

---

# Uso de WebAssembly para proporcionar pistas ligeras en jueces en línea

---



TRABAJO FIN DE GRADO  
GRADO EN INGENIERÍA DEL SOFTWARE  
CURSO 2020-2021

Adrián De Andrés Alonso  
Raúl Benavente Romero

*Directores*

Pedro Pablo Gómez-Martín  
Marco Antonio Gómez-Martín

Grado en Ingeniería del Software  
Facultad de Informática  
Universidad Complutense de Madrid

Septiembre de 2021



# Uso de WebAssembly para proporcionar pistas ligeras en jueces en línea

## *Autores*

**Adrián De Andrés Alonso  
Raúl Benavente Romero**

## *Directores*

**Pedro Pablo Gómez-Martín  
Marco Antonio Gómez-Martín**

**Grado en Ingeniería del Software  
Facultad de Informática  
Universidad Complutense de Madrid**

**Septiembre de 2021**



# Dedicatoria

*A mis padres, Ignacio y Rosa, por todo el apoyo que me han dado tanto económico como moral a lo largo de todo el grado. A mis tíos Esther, Roberto, Raúl y Ana por haberme animado en todo momento. A mi primo Diego por estar siempre dispuesto a ayudarme con las asignaturas del grado. A mis amigos de toda la vida y a los nuevos que he conocido estos años en la facultad por haber hecho que esto sea más llevadero. Mención a parte a mi tía Gema por contagiar siempre la alegría que la caracteriza. Gracias.*

- Adrián

*A mis padres, Yolanda y Manuel, y a mi hermano, Diego, por apoyarme durante los años de grado, especialmente por estos dos últimos años más complicados. A mis amigos de toda la vida, y a los que he conocido durante la carrera, por hacer más fácil sobrellevar este tiempo tanto dentro como fuera de la universidad.*

- Raúl



# Agradecimientos

*La gratitud en silencio no sirve a nadie.*

Gladys Berthe Stern

A nuestros tutores, Marco Antonio y Pedro Pablo, por el seguimiento y por su ayuda a lo largo de todo el proyecto.





# Resumen

Este proyecto trata de la integración de WebAssembly en un juez en línea. El motivo por el cual creímos necesaria dicha integración es reducir la carga de trabajo en el servidor, de manera que si un ejercicio no devuelve la salida esperada a ciertos casos básicos de prueba, no se envía al servidor. WebAssembly permite compilar archivos C, C++ y Rust y correrlos en el lado del cliente. Esto lo hace gracias a unas APIs de JavaScript, mediante las cuales puedes compartir funcionalidad entre JavaScript y WebAssembly. En resumidas palabras, permite compilar y ejecutar código C, C++ y Rust en el navegador. Ya que muchos jueces en línea tienen ejercicios para ser realizados en C/C++, WebAssembly nos ofrecía una posibilidad para comprobar la corrección básica de un ejercicio en el lado del cliente, esto es, antes de subirlo al servidor del juez en línea. De esta forma, se podría hacer que aquellos archivos C/C++ que no pasen los casos de prueba básicos no lleguen al servidor.

En este proyecto vamos a aplicar la tecnología WebAssembly sobre el juez online ya existente DOMjudge, el cual es utilizado en algunas asignaturas de este grado. Para ello, previamente hemos creado una simulación local de dicho juez en la que corroboramos la posibilidad de implementarlo en jueces en línea.

El proyecto está realizado en Linux, ya que es el sistema operativo necesario para instalar DOMjudge.

Todo el trabajo realizado se encuentra en: <https://github.com/raubenav/DOMjudge-WebAssembly>

## **PALABRAS CLAVE:**

Juez en línea, WebAssembly, cliente, servidor, DOMJudge, compilar, JavaScript, jQuery, Emscripten.



# Abstract

This project is about integrating WebAssembly into an online judge. The reason why we believe such integration is necessary is to reduce the workload on the server, so if an exercise does not return the output expected to certain basic, it is not sent to the server cases. WebAssembly allows you to compile C, C++ and Rust files and run them on the client side. This is made thanks to JavaScript APIs, through which you can share functionality between JavaScript and WebAssembly. In short, it allows you to compile and run C, C++ and Rust code in the browser. Since many online judges have exercises to be performed in C/C++, WebAssembly offered us a possibility to check the basic correctness of an exercise on the client side, that is, before uploading it to the online judge's server. In this way, C/C++ files that fail the basic samples could be prevented from reaching the server.

In this project we are going to apply the WebAssembly technology on the existing online judge DOMjudge, which is used in some subjects of this degree. To do this, we have previously created a local simulation of said judge in which we corroborate the possibility of implementing it in online judges.

The project is made on Linux, because it's the required operative system to install DOMjudge

All project's work can be accessed from: <https://github.com/raubenav/DOMjudge-WebAssembly>

## **KEYWORDS:**

Online judge, WebAssembly, client, server, DOMJudge, compile, JavaScript, jQuery, Emscripten.



# Índice

<b>Agradecimientos</b>	<b>VII</b>
<b>Resumen</b>	<b>IX</b>
<b>Abstract</b>	<b>XI</b>
<b>1. Introducción</b>	<b>1</b>
1.1. Motivación . . . . .	1
1.2. Objetivos . . . . .	2
1.3. Plan de trabajo . . . . .	2
1.4. Estructura del repositorio . . . . .	3
1.5. Asignaturas relacionadas con el TFG . . . . .	4
<b>2. Estado del arte</b>	<b>5</b>
2.1. Origen de JavaScript y su problemática . . . . .	5
2.2. Emscripten . . . . .	6
2.3. WebAssembly, ¿Qué es? . . . . .	7
2.3.1. Mozilla y asm.js . . . . .	7
2.3.2. Aparición WebAssembly . . . . .	8
2.4. Jueces en línea: DOMJudge . . . . .	8

---

2.4.1.	Interfaz de DOMJudge . . . . .	9
2.4.2.	Proceso de realización de entrega de un archivo . . . . .	10
2.4.3.	Posibles resultados de entregas de DOMJudge . . . . .	11
2.4.4.	Estructura y funcionamiento de DOMJudge . . . . .	12
<b>3.</b>	<b>Herramientas de desarrollo y metodología de trabajo</b>	<b>15</b>
3.1.	Herramientas de desarrollo . . . . .	15
3.1.1.	Sistemas Operativos . . . . .	15
3.1.2.	Herramientas de trabajo . . . . .	16
3.1.3.	Herramientas de coordinación . . . . .	17
3.2.	Tecnologías empleadas en DOMJudge . . . . .	17
3.2.1.	Back-end . . . . .	17
3.2.2.	Front-end . . . . .	18
3.3.	Metodología del trabajo . . . . .	19
<b>4.</b>	<b>Desarrollo del proyecto</b>	<b>21</b>
4.1.	Juez Local . . . . .	21
4.1.1.	Arquitectura . . . . .	21
4.1.2.	Desarrollo . . . . .	21
4.2.	DOMjudge - WebAssembly . . . . .	23
4.2.1.	Arquitectura del proyecto . . . . .	23
4.2.2.	Desarrollo . . . . .	25
<b>5.</b>	<b>Resultados, conclusiones y trabajo futuro</b>	<b>37</b>
5.1.	Resultados y conclusiones . . . . .	37
5.2.	Trabajo futuro . . . . .	37

---

<b>6. Aportaciones individuales</b>	<b>41</b>
6.1. Adrián De Andrés Alonso . . . . .	41
6.2. Raúl Benavente Romero . . . . .	42
 <b>I Apéndices</b>	 <b>45</b>
 <b>A. Guía de instalación</b>	 <b>47</b>
A.1. DOMjudge . . . . .	47
A.2. Emscripten y WebAssembly . . . . .	48
A.3. Apache2 . . . . .	49
A.4. Arrancar DOMjudge . . . . .	49
 <b>B. Archivos modificados DOMjudge</b>	 <b>51</b>
B.1. test_WebAssembly.js . . . . .	51
B.2. submit_scripts.html.twig . . . . .	51
B.3. submit_modal.html.twig . . . . .	52
B.4. style_domjudge.css . . . . .	52
 <b>C. Introduction</b>	 <b>55</b>
C.1. Motivation . . . . .	55
C.2. Objectives . . . . .	56
C.3. Workplan . . . . .	56
C.4. Repository structure . . . . .	57
C.5. Subjects related to the TFG . . . . .	57
 <b>D. Results, conclusions and future work</b>	 <b>59</b>
D.1. Results and conclusions . . . . .	59

D.2. Future work . . . . .	59
----------------------------	----

<b>Bibliografía</b>	<b>63</b>
---------------------	-----------



# Índice de figuras

2.1. Compatibilidad JavaScript con WebAssembly . . . . .	8
2.2. Vista general de DOMJudge . . . . .	9
2.3. Vista de usuario de DOMJudge . . . . .	10
2.4. Vista de administrador de DOMJudge . . . . .	11
2.5. Modal submit . . . . .	12
4.1. Login juez local . . . . .	22
4.2. Vista principal juez local . . . . .	22
4.3. Vista subida de archivos juez local . . . . .	23
4.4. Ejemplo subida fichero juez local . . . . .	23
4.5. Vista subida de archivos juez local . . . . .	23
4.6. Arquitectura proyecto . . . . .	24
4.7. Vista del modal con el botón ‘Test’ añadido . . . . .	25
4.8. Diagrama secuencia función botón ‘Test’ . . . . .	26
4.9. Error selección archivo . . . . .	26
4.10. Error selección problema . . . . .	27
4.11. Error lenguaje . . . . .	27
4.12. Diagrama secuencia compilación archivo C++ . . . . .	28

4.13. Mensaje para que el usuario sepa que ha empezado la compilación . . . . .	30
4.14. Error en el proceso de compilación . . . . .	30
4.15. Función para comparar la salida obtenida con la esperada . . . . .	34
4.16. Salida correcta . . . . .	35
4.17. Salida errónea . . . . .	35
A.1. Carpetas DOMserver . . . . .	48
A.2. Carpetas DOMserver y judgehosts . . . . .	48
A.3. Mensaje instalación Emscripten . . . . .	48
A.4. Carpeta WebAssembly . . . . .	49
A.5. Modificación apache2 . . . . .	49
B.1. Import de test_WebAssembly.js . . . . .	52
B.2. Botón id 'testFile' . . . . .	52
B.3. Div ventana de salida . . . . .	52

# Capítulo 1

## Introducción

**RESUMEN:** En este capítulo se describe el por qué de nuestro proyecto, los objetivos que se esperaba alcanzar, el plan elaborado para conseguir dichos objetivos y los lenguajes utilizados y sus relaciones con lo estudiado a lo largo del grado.

### 1.1. Motivación

Un juez en línea es una herramienta que permite cargar una serie de problemas con unos ficheros que ofrecen parámetros de entrada y de salida para códigos en diferentes lenguajes de programación. Ese es el caso de DOMjudge, que en la universidad se utiliza con fines educativos en algunas asignaturas. El funcionamiento general de DOMjudge consiste crear concursos, que se utilizan para agrupar a los alumnos por asignaturas, y subir una batería de ejercicios que permitan a los alumnos comprobar sus conocimientos en la asignatura. Los alumnos realizan entregas subiendo el código correspondiente a cada ejercicio indicando el ejercicio y el lenguaje en el que está programado, el código se manda al servidor donde se compila y se le ofrece al alumno una resolución sobre la compilación del código.

La motivación de este TFG es mejorar el sistema de entregas actual, que como se ha mencionado antes, consiste en subir los archivos al servidor y compilarlos y ejecutarlos allí, lo que deriva en que se realizan muchas entregas innecesarias que contienen fallos. Para realizar esta mejora, el proyecto ofrece una solución que permite compilar los ejercicios en el cliente desde el que se realiza la entrega y comparando la salida obtenida en ciertos casos de prueba con la esperada del ejercicio, evitando así que, si el código subido no supera los samples iniciales propuestos para ejercicio, se realice la compilación en el servidor reduciendo su carga de trabajo y una entrega fallida para el alumno.

Gracias a WebAssembly (Community, 2015) se pueden crear la solución a estos problemas, de manera que sea el propio cliente quien ejecute el archivo que se acaba de subir. De esta manera, si el resultado de la ejecución no es el esperado, se evitará que el archivo llegue al servidor, creando así un primer filtro de seguridad y validación de resultados para estos jueces, evitando así la compilación de posibles archivos dañinos para el servidor.

## 1.2. Objetivos

El principal objetivo de este proyecto es ampliar la funcionalidad de DOMjudge para que sea capaz de comprobar los ejercicios previamente de subirlos al servidor. Para realizar dicha funcionalidad, es necesario crear un botón nuevo que al pulsarlo compile en el cliente el archivo C++ que se va a subir y compare la salida obtenida con la salida esperada. La meta final es conseguir que la funcionalidad de dicho botón pueda ser añadida al botón de subida al servidor, de manera que antes de enviar el archivo al servidor se compile primero en WebAssembly. De esta forma, tanto los jueces en línea como la propia universidad podrían ser capaces de implementarlo.

## 1.3. Plan de trabajo

Para poder lograr los objetivos, se siguió un plan de trabajo dividido en 4 fases. Dichas fases del plan se describen a continuación:

1. **Investigación posibilidades de WebAssembly.** En primer lugar, nos informaremos mediante diversas fuentes en qué consiste exactamente WebAssembly y, una vez asimilado esto, estudiaremos cual es su funcionamiento y cuales son sus posibles usos. Además, comenzaremos con la instalación de todo lo necesario para su funcionamiento. Una vez tengamos todo lo necesario instalado, comenzaremos a probar el funcionamiento de esta tecnología. Para ello, buscaremos tutoriales o guías de problemas sencillos, como algún 'Hello World!'. Con esto, habremos conseguido crear nuestro proyecto simple con WebAssembly y sobretodo, habremos podido comprobar que nos funciona correctamente. De esta forma, tendremos una base sólida para realizar la siguiente fase. El tiempo previsto para realizar esta fase es de aproximadamente el primer cuatrimestre, es decir, hasta febrero.
2. **Simulación local de un juez en línea.** En la segunda fase nos centraremos principalmente en crear un juez local que simule el funcionamiento de DOMjudge, en el que comprobaremos que el objetivo de este TFG es alcanzable. El objetivo principal de esta simulación es conseguir compilar mediante WebAssembly un archivo escrito en C++ correspondiente a un posible ejercicio de la

universidad. En caso satisfactorio, habría que trasladar dicha funcionalidad a DOMjudge. El tiempo estimado para esta fase es de aproximadamente un mes, es decir, hasta marzo.

3. **Estudio e instalación de DOMjudge.** DOMjudge es un programa complejo, por lo que tiene un proceso de instalación bastante tedioso y complicado. Una vez instalado, tenemos que estudiar como esta programado (que lenguaje utiliza, que tecnologías, estándares, etc) para saber exactamente que parte del código deberemos modificar para añadir la nueva funcionalidad. Es por ello que el tiempo estimado para dicho estudio e instalación es de unas dos semanas aproximadas.
4. **Integración de WebAssembly en DOMJudge.** Esta es la parte principal del proyecto, en la que se va a desarrollar la nueva funcionalidad. Para integrar dicha funcionalidad hará falta utilizar Node.js (Wikipedia, Node.js), ya que es necesario para compilar WebAssembly. Una vez estudiado el lenguaje de DOMjudge se buscará la manera más óptima y eficaz de integrar Node.js. Esta fase va a ser la más complicada debido al desconocimiento sobre la materia, por lo que no se puede predecir de antemano el tiempo estimado ni los recursos que se necesitarán para ello. Sin embargo, lo que si se puede predecir es que tanto la funcionalidad nueva necesaria para alcanzar el objetivo del TFG como la redacción de esta memoria se va a desarrollar en esta etapa. Es por ello que esta etapa durará hasta la entrega final del proyecto.

## 1.4. Estructura del repositorio

El repositorio empleado para el desarrollo del TFG, como se ha mencionado anteriormente, es GitHub, que sigue la siguiente estructura:

- **WebAssembly:** Directorio empleado para almacenar el código del servidor Node.js programado en JavaScript que permite compilar a WebAssembly las subidas del juez.
- **domjudge-7.3.3:** Directorio descargado directamente de la página de DOMjudge, que contiene los archivos de domjudge-7.3.3 sobre los cuales hicimos las modificaciones necesarias para el funcionamiento de nuestro proyecto, y al cual añadimos un archivo .js necesario.
- **README.md:** Archivo que contiene la descripción del proyecto y el contenido del repositorio.

## 1.5. Asignaturas relacionadas con el TFG

A continuación, se listan las asignaturas del grado que nos han servido para conseguir realizar este proyecto:

### 1. Aplicaciones Web (AW)

En esta asignatura se enseñan los conocimientos básicos de la programación web, así como la utilización del lenguaje HTML, CSS, JavaScript, Node, etc. Esta asignatura es la más útil para el TFG, ya que todo el proyecto está orientado al desarrollo.

### 2. Bases de datos (BD)

En esta asignatura se aprende a manejar bases de datos relaciones. Gracias a esto se adquieren conocimientos que permiten de manejar la base de datos de DOMJudge, desarrollada en MariaDB.

### 3. Sistemas Operativos y Administración de Sistemas y Redes (SO y ASR)

En esta asignatura se aprenden aspectos básicos para manejar cualquier distribución de Linux, así como instalar y utilizar máquinas virtuales. Útil, sobre todo, a la hora de la instalación de la máquina virtual y del DOMJudge.

### 4. Ingeniería del Software y Modelado de Software (IS y MS)

En estas asignaturas se aprenden los patrones de diseño más importantes utilizados en la elaboración de proyectos software. Sirve, sobre todo, a la hora de crear la simulación local.

### 5. Arquitectura Interna de Linux y Android (LIN)

En esta se aprende a manejar algunos de los procesos internos de linux, lo que resulta muy útil para poder instalar y manejar DOMJudge, ya que todo el juez se tiene que instalar en una sistema operativo en base Linux.

### 6. Gestión de Proyectos Software (GPS)

En esta asignatura se aprende a organizar un proyecto de software. Aunque no se terminen de aplicar ninguno de los métodos estudiados para la organización interna, es útil para saber como organizarlo en función de diferentes disponibilidades

## Capítulo 2

# Estado del arte

**RESUMEN:** En este capítulo se comenta el estado actual en el que se encuentran las tecnologías y como se ha llegado ahí. Para este TFG destacan las tecnologías: WebAssembly, JavaScript, DOMjudge y Emscripten.

### 2.1. Origen de JavaScript y su problemática

JavaScript fue diseñado en 1995 con una funcionalidad muy básica, apenas se utilizaba para la creación de banners, botones y elementos básicos de cualquier página web. Sin embargo, fue desarrollando sus capacidades hasta convertirse en uno de los lenguajes más utilizados en la actualidad.

En 2009 apareció Node.js, que es un intérprete de JavaScript independiente externo al navegador, que se puede ejecutar en cualquier sitio y que se pensó para poder usar JavaScript en el lado del servidor, evitando que los desarrolladores tuvieran que usar varios lenguajes ya que combinó el desarrollo de JavaScript y una API de E/S para construir una plataforma que permitiera el usar JavaScript para desarrollar un servidor.

Sin embargo, había un gran problema: si se deseaba utilizar una código ya existente escrito en C o C++ se debía reescribir a JavaScript para que el navegador pudiera interpretarlo. Esto hizo que aparecieran los denominados transpiladores, que eran compiladores encargados de convertir código fuente escritos en algún lenguaje de programación en un código fuente equivalente en otro lenguaje de programación distinto. En este sentido, aparecieron muchos transpiladores encargados de convertir cualquier código fuente en cualquier lenguaje de programación en código fuente apto para JavaScript. Dos de los principales proveedores de navegadores web - Goo-

gle y Mozilla - empezaron a trabajar en posibles soluciones a este problema. Nos centraremos en Mozilla, ya que fue el desarrollador de WebAssembly

## 2.2. Emscripten

Para poder trabajar con WebAssembly, es necesario disponer del código que queramos ejecutar en el navegador compilado en formato `.wasm`. Para ello se hace uso de una herramienta que nos permita compilar el código a uno de esos formatos: *Emscripten*.

Emscripten (Battagline, 2019, cap. 1) es un compilador Source-to-source que permite compilar código escrito en C y C++ en WebAssembly para su posterior ejecución en navegadores web. La característica principal de esta herramienta es su eficiencia a la hora de compilarse en navegadores web, con velocidades comparables a JavaScript. Una vez que Emscripten ha compilado el código escrito en C/C++, genera un fichero con una extensión `.wasm` y otro fichero `.js`. El fichero `.wasm` está escrito en código binario y en él se encuentra toda la funcionalidad que contiene el fichero original escrito en C/C++. El fichero `.js` está escrito en muy alto nivel y hace uso de funciones propias de WebAssembly que permiten ejecutar el fichero `.wasm` y realizar su funcionalidad. De esta forma es como se consigue compilar y ejecutar código C/C++ en el navegador.

Dicha compilación de Emscripten se realiza con el siguiente comando:

***emcc [..args..]***

Siendo el primer argumento del comando el que usaremos para indicar que archivo queremos compilar, además tiene numerosas opciones de compilación, entre las que destacamos, por ser necesarias para el desarrollo de nuestro proyecto, las siguientes:

- **-o *nombreArchivo.js*** que sirve para generar un archivo *JavaScript* con la compilación, que es muy útil dado que permitirá el correcto funcionamiento del archivo binario en el navegador y, además, nos dará varias herramientas necesarias para nuestra aplicación, como la capacidad de procesar las líneas que escribiría el código original `.cpp` y dirigirlos de otro modo, con lo que se podría coger los `cout` del programa y redirigirlos a un array con el que procesar después el resultado.
- **-s WASM=0 ó WASM=1** que nos permitirá elegir si queremos que se genere un archivo `.wasm` durante la compilación
- **-O0 | -O1 | -O2 | -O3 | -Os | -Oz** que nos permite elegir el nivel de optimización que tendrá el código generado siendo **-O0** el menor nivel de optimización,



pero que emplea un menor tiempo de compilación, y **-Oz** el mayor, con una diferencia en el tiempo de compilación considerable.

- **-s EXPORTED\_FUNCTIONS=\_nombreFuncion1,\_nombreFuncion2..** que nos permite exportar referencias a funciones para poder llamarlas después, útil para poder realizar llamadas al *main()* de los códigos *.cpp* compilados.
- **-s NO\_EXIT\_RUNTIME=1** que es necesario para que no se termine la ejecución una vez llamado al *main()* y poder realizar llamadas posteriores al código ya compilado
- **-s ALLOW\_MEMORY\_GROWTH=1** necesario para que el código incluya funciones que permitan reescalar el tamaño de la memoria empleada y evitar así fallos.
- **-embed-file <nombreArchivo>** que es necesario para precargar un archivo que sea necesario utilizar luego, en nuestro caso, los archivos que contienen los inputs de los casos de prueba del juez.

## 2.3. WebAssembly, ¿Qué es?

Es un lenguaje de bajo nivel con un formato binario, similar a lenguaje ensamblador, que se ejecuta con un rendimiento óptimo y permite que lenguajes como C/C++ puedan ser compilados y ejecutados en web. Además, está diseñado para implementarse con JavaScript, de manera que ambos trabajen juntos en la programación web

Es por ello que WebAssembly (Haas et al., 2017) se usa, sobre todo, en la programación web, ya que permite ejecutar código escrito en múltiples lenguajes de programación de bajo/medio nivel a la par que aplicaciones cliente, algo que hasta entonces no era posible hacer

### 2.3.1. Mozilla y asm.js

En 2013, Mozilla (Mozilla y individual contributors, 1998) desarrolló una forma de traducir código fuente escrito en C y C++ a JavaScript, el subconjunto asm.js. Este archivo logra mejorar el rendimiento de ejecución al administrar la memoria de forma manual y tener un 100% de consistencia de tipos. Se utiliza la herramienta Emscripten (Contributors, 2015), que sirve para compilar el código y, con ello, generar el asm.js. Este archivo es fácilmente distribuible, ya que es código JavaScript, y por lo tanto, hace uso del mecanismo AOT (Ahead Of Time), que es una técnica que usa el motor de JavaScript para compilar el código en un código de máquina nativo, de ahí su nombre, ya que coge el código JavaScript y lo convierte en ensamblador.

### 2.3.2. Aparición WebAssembly

En abril de 2015, la World Wide Web Consortium, abreviado como W3C, formó un grupo de trabajo para estandarizar WebAssembly y supervisar el proceso de especificación y propuesta. La implementación inicial de WebAssembly se basó en las características de asm.js, con una principal ventaja: carga más rápida, especialmente en bases de códigos grandes y en dispositivos móviles WebAssembly no reemplaza a JavaScript, es una nueva característica de sus motores que se basa en su infraestructura, lo que se traduce en un buen acoplamiento. Esto se puede ver en tres puntos clave:

- Tiene la misma estrategia de evolución, lo que significa que todo siempre es compatible con versiones anteriores.
- Permite llamadas síncronas desde y hacia JavaScript, incluyendo las API del navegador.
- La seguridad se maneja al igual que se hace con JavaScript, esto es, con las mismas políticas, origen y permisos.

Como ya se ha comentado, WebAssembly es un formato binario totalmente distinto a JavaScript. Sin embargo, debido a la aparición de WebAssembly, los navegadores saben interpretar ambos lenguajes. Para que esto sea posible y que el código C/C++ se traspile a WebAssembly, los navegadores tienen que tener su propio motor WebAssembly. Esta diferencia y compatibilidad al mismo tiempo se puede observar en la figura 2.1

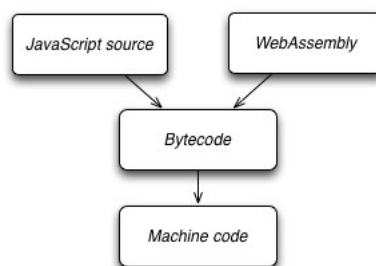


Figura 2.1: Compatibilidad JavaScript con WebAssembly

## 2.4. Jueces en línea: DOMJudge

Actualmente, una parte fundamental del modelo educativo, que incluye enseñanza online, hace uso de los jueces en línea como parte de su modelo educativo, ya que permiten tener concursos durante determinados periodos de tiempo, lo cual es

útil para mantener, durante la duración del curso, el juez activo para evaluar los conocimientos de los alumnos sobre la asignatura en cada parte del temario, ya que se pueden ir activando diferentes ejercicios en función del punto en el que se esté.

Existen una multitud de estos jueces que dan servicio de forma constante, y no mediante periodos determinados por la duración de los concursos, como pueden ser [onlinejudge.org](http://onlinejudge.org) o [acceptaelreto.com](http://acceptaelreto.com) que se mantienen activos permanentemente y actualizan de forma periódica sus ejercicios.

### 2.4.1. Interfaz de DOMJudge

DOMJudge (Pham y Nguyen, 2019) tiene distintos tipos de usuarios: los comunes que realizan entregas y pueden consultarlas, los jueces que pueden ver los envíos, analizarlos, anularlos y realizar y contestar aclaraciones sobre entregas y los administradores que manejan el juez. En la figura 2.2 se observa la pantalla general de un concurso creado en DOMJudge, que en el caso de una asignatura de la Facultad, el concurso no sería un concurso en sí, sino una forma de subir ejercicios agrupados por temas y permitir a los alumnos hacer subidas de cada uno de ellos para valorar el conocimiento que tienen sobre la asignatura.

DOMjudge													
<a href="#">Home</a> <a href="#">Problemset</a> <a href="#">Scoreboard</a> <a href="#">Submit</a> <span>2:24:53</span>													
The 2018 ACM ICPC World Finals													
starts: 21:00 - ends: 02:00													
RANK	TEAM	SCORE	A	B	C	D	E	F	G	H	I	J	K
1	<b>unsigned</b> University of Engineering and Technology - VNU	4 162		12 1 try		27 1 try			100 1 try	23 1 try			
2	<b>Nebula</b> Huazhong University of Science & Technology	4 175		13 1 try		34 3 tries				42 3 tries			6 1 try
3	<b>Triangulation</b> Indian Institute of Technology - Roorkee	4 267		51 1 try		56 2 tries			0 + 1 tries	36 2 tries			84 1 try
4	<b>Pachirisu</b> Fuzhou University	3 91		11 1 try		51 1 try				29 1 try			
5	<b>NCTU_Foudre</b> National Chiao Tung University	3 101		14 1 try	0 + 1 tries	47 1 try				40 1 try			
6	<b>Ukkonen Fan Club</b> University of Helsinki	3 112	1 try	22 1 try		42 2 tries	1 try	0 + 1 tries		28 1 try			
7	<b>Unicorn</b> University of Illinois at Urbana-Champaign	3 125		11 1 try		108 1 try				6 1 try	2 tries	2 tries	
8	<b>Pragma</b> ITESM Campus Queretaro	3 155		51 1 try		30 1 try			74 1 try				
9	<b>Cxiv-Dxiv</b> The University of Tokyo	3 155		13 1 try					105 2 tries	17 1 try			
10	<b>MIT TWO</b> Massachusetts Institute of Technology	3 181		58 2 tries		44 3 tries				19 1 try			
11	<b>Perm SU: Fire Mind</b> Perm State University	3 182		12 1 try		48 1 try							102 2 tries
12	<b>Codembia</b> Columbia University	3 191		20 1 try		45 1 try				106 2 tries			
13	<b>LNU Algotesters</b> Lviv National University	3 214	1 try	18 1 try		58 1 try				98 3 tries			

Figura 2.2: Vista general de DOMJudge

La figura 2.3 se corresponde con la pantalla personal de un participante, en nuestro caso, la de un alumno, en la que se observa: el puesto que se ocupa en el concurso, junto con la cantidad de ejercicios entregados correctamente y la puntuación obtenida por los mismos, el estado de los resultados de las entregas realizadas para cada problema, presentando el número de entregas realizadas para ese ejercicio, aclaraciones realizadas sobre cada entrega y cada una de sus entregas realizadas acompañada de información relativa a la misma, entre la que está la hora, el problema, el lenguaje en el que se ha escrito el código, en nuestro caso nos centramos en C++ y el resultado de la entrega.

The screenshot shows the DOMJudge user interface for a participant. At the top, there's a navigation bar with 'DOMjudge', 'Home', 'Problemset', and 'Scoreboard' links, along with a 'Submit' button and a clock showing '2:22:56'. Below this is a table showing the participant's rank (6), team name ('Ukkonen Fan Club'), and score (3/112). A progress bar shows the status of various problems (A through K) with their respective scores and attempts. Below the progress bar, there are two main sections: 'Submissions' and 'Clarifications'. The 'Submissions' section lists recent submissions with columns for time, problem, language, and result. The 'Clarifications' section shows a list of clarification requests with columns for time, from, to, subject, and text. A 'request clarification' button is also visible.

RANK	TEAM	SCORE	A	B	C	D	E	F	G	H	I	J	K
6	Ukkonen Fan Club University of Helsinki	3 / 112	1 try	22 1 try		42 2 tries	1 try	0 + 1 tries		28 1 try			

time	problem	lang	result
2:34	C	CPP	PENDING
2:12	H	CPP	PENDING
1:41	F	JAVA	PENDING
1:39	E	CPP	COMPILER-ERROR
1:39	A	CPP	COMPILER-ERROR
0:46	H	CPP	CORRECT
0:42	D	CPP	CORRECT
0:41	H	CPP	WRONG-ANSWER
0:35	D	CPP	WRONG-ANSWER
0:28	H	CPP	CORRECT
0:25	B	CPP	CORRECT
0:22	B	JAVA	CORRECT

time	from	to	subject	text
2:36	Jury	All	problem C	Read the problem statement.

time	from	to	subject	text
2:35	You	Jury	problem C	Do I have to connect all dots?

request clarification

Figura 2.3: Vista de usuario de DOMJudge

La figura 2.4, muestra la interfaz que tiene un usuario administrador. Un administrador puede añadir o eliminar usuarios del concurso, añadir, eliminar o modificar problemas, o crear y borrar lenguajes de compilación distintos, entre otras cosas. En general se encarga de que los concursos tengan los usuarios organizados por equipos, problemas, lenguajes y judgehost necesarios para que funcionen correctamente y de solucionar los problemas asociados a los mismos que puedan surgir.

#### 2.4.2. Proceso de realización de entrega de un archivo

Para realizar una entrega un usuario, que participe en un concurso, debe estar logeado y pulsar en el botón submit de la barra superior de la vista de usuario fi-

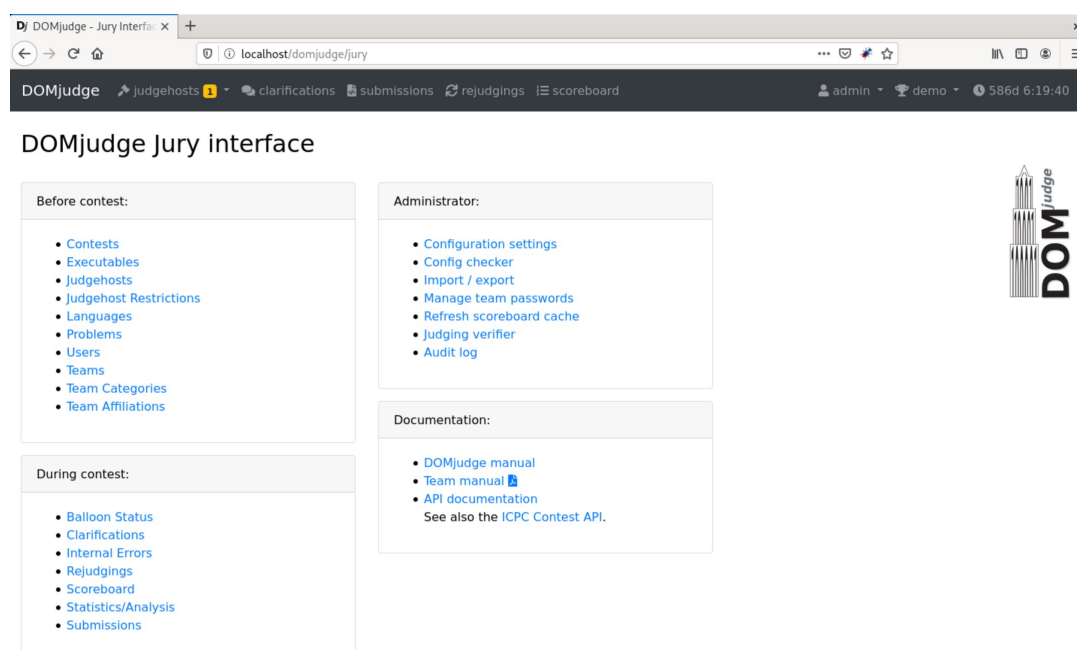


Figura 2.4: Vista de administrador de DOMJudge

gura 2.3, lo cual desplegará la ventana de la figura 2.5 en la que el usuario deberá seleccionar un archivo, el problema para el que realiza la entrega y el lenguaje de la entrega (que si es uno de los lenguajes permitidos en el concurso se detectará automáticamente una vez seleccione el archivo), pulsar en el botón de submit del modal y esperar a que el resultado de la entrega aparezca en el apartado de Submissions de la vista de usuario figura 2.3.

### 2.4.3. Posibles resultados de entregas de DOMJudge

Los resultados posibles después de realizar una entrega en DOMJudge son los siguientes:

- **CORRECT:** Entrega realizada correctamente, sin errores.
- **COMPILER-ERROR:** El código tiene algún tipo de error que no permite, ni tan siquiera, compilarlo
- **TIMELIMIT:** El código enviado supera el tiempo de ejecución máximo permitido, puede estar originado por un código mal optimizado hasta bucles infinitos
- **RUN-ERROR:** Error durante la ejecución, quiere decir que compila correctamente, pero falla debido a cosas como divisiones por cero o intentar asignar un valor a un array en una posición que no existe, por ejemplo.

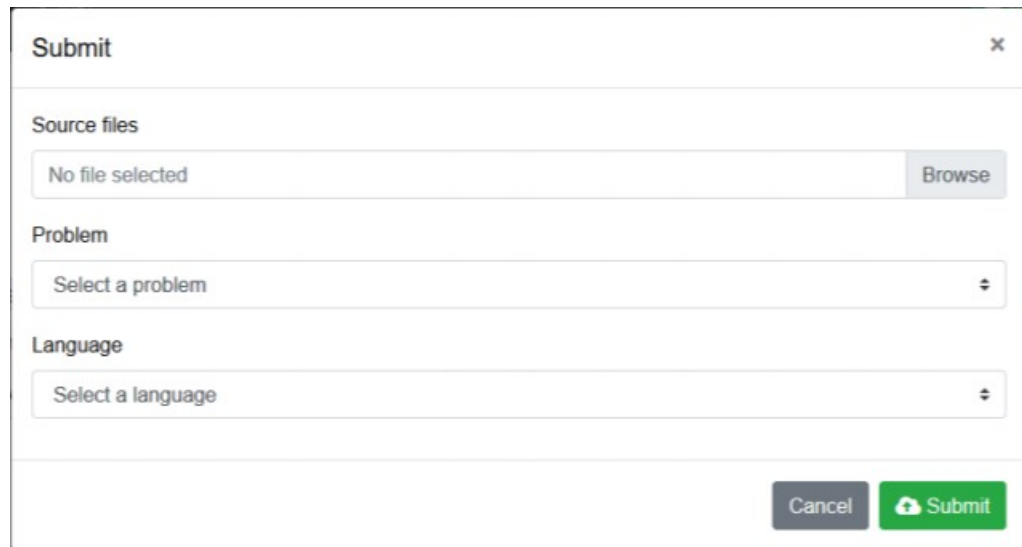
A modal window titled "Submit" with a close button (X) in the top right corner. The form contains three sections: "Source files" with a text input showing "No file selected" and a "Browse" button; "Problem" with a dropdown menu showing "Select a problem"; and "Language" with a dropdown menu showing "Select a language". At the bottom right, there are two buttons: "Cancel" and "Submit" (which is green and has a cloud icon).

Figura 2.5: Modal submit

- **NO-OUTPUT:** El programa no genera una salida.
- **OUTPUT-LIMIT:** La salida generada por el programa es mayor de la permitida.
- **WRONG-ANSWER:** No se produce ninguno de los errores anteriores pero la salida generada por el código es errónea.
- **TOO-LATE:** Intentar entregar un código cuando el tiempo permitido para hacerlo se ha acabado.

Además el juez permite solicitar aclaraciones sobre algunas entregas, para obtener información sobre por qué se ha obtenido un resultado, esta aclaración llegaría a alguno de los jueces de ese concurso, en nuestro caso los profesores, y estos tendrían que responder aportando mayor información sobre dicha entrega. Estas aclaraciones se pueden solicitar en un botón situado a la derecha de la lista de entregas, justo debajo de las listas de clarificaciones hechas por los profesores y solicitadas por los alumnos.

#### 2.4.4. Estructura y funcionamiento de DOMJudge

Ahora pasaremos a explicar como está organizado el código de DOMJudge, y por tanto, la parte con la que interaccionan los desarrolladores. Esta se compone de dos partes:

- **Domserver:** Centraliza todo el código empleado en el funcionamiento tanto de

la interfaz web descrita antes como de la API donde se conectan los usuarios que realizan entregas, los jueces del concurso, y los llamados judgehosts, descritos más adelante. Emplea una estructura cliente-servidor, como es habitual, por lo tanto es compuesto de dos partes:

- **Frontend:** Está compuesto por todas las vistas de la aplicación web y sus elementos, como la pantalla en donde se muestran las subidas del usuario o el propio modal para realizar dichas subidas, empleando el lenguaje Javascript para gestionar algunos de estos comportamientos en el cliente. Todas las vistas y elementos se generan con Twig.
- **Backend:** Engloba prácticamente todo el código que utiliza el juez para gestionar su funcionamiento, desde el control de las subidas de archivos, una vez se pulsa el botón para enviarlos, hasta el almacenamiento de dichas subidas en la base de datos del propio juez. Para ello hace uso de PHP, el cual utiliza para manejar los archivos de las subidas, los samples de los ejercicios y la base de datos, entre otras cosas.
- **Judgehost/s:** Son los encargados de juzgar el código de las entregas realizadas por los usuarios. Para que el juez funcione correctamente es necesario tener al menos un judgehost operativo. Estos judgehost permanecen siempre activos a la espera de recibir nuevas entregas y compilarlas. Se pueden modificar para compilar cualquier tipo de lenguaje, o los que se especifiquen, o para solo ser utilizados en caso de que se tenga que rejuzgar el código de alguna entrega. Si se realizan entregas cuando no hay ningún judgehost activo quedarían en estado de pending, y cuando se encontrase uno operativo si irían juzgando una a una por orden de entrega.

Por lo tanto DOMJudge funciona de la siguiente forma: la parte del servidor (dom-server) genera las plantillas de la página web, incluidas todas las vistas de la página de subidas y vista general del concurso antes mencionada. Proporciona también el modal que se despliega cuando un usuario va a realizar una entrega, y se encarga de construir un objeto con los datos necesarios obtenidos del modal, que son, el lenguaje de compilación y el problema para el que se realiza la entrega, además almacena el código de la subida, el usuario que la realizó y el concurso. Con la entrega generada, los judgehost se comunican con el servidor para ver si hay entregas que juzgar y le solicitan los elementos necesarios, que son el código y los diferentes casos de prueba, al juez. Estos elementos llegan al judgehost que es el encargado, con la información que dispone, de compilar el código para los casos de entrada y compararlo con los casos de salida para generar una respuesta. Esto vuelve al domserver que almacena la subida incluyendo problema para el que se realizó, hora en la que fue realizada, el código y la respuesta recibida, generando una entrada más en la lista de entregas correspondientes del usuario que la realizó mostrando la información mencionada antes.





## Capítulo 3

# Herramientas de desarrollo y metodología de trabajo

**RESUMEN:** En este capítulo se describen las herramientas y las tecnologías utilizadas para desarrollar y organizar todo el trabajo llevado a cabo en el proyecto.

### 3.1. Herramientas de desarrollo

En esta sección se detallarán todas las herramientas utilizadas a lo largo de todo el proyecto, ya sea para la implementación de WebAssembly en DOMJudge, para el desarrollo del juez de prueba o para el desarrollo de la memoria.

#### 3.1.1. Sistemas Operativos

En este apartado se describen los dos sistemas operativos con los que hemos trabajado a lo largo de todo el proyecto

- **Windows 10:** Empleado a lo largo de todo el proyecto, especialmente durante las dos primeras fases, las fases de investigación y simulación local de un juez, debido a que es un sistema operativo que tiene soporte para todas las herramientas utilizadas.
- **Debian 10:** Empleado en la fase final del proyecto, la que implica trabajo directo sobre DOMjudge, debido a que, como ya se ha mencionado, se necesita un sistema operativo con base Linux para poder instalar dicho juez y sus componentes.

### 3.1.2. Herramientas de trabajo

En este apartado se describen los programas y aplicaciones utilizados durante el desarrollo del proyecto.

- **Visual Studio Code:** Es un editor de código fuente que permite trabajar sobre muchos lenguajes de programación, que esta disponible tanto en Windows como en Linux. Fue utilizado para desarrollar el juez local así como para integrar WebAssembly en DOMJudge. Uno de los grandes motivos por lo que resulta tan útil este editor es por la extensión VisualShare, la cual permite trabajar sobre el mismo código simultáneamente entre varios equipos.
- **Emscripten:** Es un programa informático necesario para realizar la compilación de C/C++ a WebAssembly. Este programa es el que permite generar un archivo de código binario que contiene toda la funcionalidad desarrollada en C/C++ que, mediante ciertas funciones propias de WebAssembly ejecutadas en JavaScript, va a poder ser compilado en el cliente.
- **Oracle VM Virtualbox:** Es una herramienta que sirve para crear, configurar y utilizar máquinas virtuales con sistemas operativos diferentes al propio del PC. Debido a que en las primeras etapas se trabajó únicamente con Windows, este programa sirvió para poder utilizar Debian en los equipos sin necesidad de instalarlo, pudiendo acarrear pérdidas de información o delay ocasionados por sobrecarga de la memoria.
- **XAMPP:** Es una herramienta que sirve para gestionar bases de datos MySQL, muy útil para desarrollar proyectos de prueba en local, pudiendo utilizar bases de datos sin mucha complejidad.
- **OverLeaf:** Es un editor de LaTeX online, el cual se utilizó para redactar esta memoria. Esta herramienta fue empleada porque ofrece la opción de redactar texto de manera simultánea, de forma que varias personas pueden trabajar sobre los mismos archivos y los cambios se aplican en tiempo real.
- **DOMJudge:** Es un juez en línea automatizado, ampliamente utilizado en la facultad que permite subir y ejecutar ejercicios hechos por los alumnos para comprobar que la funcionalidad del código subido es la correcta. Se eligió como juez sobre el que desarrollar este proyecto ya que, además de ser empleado de manera frecuente en la facultad, es libre y de código abierto, lo que implica que había opciones reales de que todo el trabajo sirviese a la facultad en un futuro.
- **Mozilla Firefox:** Es un navegador de código abierto, empleado en el proyecto para cargar la página de DOMjudge y depurar su código, haciendo uso de jQuery para ejecutarlo y comprobar su correcto funcionamiento con la consola propia del navegador.

### 3.1.3. Herramientas de coordinación

En este apartado se describen las herramientas utilizadas para la comunicación y la coordinación del trabajo.

- **GitHub:** Es un sistema de repositorios empleado para guardar todos los cambios realizados en un trabajo. Gracias a su sistema de subida y bajada de ficheros, se pudo trabajar de manera independiente, de forma que cada miembro del equipo pudiera ver, acceder y modificar los cambios realizados por el resto de miembros.
- **Discord:** Es una aplicación de mensajería que permite la opción de realizar videollamadas y escribir mensajes en un chat de texto, utilizada durante todo el proyecto como medio de reunión e intercambio de enlaces y archivos en las sesiones de trabajo.
- **Google Meet:** Es una extensión de Google que, al igual que Discord, proporciona un servicio de mensajería y videollamadas. La razón por la que se utiliza este servicio es para la comunicación con los tutores, gracias a la posibilidad de programar llamadas para una fecha y una hora concreta, además que no es necesario instalar nada en el ordenador.
- **Visual Studio Live Share:** Es una extensión de Visual Code mediante la cual se puede editar código fuente de manera colaborativa en tiempo real, permitiendo así una coordinación muy eficiente. Esta extensión esta pensada para que un miembro cree la sesión colaborativa, el anfitrión, y el resto de miembros se unan a dicha sesión como invitados. El problema es que los cambios realizados solo se guardan en los ficheros del anfitrión, por lo que es necesario un repositorio en el que subir dichos ficheros para que los invitados puedan acceder a ellos.

## 3.2. Tecnologías empleadas en DOMJudge

Ya que el proyecto final es nuestro juez en línea DOMJudge implementado con WebAssembly, en este apartado vamos a describir cuales han sido las tecnologías empleadas tanto en el back-end como en el front-end

### 3.2.1. Back-end

En este apartado se describen todas las tecnologías estudiadas y empleadas en el desarrollo del back-end de nuestro proyecto, en este caso la parte del servidor.

- **PHP** Lenguaje de programación empleado fundamentalmente para el desarrollo web. El código fuente de DOMjudge utiliza este lenguaje para programar la

parte del servidor. En este aspecto, será necesaria su integración junto con Node.js, ya que este último es el necesario para desarrollar WebAssembly.

- **Symfony:** Se trata de un framework de PHP usado en el desarrollo de aplicaciones web y, en este caso, empleado por los desarrolladores originales de DOMJudge para la creación del servidor. Se distribuye con licencia MIT lo que permite trabajar sobre él sin problemas de Copyright.
- **Node.js:** Es un entorno de ejecución para JavaScript utilizado para la programación de servidores web. En este proyecto tuvo que ser integrado junto con Symfony, ya que en este último no era posible implementar WebAssembly. La simulación local del juez está programada únicamente con este entorno, por lo que se comprobó que era posible integrar ambas tecnologías.
- **Apache HTTP:** Es un servidor web HTTP de código abierto el cual utilizaron los desarrolladores de DOMJudge en el desarrollo del mismo. Para poder integrar Symfony y Node.js fue necesario realizar unas modificaciones en el fichero de configuración de este servidor como se explicará más detalladamente en el Apéndice A.

### 3.2.2. Front-end

En este apartado se describen todas las tecnologías estudiadas y empleadas en el desarrollo del front-end de nuestro proyecto, en este caso la parte del cliente.

- **JavaScript (Mozilla y individual contributors (2005))** Lenguaje de programación orientado a objetos, basado en prototipos, débilmente tipado. Se utiliza para programar tanto en el lado del cliente como en el del servidor. En este proyecto se utiliza junto con Node.js para desarrollar el código del servidor y, sobre todo, para la programación de WebAssembly en la parte del cliente.
- **Twig:** Es un motor de plantillas para PHP que se encarga de separar el código HTML del código PHP. Al igual que Symfony, los desarrolladores de DOMJudge lo utilizaron para su creación. Es el encargado de proporcionar una manera cómoda de trabajar sobre la parte de la vista, de manera que el código HTML se divide en módulos.
- **CSS:** Es el lenguaje de estilos que se utiliza para definir la presentación de documentos HTML, que se emplea para modificar el estilo de todos los elementos de DOMjudge, tanto los originales como los que se añadieron a raíz de este proyecto, como pueden ser algunos botones o la ventana de resultados.
- **jQuery:** Es una librería de JavaScript que simplifica la interacción con los elementos de HTML y una de las opciones que permite ejecutar código JavaScript en el lado del cliente, lo que en el caso de este TFG permite ejecutar todas las llamadas necesarias para obtener los archivos del juez, el código compilado del

servidor, procesar los resultados obtenidos, compararlos con los esperados y dar una respuesta al usuario.

- **Ajax** Es una tecnología web utilizada para intercambiar datos entre el servidor y el cliente. Se utiliza para poder compilar el archivo C++ en Node.js y pasar el resultado de dicha compilación al cliente para que lo ejecute y compare.

### 3.3. Metodología del trabajo

La metodología utilizada para realizar el trabajo ha sido Metodología Waterfall o Cascada. Si bien es cierto que no se utilizan las mismas fases que en una metodología Waterfall prototípica, la metodología de trabajo utilizada en este proyecto se asemeja a esta debido a que las 3 últimas fases, las de desarrollo, están bien definidas y se realizan en orden descendente, es decir, hasta que no se termina una etapa no se pasa a la siguiente, y una vez que se da una etapa por finalizada no se vuelve a trabajar en ella. Bien es cierto que esto no pasa con la etapa de investigación, la primera, que aunque en los primeros meses se investiga mucho sobre las posibilidades y funcionamiento de WebAssembly, en las otras tres etapas surgen dudas, por lo que la fase de investigación ha estado de manera presente (aunque en mucha menos medida) a lo largo de todo el proceso de desarrollo. Es por esto que la metodología Waterfall es la que más se asemeja a este proyecto, aunque con variaciones significativas.

Esta metodología también tuvo algunos matices propios de las metodologías ágiles, ya que los requisitos y las tareas fueron evolucionando sobre la marcha, estaban bien definidos al principio de cada etapa pero a medida que se avanzaba con el trabajo y surgían los problemas, era necesario cambiar algunas tareas o que los tutores sugiriesen otras formas posibles de realizar dichas tareas.

El flujo de trabajo que se siguió una vez comenzada la fase de desarrollo tenía una estructura en la que, cuando se desarrollaba alguna característica nueva de la aplicación, se procedía a una sesión de control en la que se comunicaba y mostraba a los tutores lo desarrollado y se establecían los siguientes pasos en el proyecto. Si en la reunión se obtenía feedback positivo sobre lo ya desarrollado, se avanzaba en el proyecto y se comenzaba de nuevo con el flujo de trabajo; pero si el feedback no era positivo y había que realizar alguna corrección sobre lo ya desarrollado, se paraba lo nuevo hasta realizar las correcciones necesarias para retomar el trabajo. Una vez terminada la sesión de control, empezaban las sesiones de trabajo. En estas sesiones es donde se avanzaba con el desarrollo del proyecto, bien fuera corrigiendo fallos detectados en la sesión de control o desarrollando nueva funcionalidad. Estas sesiones tenían una duración aproximada de dos a tres horas y una frecuencia de cuatro a seis días a la semana, en función de la carga de trabajo. Una vez terminado un volumen de trabajo considerado, se volvía a realizar otra sesión de control con los tutores para recibir feedback sobre el avance.

Toda el desarrollo del proyecto fue, en su mayoría, de forma colaborativa. Siempre que se trabaja en las etapas de desarrollo se hacía de manera simultánea. Solamente en la redacción y documentación de esta memoria y en alguna ocasión esporádica se dividió el trabajo. De esta forma, se tenía la garantía de que ambos miembros del grupo iban a saber y a controlar las modificaciones que se realizasen.

## Capítulo 4

# Desarrollo del proyecto

**RESUMEN:** En este capítulo se explica con detalle todo el desarrollo del proyecto, empezando por explicar el juez local y terminando con el trabajo realizado sobre DOMjudge.

### 4.1. Juez Local

#### 4.1.1. Arquitectura

Dado que WebAssembly esta pensado para utilizarse en aplicaciones en linea, la arquitectura utilizada fue el modelo cliente-servidor, en la cual el cliente es el que ejecuta el archivo de código fuente en lenguaje C/C++ sin necesidad de mandárselo al servidor, evitando así que el servidor realice el proceso de ejecución.

#### 4.1.2. Desarrollo

Por motivos de simpleza y comprobación, y con el fin de evitar empezar el desarrollo del proyecto directamente sobre el código de DOMjudge, lo que habría supuesto un esfuerzo mayor, se realizó un juez local que emulase la funcionalidad básica de cualquier juez en línea. El objetivo era poder comprobar que el proyecto era viable y establecer una base que sirviese como plantilla a la hora de programar en DOMjudge.

El diseño de esta simulación tiene una funcionalidad común a la de DOMjudge y dispone de elementos para la realización de entregas similares. Consistió en un proyecto bastante simple, en el cual un usuario iniciaba sesión en el juez, como se ve

en la figura 4.1, accedía a la página inicial representada en la figura 4.2 y, mediante un botón de "Subida archivos", se desplegaba la vista de la figura 4.3 en la que se podían subir archivos en código C/C++ y modificar su contenido antes subirlo, como se ve en la figura 4.4, para posteriormente enviarlo al servidor, que realizaba la compilación en WebAssembly. Una vez compilado dicho archivo fuente, el cliente mostraba el resultado de la ejecución en la consola del navegador, con el formato que se ve en la figura 4.5. Los archivos que se empleaban en las subidas eran plantillas de ejercicios reales utilizados en la asignatura TAIS, perteneciente al grado en Ingeniería Software, ya que en esta asignatura se utiliza DOMjudge. De esta forma, se aseguraba que se podía compilar ejercicios reales de DOMjudge.

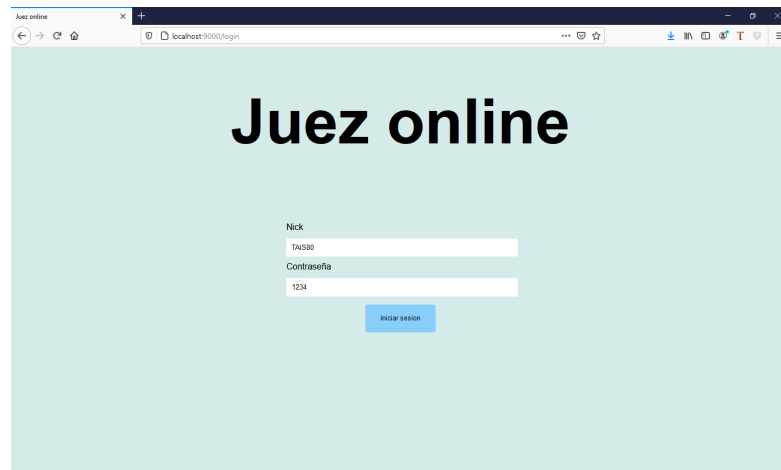


Figura 4.1: Login juez local

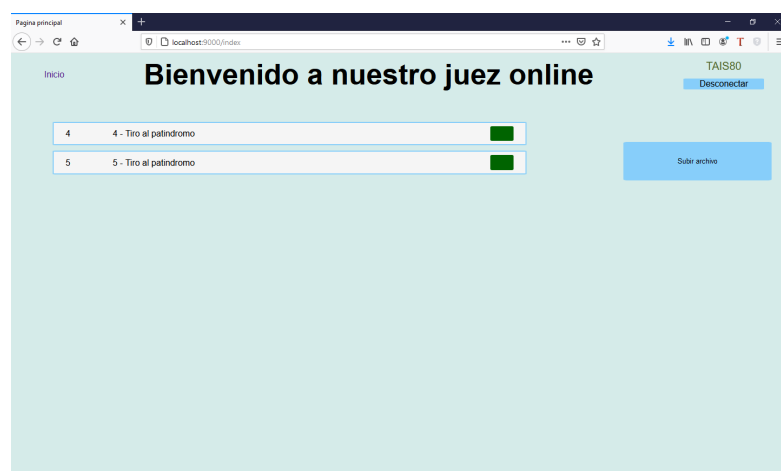


Figura 4.2: Vista principal juez local



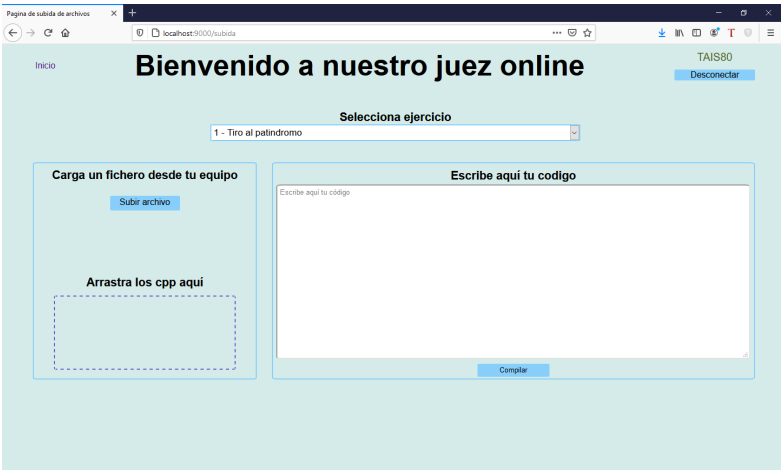


Figura 4.3: Vista subida de archivos juez local

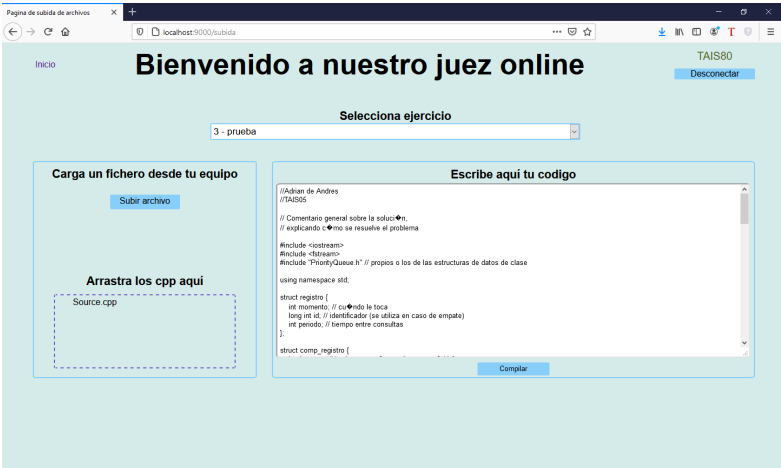


Figura 4.4: Ejemplo subida fichero juez local

1234	TAIS80_3.js:2144:16
9000	TAIS80_3.js:2144:16
1234	2 TAIS80_3.js:2144:16
9000	TAIS80_3.js:2144:16
---	TAIS80_3.js:2144:16

Figura 4.5: Vista subida de archivos juez local

## 4.2. DOMjudge - WebAssembly

### 4.2.1. Arquitectura del proyecto

Como se ha mencionado antes en esta memoria, el objetivo de este proyecto es incorporar un botón que amplíe el funcionamiento de DOMjudge, añadiendo al

posibilidad de realizar una comprobación previa del código subido sobre los samples del problema, lo cual se consigue elaborando una arquitectura como la mostrada en la figura 4.6.

Para ello ha sido necesario introducir las siguientes modificaciones en el front-end y back-end de DOMjudge:

- **Front-end:** Dado que se corresponde con la parte con la que interactúa el usuario, es decir, la página de DOMjudge en su navegador y es desde donde se realiza la subida de los archivos, aquí se añade el botón de test, aprovechando elementos ya existentes en DOMjudge, y que se detallan más adelante en esta memoria. Además, añade un archivo JavaScript, también detallado más adelante, que aprovecha código actual de DOMjudge para que, gracias a jQuery, se realicen peticiones al cliente para obtener los samples del ejercicio actual ejecutarlos uno a uno en el propio navegador, para después mostrar los resultados obtenidos.
- **Back-end:** Se amplía el funcionamiento actual del DOMjudge añadiendo un servicio que permite recibir un archivo C++ y devolver el contenido del archivo compilado con Emscripten a WebAssembly.

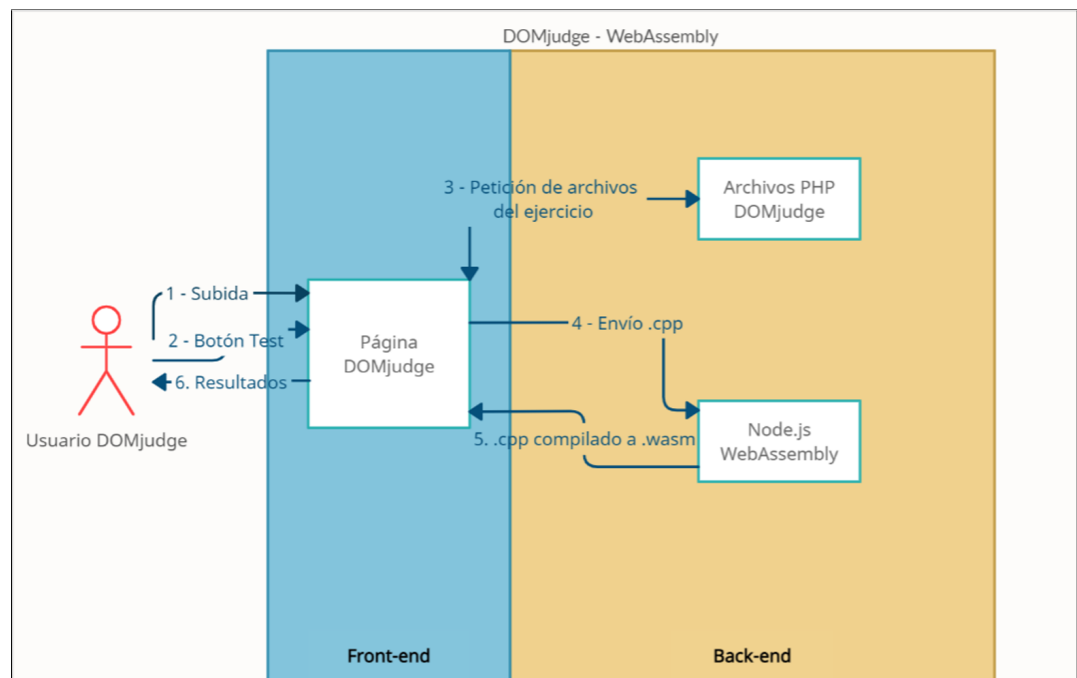


Figura 4.6: Arquitectura proyecto

## 4.2.2. Desarrollo

### 4.2.2.1. Botón ‘Test’

Partiendo de lo desarrollado en el juez local, se estudió cómo trasladar el servidor con WebAssembly a DOMjudge. El resultado fue incluir un botón ‘Test’ en el modal que permite realizar la compilación en WebAssembly de una entrega. Este botón se sitúa al lado del botón ya existente que sirve para realizar una entrega. El resultado final se puede observar en la figura 4.7. Los pasos secuenciales que realiza el botón ‘Test’ se mencionan con un bajo nivel de detalle en el diagrama de la figura 4.8 y se describen en las siguientes subsecciones de este apartado 4.2.2

Figura 4.7: Vista del modal con el botón ‘Test’ añadido

### 4.2.2.2. Obtención y validación de los parámetros de entrada

Una vez el usuario pulsa en el botón de test, se emplea jQuery para deshabilitar los botones del modal encargado de realizar una subida, con el fin de evitar problemas de funcionamiento.

A continuación, se almacenan en variables los distintos campos del modal para poder procesarlos y trabajar con ellos. Dichos campos son los siguientes:

- **Source files:** El programa se encarga de comprobar que se ha seleccionado algún archivo en la casilla correspondiente. En caso de que no se haya seleccionado se mostraría el error de la figura 4.9:
- **Problem:** El programa se encarga de comprobar que se ha seleccionado un

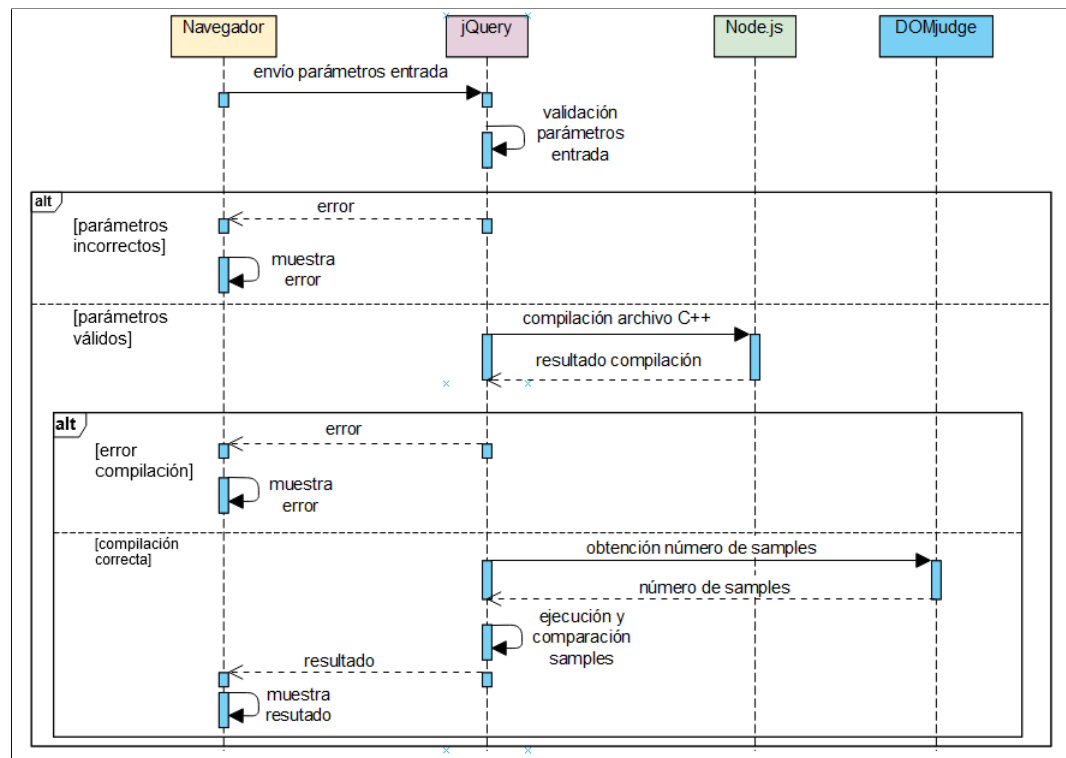


Figura 4.8: Diagrama secuencia función botón ‘Test’

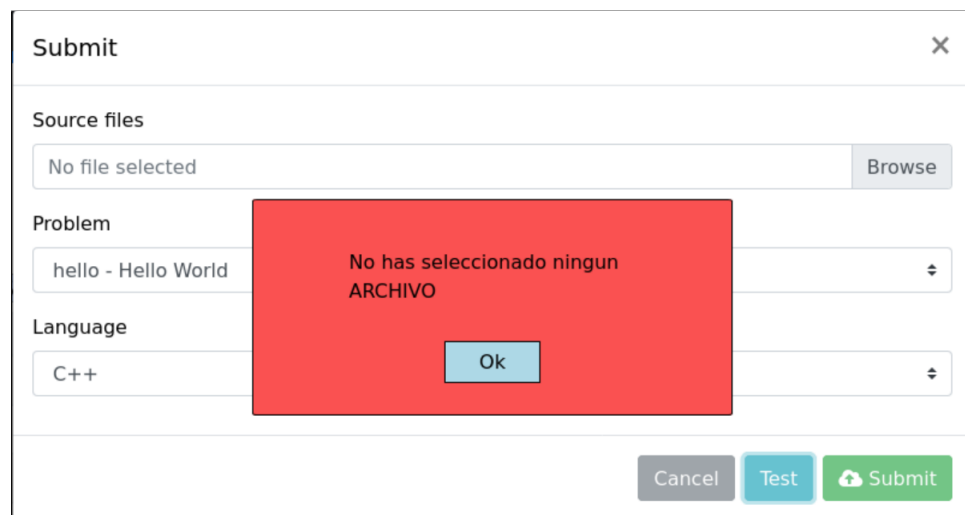


Figura 4.9: Error selección archivo

problema en la casilla correspondiente. En caso de que no se haya seleccionado se mostraría el error de la figura 4.10:

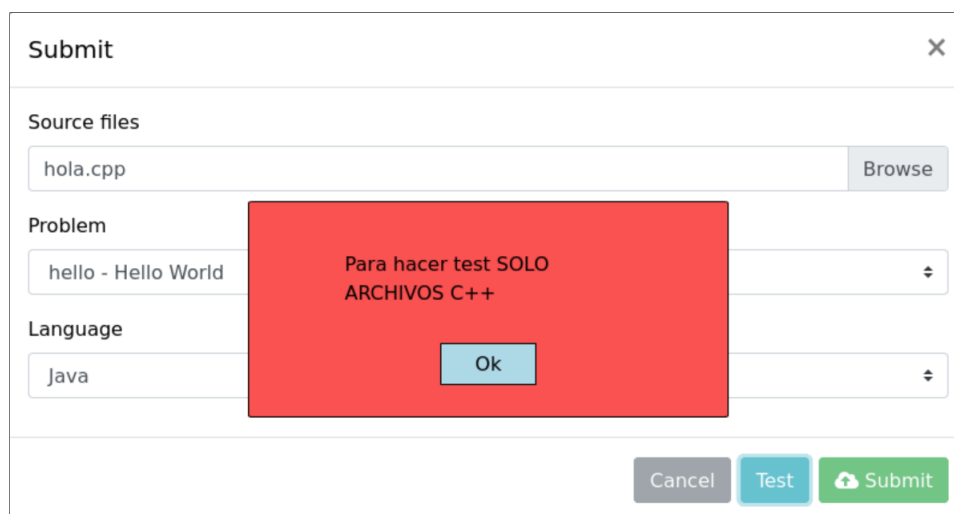
- **Language:** El programa se encarga de comprobar que se ha seleccionado algún



The screenshot shows the 'Submit' form in DOMjudge. The 'Source files' field contains 'hola.cpp'. The 'Problem' dropdown is set to 'Select a problem'. The 'Language' dropdown is set to 'C++'. A red error box is displayed in the center, containing the text 'No has seleccionado ningun PROBLEMA' and an 'Ok' button. At the bottom right, there are three buttons: 'Cancel', 'Test', and 'Submit'.

Figura 4.10: Error selección problema

lenguaje en la casilla correspondiente y que además se corresponde con la extensión .cpp. En caso de que no se cumplirá alguna de las condiciones anteriores se mostraría el error de la figura 4.11



The screenshot shows the 'Submit' form in DOMjudge. The 'Source files' field contains 'hola.cpp'. The 'Problem' dropdown is set to 'hello - Hello World'. The 'Language' dropdown is set to 'Java'. A red error box is displayed in the center, containing the text 'Para hacer test SOLO ARCHIVOS C++' and an 'Ok' button. At the bottom right, there are three buttons: 'Cancel', 'Test', and 'Submit'.

Figura 4.11: Error lenguaje

#### 4.2.2.3. Compilación archivo C++

En la figura 4.12 se muestra un diagrama de secuencia con un nivel de detalle bajo en el que están reflejados los pasos que se siguen en Node.js para realizar la

compilación del archivo C++ subido. A continuación, se explicará dicho diagrama, aunque antes cabe destacar que todas las carpetas y archivos que se creen van a ser temporales y al final de la ejecución se borrarán. El motivo de esto es que Emscripten necesita archivos físicos en el equipo a la hora de realizar la compilación, pero para el resto de la funcionalidad no se necesitan dichos archivos, sino el contenido de ellos.

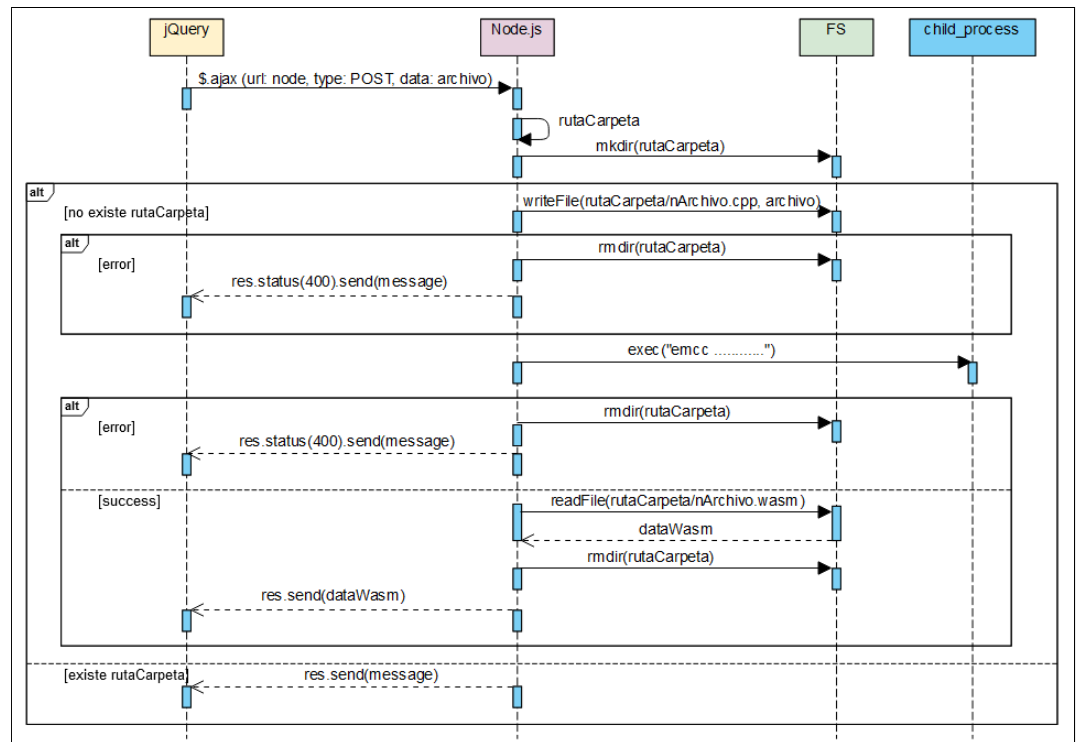


Figura 4.12: Diagrama secuencia compilación archivo C++

Una vez se han obtenido y validado los campos, se procede a enviar el contenido del archivo C++ al servidor en Node.js. Para esto, se hace una llamada Ajax de tipo POST desde jQuery en la cual se manda el contenido de dicho fichero.

El servidor se encarga de compilar a WebAssembly. Para ello, lo primero que hace es establecer una ruta, llamada 'rutaCarpeta', en la que poder crear las carpetas y los archivos temporales.

Para crear dicha carpeta temporal, se hace uso del módulo *FS* de Node.js para poder utilizar la función *mkdir*, especificándole la ruta que hemos definido antes. Si la ruta ya existiese (que nunca debería ser el caso), se devuelve una respuesta al navegador con un mensaje de error. Si no existiese, se procedería a crear el fichero temporal.

Este fichero temporal es un archivo C++ con extensión `.cpp` en el que se escribe el

contenido del archivo subido, es decir, una copia de lo que el usuario sube al juez. Este archivo se crea dentro de la carpeta temporal mediante la función *writeFile*, pasándole el nombre del archivo que tendrá el archivo temporal, llamado 'nArchivo.cpp', y el contenido del archivo C++ original. Si la escritura fallase por cualquier motivo, se borraría la carpeta temporal haciendo uso de la función *rmdir* y se devolvería al servidor una respuesta con estado 400 y un mensaje de error. Si se escribiese bien, lo siguiente que habría que hacer es ejecutar el comando `emcc .....` de Emscripten para compilar a WebAssembly.

Para poder compilarlo en el servidor necesitaremos hacer uso de la función *exec* del módulo *child\_process*. Si la función produjese algún tipo de error, se eliminaría la carpeta temporal con todos los archivos temporales que hubiese dentro y se devolvería al cliente una respuesta con estado 400 y un mensaje de error. Si la función produjera un *stderr* se seguiría con la compilación y se mostraría el error en la consola, ya que este error está asociado a los warnings generado por el código C++, por lo que no son errores que impidan la compilación y ejecución del código.

Si no se ha producido ningún error en la compilación, haya habido o no warnings, se lee el contenido del archivo compilado en formato *.wasm*, llamado 'nArchivo.wasm', haciendo uso de la función *readFile* de *FS*. El contenido de este archivo se llamará en una variable llamada 'dataWasm'. Una vez leído el archivo *.wasm*, se procede a eliminar la carpeta temporal y todos los archivos que haya dentro y se devuelve al cliente una respuesta con el contenido de la variable 'dataWasm'. En este punto, ya habría terminado la compilación del fichero C++ a formato WebAssembly y se pasaría al siguiente apartado.

Mientras que se realiza todo el proceso de compilación, en el navegador se muestra una ventana como la de la figura 4.13 con un mensaje para que el usuario sepa que ya ha empezado la compilación.

Si al compilarlo se produce un error de compilación, dicha ventana aparecerá de color rojo con un mensaje de error, como se ve en la figura 4.14.

#### 4.2.2.4. Obtención del número de samples/ficheros de prueba

Una vez el proceso de compilación ha terminado con éxito, se obtiene el número de samples públicos que hay para el problema seleccionado.

Los samples públicos se encuentran en una ruta propia de DOMjudge accesible desde el navegador. Esta ruta es la siguiente:

`/domjudge/team/<idProblema>/sample/<nActual>/input`

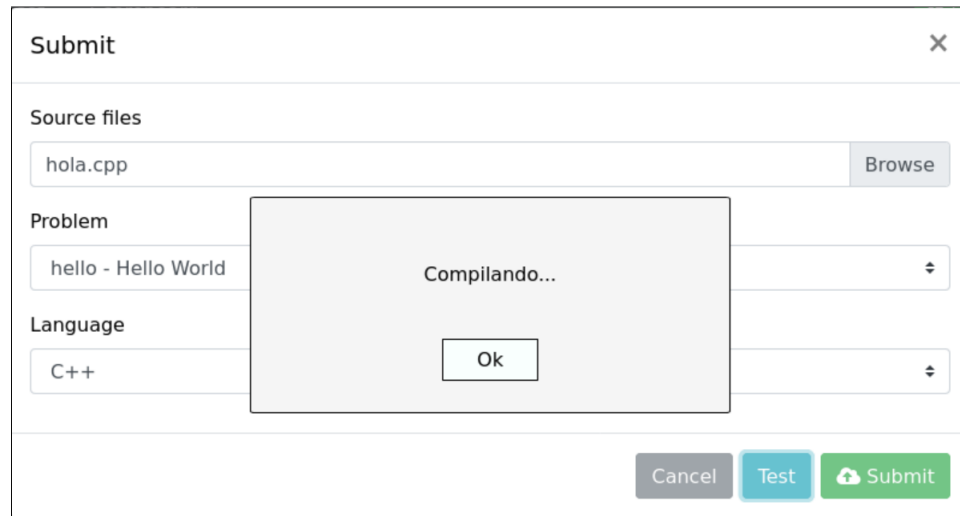


Figura 4.13: Mensaje para que el usuario sepa que ha empezado la compilación



Figura 4.14: Error en el proceso de compilación

Como se puede apreciar, tiene 2 parámetros variables: `<idProblema>`, un número que funciona de identificador para el problema seleccionado; y `<nActual>`, un número que corresponde a un sample público determinado para ese problema. `<nActual>` toma valores consecutivos, es decir, si el sample número cinco no existiese se asume que no va a existir ningún sample posterior a él.

Sabiendo esto, el recurso utilizado para averiguar cuantos samples públicos hay es con peticiones de Ajax de tipo HEAD. Las llamadas de tipo HEAD funcionan de la siguiente manera: si la ruta existe, es decir es alcanzable, devuelve un valor indicando



que si es alcanzable, pero si no existe, devuelve un error. Teniendo en cuenta esto y que los samples públicos son consecutivos, la mejor forma de calcular el número de samples es mediante un bucle en el que se realicen de forma asíncrona llamadas de tipo HEAD consecutivas, aumentando en cada iteración el valor de `<nActual>` una unidad y guardando en una variable un contador con el número de samples públicos que se van encontrando. Se realizan llamadas asíncronas para asegurar que solamente se continúan con las llamadas si la anterior no devolvió un error. En el momento en que una llamada devuelve error, quiere decir que no va a haber mas samples públicos para ese ejercicio.

#### 4.2.2.5. Ejecución y comprobación de samples/ficheros de prueba

Una vez obtenido el número de samples que hay para un determinado problema, se ejecuta el código en formato WebAssembly. En esta parte se emplean llamadas a funciones asíncronas acompañadas de pausas en la ejecución (`async-await`). El motivo por el cual se utilizan llamadas asíncronas es acelerar el proceso, de manera que todas las funciones necesarias empiezan a ejecutarse al mismo tiempo. Sin embargo, hay funciones que dependen del resultado de otra, por lo que es necesario introducir sentencias `await` para asegurarse de que no empiezan su ejecución antes de tiempo.

Para implementar la función de este apartado se emplea un bucle con tantas iteraciones como número de samples haya. En cada iteración del bucle se realizan 4 funciones: la primera, encargada de obtener el sample de salida; la segunda, encargada de obtener el sample de entrada; la tercera, encargada de la ejecución del código; y la cuarta, encargada de la comparación de resultados.

##### Obtención sample output

Mediante una llamada Ajax de tipo GET se obtiene el fichero de salida esperado para el sample que estamos analizando. Como ya se ha explicado en el apartado 4.2.2.4, la ruta de la llamada siempre es la misma, lo que varían son los parámetros `<idProblema>` y `<sampleActual>`, que corresponde al número del sample público que se está comprobando.

##### Obtención sample input

Mediante una llamada Ajax de tipo GET se obtiene el fichero de entrada necesario para el sample que se esta analizando. En este caso, al igual que en la obtención del output, la ruta no varía y los parámetros `<idProblema>` y `<sampleActual>` corresponden a los mismos datos.

##### Ejecución archivo .wasm

Esta parte es el centro de toda la funcionalidad WebAssembly, la parte en la que

más problemas han surgido y en la que se ha empleado más tiempo. Es por ello que probablemente sea el apartado más largo y con más conceptos técnicos.

A modo de apunte, recordemos que la funcionalidad del fichero C++ subido se ha guardado y transformado en un fichero en binario `.wasm`. Esto es importante, ya que habrá numerosas referencias al fichero `.wasm`.

Cuando se compila un trozo de código de C/C++ con su main a WebAssembly, el `.wasm` incluye las librerías estándar de C. Además, al lanzar el `.wasm`, hay que crear un `.entorno de ejecución` para que se pueda ejecutar de manera segura. El comando `'emcc'` genera automáticamente el fichero `.wasm` y un fichero JavaScript. Cabe destacar que dicho fichero `.js` autogenerado consta de, aproximadamente, unas cinco mil líneas, en las cuales está toda la funcionalidad necesaria para ejecutar el contenido del fichero `.wasm`. Ahora bien, este fichero es una plantilla de ejemplo que permite una integración fácil en un HTML simple, por lo que eran necesarias varias modificaciones importantes acorde a las necesidades del proyecto. Este fichero, depurado hasta dejar únicamente lo necesario, fue modificado para añadirle todo la funcionalidad principal y se le otorgó el nombre de `test_WebAssembly.js`.

A continuación, se exponen algunos de los cambios mas significativos que realizaron:

- El primer gran cambio fue la modificación de la función llamada `'createWasm'`, debido a que ocasiona errores a la hora de instanciar el módulo WebAssembly, utilizando únicamente el contenido compilado, del archivo C++, a `.wasm`, sin usar un archivo que esté presente en el disco duro. Para ello, se modificó el comportamiento habitual del código para que recibiese este contenido compilado y se encargase de instanciarlo directamente.
- WebAssembly, por defecto, en el fichero `.wasm` autogenerado, transforma automáticamente el `'cout'` de C++ en `'console.log()'` de JavaScript, por lo que el resultado de la ejecución se muestra siempre por la consola del navegador. Para llevar a cabo esto, el módulo de WebAssembly utiliza una función propia llamada `'print'` encargada de dicha transformación. Pues bien, para conseguir que el resultado de la ejecución se guardase en un fichero de texto fue necesario sobrescribir dicho método `print`.
- WebAssembly está desarrollado de tal forma que, por motivos de seguridad, no trabaja con ficheros de texto de tu equipo durante el proceso de ejecución, si no que utiliza su propio sistema de ficheros temporales. WebAssembly crea un sandbox seguro con su sistema de ficheros temporales, de manera que el código escrito en C++ y compilado a WebAssembly va a trabajar únicamente con los archivos de texto pertenecientes al sandbox. Cabe destacar que debido a ese sandbox es imposible que los cambios realizados en un fichero de salida se vean reflejados en el fichero físico del equipo, ya que dichos ficheros son temporales y al final de la ejecución se eliminan.

Si se añade el flag ‘`--embed-files <archivo_texto>`’ al comando ‘`emcc`’, en el sistema de ficheros temporal del sandbox se va a crear un archivo llamado `<archivo_texto>` con su contenido original. Esta solución solo serviría en el caso de que hubiese un archivo de texto.

DOMjudge no tiene un único archivo de texto por ejercicio, sino que puede tener varios archivos de texto (samples), uno para cada caso de entrada (input). Si se utilizara el flag, solo va a existir un fichero de entrada en el sandbox, por lo que no sería posible la comprobación de todos los casos de entrada que tenga dicho problema.

El fichero JavaScript autogenerado contiene un módulo WebAssembly, que a su vez trabaja con distintos módulos para realizar determinadas funciones. Uno de esos módulos se llama ‘FS’ y es el encargado de crear todo el sistema de ficheros temporales necesario para el sandbox. En ese módulo se puede añadir directamente un fichero al sandbox, sin necesidad de utilizar el flag ‘`--embed-files`’.

Añadir los ficheros al sandbox se realiza de la siguiente forma: para cada sample de entrada (input) se crea en el directorio ‘`/tmp/<nArch>/`’ un fichero de texto con el contenido de la entrada de dicho sample, donde `<nArch>` es el nombre del directorio temporal creado durante el proceso de compilación en Node.js (apartado 4.2.2.3). Mediante el módulo ‘FS’ se crea en el sandbox una carpeta llamada ‘`/tmp/<nArch>/`’ con el fichero de texto que contiene la entrada de dicho sample. De esta forma, en cada iteración del bucle, en el sandbox se sobrescribe el fichero de texto con el contenido de la entrada, permitiendo la comprobación de todos los samples de un ejercicio habiendo compilado el código C++ a WebAssembly una sola vez.

### Comparación salida obtenida con salida esperada

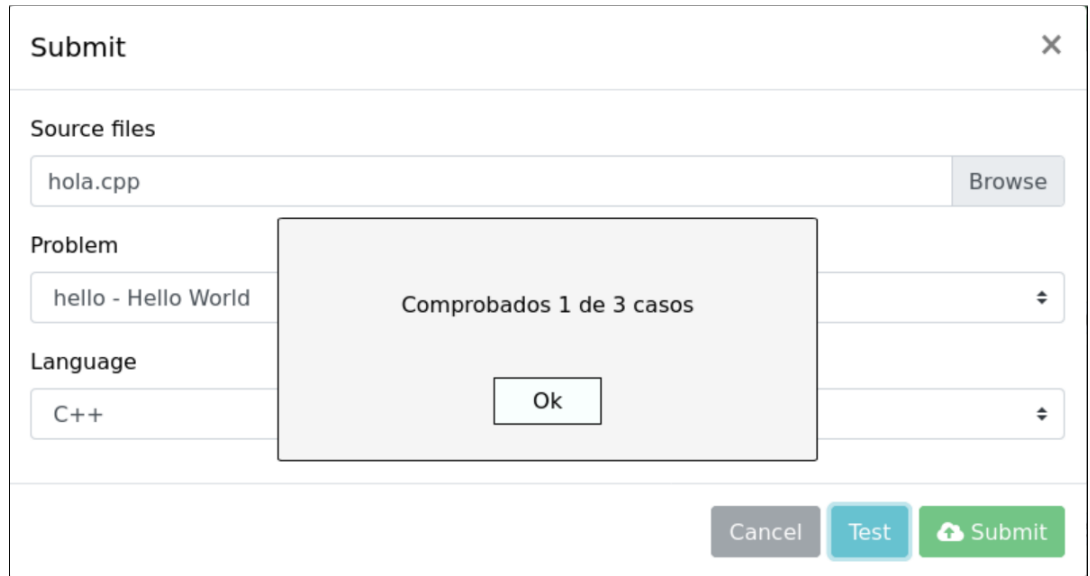
Una vez se ha ejecutado el código compilado, se obtiene una salida, cuyo contenido se va a comparar con el de la salida esperada, obtenido con la función encargada de obtener el sample output, para saber si es correcto o no.

Este proceso se llevará a cabo mediante una función que recibe como parámetros de entrada dos arrays: uno con la salida del código subido y otro con lo descargado de los samples del juez. Es una función simple que realiza comprobaciones habitualmente empleadas en la comparación de dos arrays, comenzando por el tamaño de ambos, que, si no coincide, se deduce que la respuesta dada es errónea.

Si pasa esta primera prueba, se comienza a comparar el contenido de ambos arrays hasta llegar al final, si alguna posición no coincide, la respuesta es errónea, pero, si llega hasta el final sin fallo, la respuesta es correcta.

La función devuelve un booleano al final de su ejecución: *false* en caso de que haya algún error y *true* en caso de que la respuesta sea correcta.

Durante todo el tiempo que se esta comparando la salida, el navegador muestra en la ventana, el progreso que lleva sobre los casos comparados, indicando cuantos se han comparado sobre el total de casos a comparar, como se muestra en la figura 4.15.



The image shows a web interface for submitting code. The main window is titled "Submit" and contains the following elements:

- Source files:** A text input field containing "hola.cpp" and a "Browse" button.
- Problem:** A dropdown menu showing "hello - Hello World".
- Language:** A dropdown menu showing "C++".
- Buttons:** "Cancel", "Test", and "Submit" (with a cloud icon).

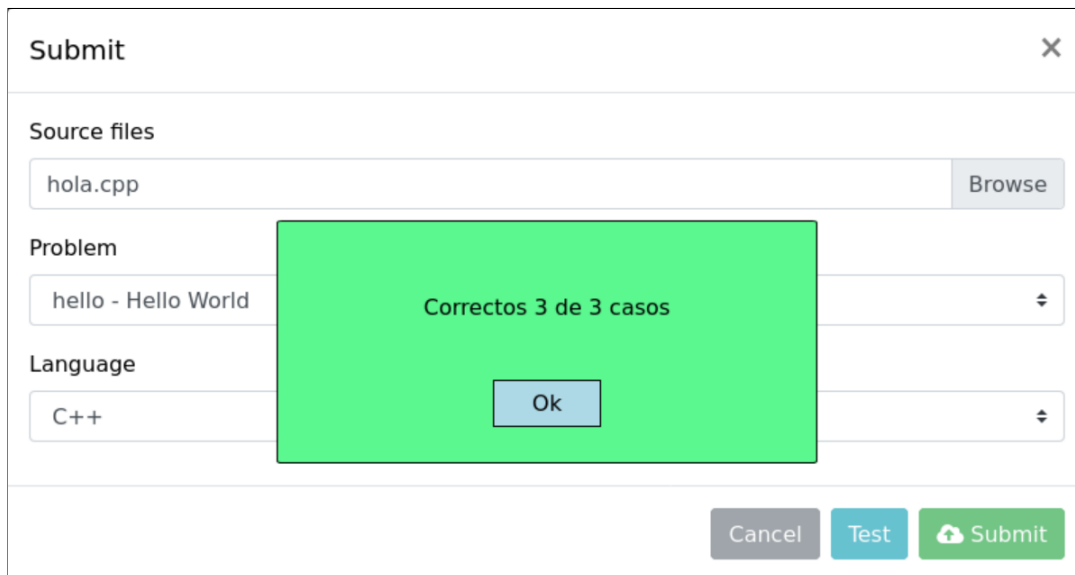
A modal dialog box is overlaid in the center of the window. It has a light gray background and contains the text "Comprobados 1 de 3 casos" (Checked 1 of 3 cases) and an "Ok" button.

Figura 4.15: Función para comparar la salida obtenida con la esperada

Una vez acaba la comparación, la ventana de salida mostrará el número de casos correctos respecto al total de samples, por lo que aquí surgen dos posibilidades.

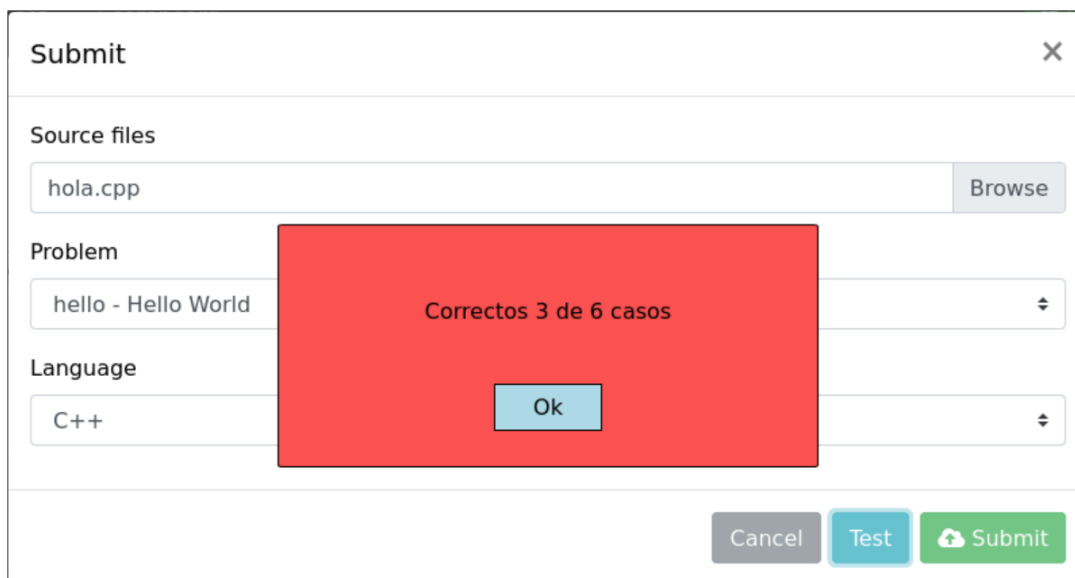
La primera de las posibilidades, es en la que todos los casos comprobados son correctos, y la ventana de salida cambia su color de fondo a un tono verde, para indicar la validez de la respuesta, como se muestra en la figura 4.16.

La otra de las posibilidades se da cuando se ha producido algún error en la salida de los casos de prueba, pudiendo ser solo uno o incluso todos, y en la que la ventana cambia su color de fondo a un tono rojo, como se muestra en la figura 4.17.



The image shows a 'Submit' dialog box from DOMjudge. It has a title bar with a close button (X). The dialog contains three main sections: 'Source files' with a text input 'hola.cpp' and a 'Browse' button; 'Problem' with a dropdown menu showing 'hello - Hello World'; and 'Language' with a dropdown menu showing 'C++'. A large green modal box is centered over the dialog, displaying the text 'Correctos 3 de 3 casos' and an 'Ok' button. At the bottom right of the dialog are three buttons: 'Cancel', 'Test', and 'Submit' (which has a cloud icon).

Figura 4.16: Salida correcta



The image shows a 'Submit' dialog box from DOMjudge, similar to the one in Figure 4.16. It has the same layout: 'Source files' with 'hola.cpp' and 'Browse', 'Problem' with 'hello - Hello World', and 'Language' with 'C++'. However, a large red modal box is centered over the dialog, displaying the text 'Correctos 3 de 6 casos' and an 'Ok' button. The bottom right buttons 'Cancel', 'Test', and 'Submit' are also present.

Figura 4.17: Salida errónea



## Capítulo 5

# Resultados, conclusiones y trabajo futuro

**RESUMEN:** En este apartado se detallan los resultados y conclusiones obtenidas, además del trabajo futuro que se podría realizar sobre este proyecto.

### 5.1. Resultados y conclusiones

Se alcanzó el objetivo final del proyecto, que era conseguir añadir a un juez en línea un sistema para ejecutar en el navegador del usuario una serie de pruebas simples, haciendo uso de WebAssembly, para evitar subidas innecesarias.

Por ello, el desarrollo del proyecto consta de dos partes: una, en la que se investiga y desarrolla un pequeño juez local, que utiliza en su servidor la tecnología de WebAssembly, y con lo que se constata la posibilidad de desarrollar ese sistema de pistas ligeras sobre las entregas, haciendo uso de esta tecnología; y una segunda, en la que se escala lo implementado para el juez local, y se adapta a un juez en línea ya desarrollado, generando una solución que puede ser llevada a otros jueces en línea.

### 5.2. Trabajo futuro

Debido a lo que se ha comentado en las conclusiones, esta forma de integrar WebAssembly es trasladable al resto de jueces en línea, ya que el servidor hecho con Node.js, que compila a WebAssembly, está fuera del propio juez, y que, el archivo JavaScript con la ejecución del código compilado, que a su vez usa jQuery, tiene una

código que se puede modificar fácilmente para darle otros funcionamientos, según el juez al que se traslade. La única salvedad podría ser añadir los botones nuevos en el código del juez, dado que la estructura podría no ser la misma, pero mientras se modifiquen apropiadamente, todo el código sería reutilizable.

El principal objetivo a desarrollar en un futuro sería la integración de la funcionalidad del botón que hemos implementado en el flujo de entregas actual del juez. De esta forma, añadiríamos nuestro trabajo al proceso habitual, de manera que si la entrega subida no supera los samples de prueba para el ejercicio, se muestra la ventana del navegador indicando el número de errores y se detiene el proceso de entrega. Pero si se superan con éxito los samples se continúa de forma natural la entrega y se envía al servidor.

El estado actual del proyecto no permite compilar problemas complejos que empleen librerías o archivos de cabecera adicionales, que son empleados en asignaturas como FAL o TAIS, dado que funciona generando carpetas y archivos temporales durante las compilaciones. A raíz de esto, un trabajo a futuro, sería conseguir cargar las librerías necesarias en el directorio temporal, que crea el propio código de pegamento de WebAssembly, para poder añadirlos a los archivos temporales de forma que pueda acceder a ellos en el proceso de ejecución.

Un trabajo a futuro podría ser preparar el código para que sirviese para compilar cualquier lenguaje de programación de los que acepta WebAssembly, ya que actualmente solo está pensado para funcionar con C++. Es por esto que si en el desplegable de lenguaje no seleccionamos C++, salta un mensaje de error. El trabajo a realizar sería adaptarlo para compilar también en C y Rust, los otros 2 lenguajes aceptados por WebAssembly. Esto sería muy útil para cualquier juez en línea que permita la subida de ejercicios en alguno de estos lenguajes. DOMjudge tiene la opción de realizar entregas en C, por lo que sería interesante implementarlo para usarlo en la facultad, ya que en algunas asignaturas también se admiten entregas en C, por ejemplo.

Otro trabajo a futuro podría ser profundizar en el tipo de pistas que da, y estudiar los tiempos de ejecución y uso de memoria de las funciones implementadas en el código entregado, de tal forma que no solo si darían pistas sobre la salida generada, si no que además se podrían dar a su vez pistas sobre como optimizar esas funciones. Para realizar esto, una posible solución sería tener para cada problema unos resultados previos obtenidos realizando una entrega con código correcto de tal forma que se puedan comparar las entregas que realizan los alumnos con esta y dar las pistas en función de los resultados que generan.

Una funcionalidad que puede resultar muy útil, puede ser adaptar, de lo desarrollado para el juez local, la posibilidad de modificar el código en el propio navegador, incorporando al juez, un área de texto que se rellena con el contenido del .cpp subido o con código escrito directamente ahí, de forma que, cuando se pulsa el botón de



---

test y se recibe información sobre los resultados, se puedan introducir correcciones sin acudir al IDE, o similar, en el que se haya trabajado ese código.



## Capítulo 6

# Aportaciones individuales

### 6.1. Adrián De Andrés Alonso

Mi aportación individual a este trabajo ha sido la siguiente:

- Durante la etapa de investigación, debido a las asignaturas que por el momento cursaba, experimenté con tecnologías de desarrollo web, como Node.js, HTML, CSS, JavaScript y jQuery. Pero sobre todo, investigué acerca de WebAssembly, qué era, que posibilidades tenía y cómo utilizarlo. De manera implícita, también investigué sobre Emscripten, ya que es lo necesario para hacer que WebAssembly funcione.
- Después vino un prototipo local de juez en línea, el cuál utilizamos para hacer ensayos de compilar a través de WebAssembly subidas de códigos. En esta parte colaboré en el desarrollo de la página web en sí, creando el servidor en Node.js, en el cual integramos WebAssembly y comprobamos que realmente sí se podía trasladar a un juez en línea real. Además, participé en el desarrollo de front-end hecho con HTML y CSS, y en la creación de la base de datos montada con SQL.
- Seguido al juez local, empezamos a trabajar con DOMjudge. Para esto, fue necesaria la instalación de Debian y la modificación de algún archivo del sistema. Una vez hecho esto, colaboré investigando cómo instalar DOMjudge y como poder solventar los errores de instalación que nos iban surgiendo. Con el DOMjudge ya instalado, investigué como estaba programada para saber en que parte de todo el código de DOMjudge se realizaba la subida del fichero al servidor y como lo hacía. Para ello, tuvimos que informarnos sobre PHP, twig y Symfony, ya que esta programado con estas tecnologías. También colabore investigando sobre las posibilidades administrativas de DOMjudge, para así poder crear un usuario y un ejercicio de prueba con los que poder probar los cambios que fuéramos realizando.

- En el desarrollo final del proyecto colaboré en la creación de todo lo necesario para el funcionamiento: adición del botón de test al modal de las subidas, creación del div que muestra los resultados y desarrollar el archivo `test_WebAssembly.js`, que es el encargado de toda la compilación de WebAssembly. En este fichero, colaboré en la investigación de como realizar peticiones a servidor en Node.js para realizar la compilación con Emscripten, el manejo de funciones síncronas y asíncronas, cómo adaptar el fichero JavaScript autogenerado por Emscripten para usarlo en nuestro proyecto y en la programación en general del código de este archivo. Del servidor desarrollado con Node.js, colaboré en su programación en general, además de investigar como crear directorios y archivos temporales para las subidas, cómo ejecutar Emscripten, y cómo enviar el fichero compilado a `.wasm` en la respuesta a la petición realizada, habiéndolo leído previamente y enviándolo en un formato concreto para que pudiese ser procesado después. También participé en la comprobación de que todo lo programado funcionase, utilizando Mozilla Firefox como cliente.
- Para la redacción de la memoria, colaboré en la decisión de la herramienta a utilizar, y, una vez elegido LaTeX, cómo empezar a trabajar usando Overleaf. Colaboré redactando todos los puntos, ya sea escribiendo apartados de los capítulo o revisando lo que mi compañero redactaba, para así asegurarnos que no nos olvidaba nada. También busqué referencias bibliográficas y realicé capturas de pantalla necesarias para aportar mas información

## 6.2. Raúl Benavente Romero

Para el desarrollo del TFG, mi aportación individual ha sido la siguiente:

- Durante las primeras etapas, las consideradas de investigación, me dediqué a estudiar sobre tecnologías web en general, como el modelo cliente-servidor, Node.js para poder montar un servidor, HTML y css para desarrollar una interfaz con la que trabajar con ese servidor y JavaScript y jQuery para saber programar el código del servidor y ser capaz de hacer una página web que fuera responsive. Pero sobre todo, sobre WebAssembly, cómo surge y cómo utilizarlo. Lo cuál derivó en tener que investigar sobre Emscripten para poder saber cómo compilar los `.cpp` y qué hacer después con ellos.
- Después de la fase de investigación realizamos un prototipo de juez en línea, el cuál utilizamos para hacer ensayos sobre compilar subidas de códigos a jueces y procesamiento de la respuesta. En esta parte colaboré en el desarrollo de la página web en sí, escribiendo HTML y css, también en el desarrollo de la base datos SQL utilizada y en el desarrollo del servidor, creado con Node.js, en el que participé en descubrir cómo integrar correctamente Emscripten a este servidor haciendo uso de los módulos de Node.js.

- Cuando empezamos a trabajar con DOMjudge, colaboré investigando sobre cómo instalarlo y en qué sistema operativo era mejor. Una vez elegido Debian, tuve que investigar sobre cómo instalar DOMjudge correctamente, ya que había errores de instalación, y, posteriormente, sobre qué archivos de configuración del sistema operativo era necesario modificar, como Apache2. Con el DOMjudge ya levantado, colaboré en descubrir cuál es el proceso de subida de archivos, trazando un flujo a través, del código HTML y PHP que compone el juez, investigando sobre twig y Symfony, determinando los elementos HTML usados y en qué archivos twig se encuentran, que archivos css le daban forma a la página y las funciones empleadas por el código y en qué archivos PHP se encuentran. También colabore investigando sobre como cargar problemas al juez, con inputs y outputs propios, y cómo marcar samples para cada ejercicio, como añadir usuarios adicionales y como crear nuevos concursos.
- En el desarrollo final del proyecto colaboré en la adición de los elementos HTML necesarios para el proyecto: adición del botón de test al modal de las subidas, creación del div que muestra los resultados e investigar sobre donde escribir correctamente el código que permite importar el código que importa nuestro archivo `test_WebAssembly.js`. También colaboré en el trabajo sobre el css para darle forma al div que usamos para mostrar los resultados. Colaboré en el desarrollo del código del archivo `test_WebAssembly.js`, investigando como realizar peticiones a servidor, entre las que se incluye la del nuestro y cómo procesar el archivo compilado una vez recibido, el manejo síncrono de las mismas, cómo adaptar el `.js` autogenerado por Emscripten para usarlo en nuestro proyecto, cómo reutilizar código del juez para obtener los archivos necesarios para cada ejercicio, y en la programación en general del código de este archivo. Del servidor desarrollado con Node.js, colaboré en su programación en general, además de investigar como crear directorios y archivos temporales para las subidas, cómo ejecutar Emscripten, y cómo enviar el fichero compilado a `.wasm` en la respuesta a la petición realizada, habiéndolo leído previamente y enviándolo en un formato concreto para que pudiese ser procesado después. En general, también participé en la fase de debugeo del código, utilizando como herramienta Mozilla Firefox y su consola, investigando como utilizarla y como mostrar los datos en ella para poder obtener conclusiones sobre el funcionamiento del código programado.
- Para la redacción de la memoria, empecé colaborando en la investigación sobre qué herramienta utilizar, y, una vez elegido LaTeX, cómo empezar a trabajar usando Overleaf. De la redacción del texto que incluye la memoria colaboré en todos los puntos escribiendo los apartados de cada capítulo que me correspondían, buscando referencias bibliográficas, realizando capturas del código programada y buscando cómo se usan en LaTeX algunos elementos básicos como listas o insertar dichas imágenes.



Parte I

Apéndices





## Apéndice A

# Guía de instalación

**RESUMEN:** En este apéndice se describe el proceso de instalación y configuración de todos los archivos necesarios para el funcionamiento del proyecto.

### A.1. DOMjudge

- El primer paso es instalar DOMjudge en nuestro equipo/servidor desde el siguiente enlace: [DOMjudge](#)
- Sin embargo, para instalarlo, no utilizaremos el contenido del repositorio oficial de DOMjudge, si no el contenido de nuestro repositorio. Para ello clonamos o descargamos el repositorio **DOMjudge-WebAssembly** y, situándonos en la carpeta `domjudge-7.3.3`, seguimos el proceso de instalación de DOMserver. Con esto tendremos instalado DOMserver con los archivos, ya modificados, necesarios para el funcionamiento de nuestro proyecto. Cabe destacar que, a la hora de instalar DOMserver, no se hiciera dentro de la carpeta `domjudge-7.3.3`, si no que se hiciera a su mismo nivel.
- Para probar nuestro proyecto, solo es necesaria la instalación del DOMserver. Sin embargo, también sería posible la instalación de `judgehosts` y demás elementos que se consideren necesarios.
- Nuestro objetivo es tener una estructura de carpetas similar a la de la figura A.1 una vez terminado todo el proceso de instalación. Las carpetas que nos interesan son `domserver` y `WebAssembly`. Solo en el caso de haber añadido `judgehosts`, aparecerán las carpetas `judgehost` y `lib`, como se ve en la figura A.2.

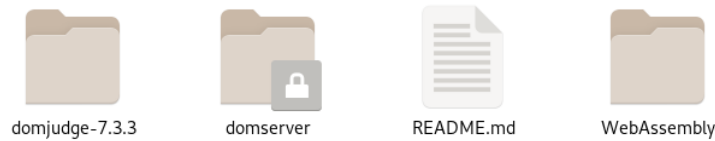


Figura A.1: Carpetas DOMserver



Figura A.2: Carpetas DOMserver y judgehosts

## A.2. Emscripten y WebAssembly

- Si la carpeta domserver y WebAssembly no estuvieran al mismo nivel, copiamos la carpeta WebAssembly que hemos traído de nuestro repositorio, dentro de la carpeta donde hemos instalado DOMjudge, a la altura de domserver.
- Para instalar Emscripten, nos situamos dentro de la carpeta WebAssembly y seguimos el proceso descrito en este enlace: [Emscripten](#). Importante ver el punto siguiente.
- Durante la instalación de Emscripten, cuando ejecutes los comandos './emsdk', aparecerá un mensaje en la consola del estilo de la figura A.3. En el ultimo guión de la sección 'Next steps', aparece una línea en la que se explica el comando a realizar para configurar Emscripten en el equipo de manera que cada vez que se encienda, no sea necesario volverlo a instalar. Si no ejecutamos este comando, va a ser necesario realizar la instalación cada vez que abramos o cerremos un terminal para arrancar el proyecto.

```
Next steps:
- To conveniently access emsdk tools from the command line,
  consider adding the following directories to your PATH:
    /home/adri/DOMjudge-WebAssembly-main/WebAssembly/emsdk
    /home/adri/DOMjudge-WebAssembly-main/WebAssembly/emsdk/node/14.15.5_64bit/bin
    /home/adri/DOMjudge-WebAssembly-main/WebAssembly/emsdk/upstream/emscripten
- This can be done for the current shell by running:
  source "/home/adri/DOMjudge-WebAssembly-main/WebAssembly/emsdk/emsdk_env.sh"
- Configure emsdk in your shell startup scripts by running:
  echo 'source "/home/adri/DOMjudge-WebAssembly-main/WebAssembly/emsdk/emsdk_env.sh"' >> $HOME/.bash_profile
```

Figura A.3: Mensaje instalación Emscripten

- El objetivo es que nuestra carpeta WebAssembly tenga la estructura de la figura A.4.



Figura A.4: Carpeta WebAssembly

## A.3. Apache2

Habilitamos el módulo del proxy con el comando:

```
a2enmod proxy proxy_http
```

Modificar la configuración de Apache2, generalmente situada en el archivo `./etc/c/apache2/apache2.conf` para poder utilizar un proxy, de tal forma que permita dirigir las llamadas al puerto 3000 del localhost al servidor en el que tendremos montado todo el funcionamiento, con node.js, del compilador Emscripten. Para ello añadimos en dicho archivo el contenido de la figura A.5.

```
#ServerRoot "/etc/apache2"  
ProxyPass /node http://localhost:3000  
ProxyPassReverse /node http://localhost:3000
```

Figura A.5: Modificación apache2

Después ejecutamos:

```
systemctl restart apache2
```

## A.4. Arrancar DOMjudge

- Antes de arrancar DOMjudge, hay que asegurarse de tener instalado el paquete npm y Node.js en la carpeta WebAssembly. Para ello, ejecutamos los siguientes comandos:

```
sudo apt-get install npm
```

```
npm install node
```

- Una vez instalados, abrimos la carpeta WebAssembly en un terminal y ejecutamos el comando:

```
node index.js
```

En el archivo `index.js` es donde se encuentra toda la información necesaria para arrancar Node.js.

- En este punto ya tendríamos arrancado Node.js. Es muy importante haber instalado de manera Emscripten, ya que de lo contrario no va a ser posible la compilación a WebAssembly

## Apéndice B

# Archivos modificados DOMjudge

**RESUMEN:** En este apéndice se detallan las modificaciones mas relevantes y todos los ficheros originales que han sido modificados para poder llevar a cabo este trabajo

### B.1. test\_WebAssembly.js

La ruta completa en la que se encuentra este fichero es la siguiente:

*domserver/webapp/public/js/test\_WebAssembly.js*

En este fichero, como ya se explicó a lo largo del capítulo 4.2, es en el que se encuentra toda la funcionalidad realizada. Consta de unas dos mil quinientas líneas aproximadamente, las cuales contienen toda la funcionalidad autogenerada junto con las modificaciones realizadas.

### B.2. submit\_scripts.html.twig

La ruta completa en la que se encuentra este fichero es la siguiente:

*domserver/webapp/templates/team/partials/submit\_scripts.html.twig*

La modificación realizada es la adición de la línea de la figura B.1 al final de dicho archivo. Con esto se consigue importar el fichero test\_WebAssembly.js y que

se ejecute mediante jQuery.

```
<script src="{ asset("js/test_WebAssembly.js") }"></script>
```

Figura B.1: Import de test\_WebAssembly.js

### B.3. submit\_modal.html.twig

La ruta completa en la que se encuentra este fichero es la siguiente:

*/domserver/webapp/templates/team/submit\_modal.html.twig*

En este archivo se han realizado dos modificaciones. La primera modificación realizada es la adición del botón con id 'testFile' en el div que incluye el resto de botones, tal y como se ve en la figura B.2 Con esto se consigue añadir el botón 'Test' junto a los botones 'Submit' y 'Cancel'.

```
<div class="modal-footer">
  <button type="button" id="cancelBtn" class="btn btn-secondary" data-dismiss="modal">Cancel</button>
  <button type="button" id="testFile" name="boton" value="test" class="btn-info btn">Test</button>
  <button type="submit" id="submitBtn" name="boton" value="submit" class="btn-success btn">
    <i class="fas fa-cloud-upload-alt"></i> Submit
  </button>
</div>
```

Figura B.2: Botón id 'testFile'

La segunda modificación es la adición de un nuevo div, correspondiente a la ventana de salida asociada al estado de la compilación y el resultado del ejercicio. Este nuevo div se observa en la figura B.3.

```
<div id="ventanaSalida">
  <div id="compilando">Compilando...</div>
  <div id="comprobando"></div>
  <div id="solucion"></div>
  <button type="button" id="botonOkTest">Ok</button>
</div>
```

Figura B.3: Div ventana de salida

### B.4. style\_domjudge.css

La ruta completa en la que se encuentra este fichero es la siguiente:

---

*/domserver/webapp/public/style\_domjudge.css*

La modificación realizada es la creación de todos los estilos utilizados en los nuevos elementos HTML. Estos nuevos estilos se encuentran al final de dicho documento CSS.





## Apéndice C

# Introduction

This chapter describes the reason for our project, the objectives that were expected to be achieved, the plan developed to achieve these objectives and the languages used and their relationships with what was studied throughout the degree.

### C.1. Motivation

An online judge is a tool that allows you to load a series of problems with files that offer input and output parameters for codes in different programming languages. This is the case with DOMjudge, which is used at the university for educational purposes in some subjects. The general operation of DOMjudge consists of creating contests, which are used to group students by subjects, and uploading a battery of exercises that allow students to check their knowledge of the subject. Students make deliveries by uploading the code corresponding to each exercise indicating the exercise and the language in which it is programmed, the code is sent to the server where it is compiled and the student is offered a resolution on the compilation of the code.

The motivation of this TFG is to improve the current delivery system, which, as mentioned before, consists of uploading the files to the server and compiling and executing them there, which results in many unnecessary deliveries that contain errors. To achieve this improvement, the project offers a solution that allows compiling the exercises on the client from which the delivery is made and comparing the output obtained in certain test cases with the expected one for the exercise, thus avoiding that, if the uploaded code does not exceeds the initial samples proposed for exercise, the compilation is performed on the server reducing its workload and a failed delivery for the student.

Thanks to WebAssembly (Community, 2015) the solution to these problems can be created, so the client itself executes the file that has just been uploaded. In this way, if the result of the execution is not as expected, the file will be prevented from reaching the server, thus creating a first security filter and validation of results for these judges, thus avoiding the compilation of possible files that are harmful to the server.

## C.2. Objectives

The main objective of this project is to extend the functionality of DOMjudge so it's able to check the exercises before uploading them to the server. To achieve this functionality, it is necessary to create a new button that, when pressed, compiles the C++ file to be uploaded on the client and compares the output obtained with the expected output. The final goal is to ensure that the functionality of this button can be added to the upload button to the server, so that before sending the file to the server it is first compiled in WebAssembly. In this way, both online judges and the university itself might be able to implement it.

## C.3. Workplan

In order to achieve the objectives, a work plan divided into 4 phases was followed, these phases of the plan are described below:

1. **Research the possibilities of WebAssembly.** First, we will find out from various sources what exactly WebAssembly consists of and, once this has been assimilated, we will study how it works and what its possible uses are. In addition, we will begin with the installation of everything necessary for its operation. Once we have everything we need installed, we will begin to test the operation of this technology. To do this, we will look for tutorials or guides for simple problems, such as a 'Hello World!'. With this, we will have managed to create our simple project with WebAssembly and above all, we will have been able to verify that it works correctly for us. In this way, we will have a solid base to carry out the next phase. The expected time to carry out this phase is approximately the first four months, that is, until February.
2. **Local simulation of an online judge.** In the second phase we will focus mainly on creating a local judge that simulates the operation of DOMjudge, in which we will verify that the objective of this TFG is achievable. The main objective of this simulation is to be able to compile a file written in C++ corresponding to a possible university exercise using WebAssembly. If successful,

this functionality should be transferred to DOMjudge. The estimated time for this phase is approximately one month, that is, until March.

3. **Study and installation of DOMjudge.** DOMjudge is a complex program, so it has a rather tedious and complicated installation process. Once installed, we have to study how it is programmed (what language it uses, what technologies, standards, etc.) to know exactly what part of the code we must modify to add the new functionality. That is why the estimated time for said study and installation is approximately two weeks.
4. **Integration of WebAssembly in DOMJudge.** This is the main part of the project, in which the new functionality will be developed. To integrate this functionality it will be need to use Node.js (Wikipedia, Node.js), since it is necessary to compile WebAssembly. Once the DOMjudge language has been studied, the most optimal and efficient way to integrate Node.js will be sought. This phase is going to be the most complicated due to the lack of knowledge about the matter, so it is not possible to predict in advance the estimated time or the resources that will be needed for it. However, what can be predicted is that both the new functionality necessary to achieve the objective of the TFG and the writing of this report will be developed at this stage. That is why this stage will last until the final delivery of the project.

## C.4. Repository structure

The repository used for the development of the TFG, as mentioned above, is GitHub, which follows the following structure:

- **WebAssembly:** Directory used to store the Node.js server code programmed in JavaScript that allows WebAssembly to compile the uploads of the judge.
- **domjudge-7.3.3:** Directory downloaded directly from the DOMjudge page, which contains the domjudge-7.3.3 files on which we made the necessary modifications for the operation of our project, and to which we added a js file required.
- **README.md:** File containing the description of the project and the content of the repository.

## C.5. Subjects related to the TFG

Next, the subjects of the degree that have helped us to carry out this project are listed:

**1. Aplicaciones Web (AW)**

In this subject the basic knowledge of web programming is taught, as well as the use of HTML, CSS, JavaScript, Node, etc. This subject is the most useful for the TFG, since the entire project is development-oriented.

**2. Bases de datos (BD)**

In this subject you learn to handle relationship databases. Thanks to this, knowledge is acquired that allows to handle the DOMJudge database, developed in MariaDB.

**3. Sistemas Operativos y Administración de Sistemas y Redes (SO y ASR)**

In this subject you learn basic aspects to handle any Linux distribution, as well as to install and use virtual machines. Especially useful when installing the virtual machine and DOMJudge.

**4. Ingeniería del Software y Modelado de Software (IS y MS)**

In these subjects the most important design patterns used in the development of software projects are learned. It serves, above all, when creating the local simulation.

**5. Arquitectura Interna de Linux y Android (LIN)**

In this you learn to handle some of the internal processes of Linux, which is very useful to be able to install and handle DOMJudge, since all the judge has to be installed on a Linux-based operating system.

**6. Gestión de Proyectos Software (GPS)**

In this subject you learn to organize a software project. Although none of the methods studied for internal organization have been applied, it is useful to know how to organize it according to different availabilities.

## Apéndice D

# Results, conclusions and future work

**RESUMEN:** This section details the results and conclusions obtained, as well as the future work that could be carried out on this project.

### D.1. Results and conclusions

The final objective of the project was reached, which was to add a system to an online judge to execute a series of simple tests in the user's browser, making use of WebAssembly, to avoid unnecessary uploads.

For this reason, the development of the project consists of two parts: one, in which a small local judge is investigated and developed, who uses WebAssembly technology on his server, and with which the possibility of developing this clue system is confirmed. light on deliveries, making use of this technology; and a second, in which what has been implemented for the local judge is scaled, and is adapted to an online judge already developed, generating a solution that can be taken to other online judges.

### D.2. Future work

Due to what has been commented in the conclusions, this way of integrating WebAssembly is transferable to the rest of the online judges, since the server made with Node.js, which compiles to WebAssembly, is outside the judge itself, and that,

the JavaScript file with the execution of the compiled code, which in turn uses jQuery, has a code that can be easily modified to give it other functions, depending on the judge to which it is transferred. The only caveat could be to add the new buttons in the judge's code, since the structure might not be the same, but as long as they are modified appropriately, all the code would be reusable.

The main objective to be developed in the future would be the integration of the button functionality that we have implemented in the current delivery flow of the judge. In this way, we would add our work to the usual process, so that if the uploaded delivery does not exceed the test samples for the exercise, the browser window is displayed indicating the number of errors and the delivery process is stopped. But if the samples are successfully passed, the delivery continues naturally and is sent to the server.

The current state of the project does not allow compiling complex problems that use libraries or additional header files, which are used in subjects such as FAL or TAIS, since it works by generating temporary folders and files during compilations. As a result of this, a future work would be to load the necessary libraries in the temporary directory, which creates the WebAssembly glue code itself, to be able to add them to the temporary files so that it can access them in the execution process .

Future work could be to prepare the code to be used to compile any programming language that WebAssembly accepts, since it is currently only intended to work with C ++. This is why if we do not select C ++ in the language drop-down, an error message appears. The work to be done would be to adapt it to compile also in C and Rust, the other 2 languages accepted by WebAssembly. This would be very useful for any online judge who allows the upload of exercises in any of these languages. DOMjudge has the option to deliver in C, so it would be interesting to implement it for use in the faculty, since in some subjects, deliveries in C are also supported, for example.

Another future work could be to delve into the type of clues that it gives, and study the execution times and memory use of the functions implemented in the delivered code, in such a way that not only if they would give clues about the generated output, if not that could also give clues on how to optimize these functions. To do this, a possible solution would be to have for each problem some previous results obtained by making a delivery with the correct code in such a way that the deliveries made by the students can be compared with it and give clues based on the results they generate.

A functionality that can be very useful, can be to adapt, from what was developed for the local judge, the possibility of modifying the code in the browser itself, incorporating the judge, a text area that is filled with the content of the .cpp uploaded or with code written directly there, so that, when the test button is pressed and

---

information about the results is received, corrections can be made without going to the IDE, or similar, in which that code has been worked.





# Bibliografía

BATTAGLINE, R. *Hands-On Game Development with WebAssembly : Learn WebAssembly C++ Programming by Building a Retro Space Game*. PACKT Publishing, 2019. ISBN 1838646833.

COMMUNITY, W. Disponible en <https://webassembly.org/>.

CONTRIBUTORS, E. Disponible en <https://emscripten.org>.

HAAS, A., ROSSBERG, A., SCHUFF, D. L., TITZER, B. L., HOLMAN, M., GOHMAN, D., WAGNER, L., ZAKAI, A. y BASTIEN, J. Bringing the web up to speed with webassembly. *Association for Computing Machinery*, Disponible en <https://dl.acm.org/doi/epdf/10.1145/3140587.3062363>.

MOZILLA y INDIVIDUAL CONTRIBUTORS. Disponible en <https://www.mozilla.org/es-ES/>.

MOZILLA y INDIVIDUAL CONTRIBUTORS. Disponible en <https://developer.mozilla.org/es/docs/Web/JavaScript>.

PHAM, M. T. y NGUYEN, T. B. The domjudge based online judge system with plagiarism detection. En *2019 IEEE-RIVF International Conference on Computing and Communication Technologies (RIVF)*, páginas 1–6. 2019.

WIKIPEDIA (Node.js). Entrada: “Node.js”. Disponible en <https://es.wikipedia.org/wiki/Node.js> (último acceso, Agosto, 2017).