a futuras versiones de este Trabajo Fin de Grado, creo que sería interesante poder dar la opción al usuario de modificar el fichero de configuración para filtrar que errores quiere el usuario que se tengan en cuenta y cuales no.

2. También añadirle nuevas herramientas de análisis de otros lenguajes para poder analizar distintos tipos de ficheros de diferentes lenguajes que puedan encontrarse en un ejercicio y así, de esta manera, solventar el problema de que ejercicios en los que no se use Javascript se queden sin ser analizados.

3. Dar la opción de poder visualizar los datos de distintas formas en lo que se refiere a las gráficas, es decir, poder proporcionar los datos en otro formato de gráfica si se desea.

4. Poder compartir el progreso o cierta información del análisis en redes sociales como Twitter o Facebook.

5. Adaptar el proyecto para que no solo funcione con repositorios de GitHub puede ser buena idea para que sea más accesible a otros usuarios.

6. Comparar el progreso que tienen los diferentes usuarios desde el punto de vista del profesor, en el sentido de que pueda ver el progreso de sus alumnos a lo largo del tiempo por cada ejercicio de profesor analizado.

6.5. Valoración personal

Una vez finalizado el proyecto me siento satisfecho del trabajo que he realizado, y creo que esto es una primera versión de una herramienta que puede ayudar a futuros estudiantes a mejorar su formación. Por otra parte, en lo personal siento que he aprendido nuevas herramientas que siguen complementando mi formación y que me ayudan a seguir creciendo en mi carrera profesional.

También estoy satisfecho al ver que lo que he aprendido durante mis años de estudio lo he podido plasmar en una aplicación, y ver que todo el trabajo previo ha dado sus frutos me reconforta. Anteriormente solo realizaba pequeñas prácticas en las que aplicaba los conocimientos adquiridos en las asignaturas por separado. Ahora he tenido la oportunidad de poder aplicar todos mis conocimientos en un solo proyecto, y eso para mi ha sido una experiencia fantástica y de la que me siento muy orgulloso.

Este trabajo lo considero como un gran paso final que doy antes de finalizar mis estudios, y que me ha servido para darme cuenta de todo lo que se puede llegar a hacer con trabajo, esfuerzo y dedicación. Es para mi esa guinda final que se pone al pastel una vez está acabado, haciendo un símil con mi grado y su finalización cuando entregue este proyecto.