

Trabajo de Fin de Grado

Grado en Ingeniería Informática

UniTestCode: Plataforma web para la evaluación del código mediante tests

*UniTestCode: Web platform for code evaluation through
tests*

Alexander González Covic

D. **Alberto Hamilton Castro**, profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

C E R T I F I C A

Que la presente memoria titulada:

“UniTestCode: Plataforma web para la evaluación del código mediante tests”

ha sido realizada bajo su dirección por D. **Alexander González Covic**.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 10 de junio de 2024

Agradecimientos

Quiero agradecer a mi tutor Alberto Hamilton Castro por orientarme en el desarrollo del trabajo de fin de grado y por la retroalimentación que me ha ido aportando a lo largo del desarrollo de este, tanto en el desarrollo de la plataforma como en lo relacionado con la memoria y la presentación.

Por otra parte, agradecer a mi madre por el apoyo que me ha dado a lo largo de la carrera, por darme los valores que tengo y por sus consejos, ya que sin ella esto no sería posible.

También agradecer al resto de mi familia, por el apoyo que me han dado en todo y por estar ahí siempre.

Por último, agradecer a mis compañeros que he conocido a lo largo de la carrera, los cuales han sido de muchísimo apoyo en la carrera y han hecho la experiencia universitaria mucho mejor, además de haberse convertido en personas muy importantes.

Licencia



© Esta obra está bajo una licencia de Creative Commons Reconocimiento-NoComercial-SinDerivadas 4.0 Internacional.

Resumen

El objetivo de este trabajo ha sido la realización de una plataforma web fullstack, llamada UniTestCode, la cual es una solución que busca simplificar la tarea de evaluar el código de estudiantes en diferentes niveles y lenguajes de programación. Actualmente, esta tarea puede resultar compleja y consumir mucho tiempo tanto para los profesores como para los alumnos, desviándose del objetivo principal que es la enseñanza o el aprendizaje. UniTestCode busca resolver este problema proporcionando la opción al profesorado de crear tests personalizados de manera intuitiva y con bastante personalización, pudiendo evaluar desde el código general hasta zonas específicas. Esto permite al profesorado evaluar el código del alumnado de manera mucho más sencilla, además de poder recopilar información del aprendizaje y poder identificar mejor los problemas que surjan por parte del alumnado al aprender el lenguaje y así mejorar la enseñanza. Además, al alumnado le permite mejorar el aprendizaje del lenguaje pudiendo verificar si la solución del código es válida o no y viendo cuáles son sus fallos.

Palabras clave: Testing, FullStack, Aprendizaje, NextJS, Socket.IO, Docker, PostgreSQL, React, TypeScript.

Abstract

The objective of this work has been the realization of a fullstack web platform, called UniTestCode, which is a solution that seeks to simplify the task of evaluating the code of students at different levels and programming languages. Currently, this task can be complex and time-consuming for both teachers and students, diverting from the main objective of teaching or learning. UniTestCode seeks to solve this problem by providing the option for teachers to create custom tests in an intuitive and highly customizable way, being able to evaluate from general code to specific areas. This allows the teacher to evaluate the student's code in a much simpler way, as well as being able to collect learning information and better identify problems that arise on the part of the student when learning the language and thus improve teaching. In addition, it allows students to improve their language learning by being able to verify whether the code solution is valid or not and to see what their failures are.

Keywords: Testing, FullStack, Learning, NextJS, Socket.IO, Docker, PostgreSQL, React, TypeScript.

Índice general

Capítulo 1	Introducción.....	1
1.1	Motivación.....	1
1.2	Objetivos.....	1
1.3	Descripción del proyecto.....	2
1.4	Estado del arte.....	4
Capítulo 2	Lenguajes y tecnologías utilizadas.....	5
2.1	Lógica de la aplicación (<i>Backend</i>).....	5
2.1.1	Node y Express	5
2.1.2	TypeScript	5
2.1.3	Socket.IO	6
2.1.4	Docker.....	6
2.2	Interfaz de la aplicación (<i>Frontend</i>).....	7
2.2.1	NextJS	7
2.2.2	React.....	7
2.2.3	TailwindCSS.....	7
2.2.4	ShadcnUI	7
2.3	Base de Datos	8
2.3.1	Supabase.....	8
Capítulo 3	Desarrollo del proyecto	9
3.1	Autenticación y Usuarios.....	9
3.2	Ejecución de código desconocido.....	10
3.3	Creación de una Suite	15
3.4	Creación de Tests Generales.....	16
3.5	Creación de Especificaciones y Tests Específicos.....	17
3.6	Ejecución de Tests.....	20
3.7	Otros aspectos desarrollados	23
3.7.1	Suite con múltiples alumnos y profesores.....	23
3.7.2	Patio de juegos o <i>Playground</i>	24
3.7.3	Modo Claro y Oscuro.....	25
Capítulo 4	Resultados y Guía de uso.....	26

4.1	Autenticación	26
4.2	Creación de Suites.....	27
4.3	Creación de Tests Generales.....	30
4.4	Creación de Tests Específicos.....	31
4.5	Gestión de usuarios a una Suite.....	33
4.6	Ejecución de Tests.....	33
4.7	Obtención de estadísticas	35
Capítulo 5	Conclusiones y líneas futuras	36
Capítulo 6	Summary and Conclusions	38
Capítulo 7	Presupuesto.....	40
Bibliografía.....		42

Índice de figuras

Figura 1.1	Creación de tests	3
Figura 1.2	Ejemplo ejecución de tests.....	3
Figura 1.3	Ejemplo diferencia de salidas.....	3
Figura 1.4	Página de LeetCode	4
Figura 1.5	Diseño CheckMyCode.....	4
Figura 2.1	Relación TypeScript y JavaScript	5
Figura 2.2	Base de datos en Supabase.....	8
Figura 3.1	Diagrama ejecución del código	12
Figura 3.2	Ejemplo código suma	14
Figura 3.3	Diagrama creación de una Suite.....	15
Figura 3.4	Ejemplo Especificación Factorial.....	19
Figura 3.5	Relación Suites y Usuarios	24
Figura 3.6	Playground.....	25
Figura 3.7	Ejemplo modo oscuro.....	25
Figura 4.1	Pantalla Principal UniTestCode.....	26
Figura 4.2	Inicio de Sesión.....	26
Figura 4.3	Creación de cuenta.....	26
Figura 4.4	Desplegable usuario.....	27
Figura 4.5	Panel de control del profesor	27
Figura 4.6	Opciones de Suite.....	28
Figura 4.7	Ir al panel de edición de la Suite.....	29
Figura 4.8	Ir a la gestión de usuarios	29
Figura 4.9	Panel de gestión de usuarios.....	29
Figura 4.10	Ir al panel de creación de Tests Generales.....	30
Figura 4.11	Panel de creación de Tests	30
Figura 4.12	Módulo de entradas.....	30
Figura 4.13	Ir a crear una especificación.....	31
Figura 4.14	Menú de creación de especificación	31

Figura 4.15 Ejemplo especificación.....	32
Figura 4.16 Ir al panel de creación de Test Específico.....	32
Figura 4.17 Panel de gestión de usuarios.....	33
Figura 4.18 Apartado de Suites.....	33
Figura 4.19 Ejemplo dentro de una Suite.....	34
Figura 4.20 Ejemplo ejecución tests	34
Figura 4.21 Ejemplo diferencias de salida.....	34
Figura 4.22 Ir al panel de estadísticas de la Suite.....	35
Figura 4.23 Panel de estadísticas	35

Índice de códigos

Código 3.1 Ejemplo registro con Supabase.....	9
Código 3.2 Medida de seguridad en BDD.....	10
Código 3.3 Medida de seguridad en Backend	10
Código 3.4 Imagen Docker.....	11
Código 3.5 Opciones Inicialización Docker	12
Código 3.6 Obtención de la salida y por qué finalizó la ejecución	13
Código 3.7 Ejemplo Input	14
Código 3.8 Ejemplo Salida	14
Código 3.9 Creación de Suite (simplificado).....	16
Código 3.10 Ejemplo estructura Test	16
Código 3.11 Obtención del código de la Suite.....	17
Código 3.12 Ejecución y almacenamiento del Test.....	17
Código 3.13 Ejemplo Especificación Código General	18
Código 3.14 Ejemplo Especificación Código Factorial.....	18
Código 3.15 Ejemplo Etiqueta Especificación	20
Código 3.16 Obtener código sin etiqueta.....	20
Código 3.17 Obtener opciones de compilación/ejecución de la BDD.....	21
Código 3.18 Obtener zonas específicas del alumno.....	21
Código 3.19 Especificación no encontrada.....	22
Código 3.20 Ejecución de tests y comprobación de salidas.....	23
Código 3.21 Insertar usuario en una Suite	24
Código 4.1 Ejemplo fichero de configuración entradas	31

Índice de tablas

Tabla 2.1 WebSockets vs Long Polling.....	6
Tabla 4.1 Ejemplo fichero estadísticas.csv.....	35
Tabla 7.1 Presupuesto.....	41

Capítulo 1

Introducción

1.1 Motivación

En la actualidad evaluar el código de estudiantes es una tarea que puede llevar bastante tiempo, además de presentar una cierta complejidad tanto para el profesor que lo quiera corregir, como para el alumno que quiera que su código sea evaluado.

Esta complejidad se da tanto por la parte del alumnado como para el profesorado, donde, si bien existen otras herramientas que se puedan utilizar para esto, no son tan personalizadas ni están tan centradas en la simplicidad. Además, el aprendizaje de éstas desviaría la atención del docente, donde su objetivo debería ser, simple y llanamente, enseñar el lenguaje en cuestión y del alumnado, donde su objetivo es el aprendizaje de éste.

Por ello, UniTestCode surge como una solución a esta problemática con el fin de facilitar tanto al profesorado la tarea de evaluar el código del alumnado de diferentes grados en distintos lenguajes de programación, como al alumnado el poder verificar si la solución es correcta mediante una serie de tests con retroalimentación. Estos lenguajes de programación pueden ser lenguajes de alto nivel como C++ o Python, hasta lenguajes de bajo nivel como puede ser MIPS. La evaluación del código se realizará mediante tests diseñados por el profesorado de manera bastante sencilla e intuitiva donde el alumnado subirá el código para poder verificar si la solución del ejercicio es correcta y poder hacer cambios respecto al resultado de los tests.

1.2 Objetivos

El objetivo principal del proyecto es facilitar la evaluación por parte del profesorado sobre el código del alumno de manera sencilla e intuitiva para no tener que desviarse de la enseñanza en el aprendizaje de la plataforma. También, por parte del alumnado, el principal objetivo será permitir verificar si la solución al ejercicio es correcta y poder así realizar cambios oportunos y con ello mejorar el aprendizaje. Por ello, podríamos considerar los siguientes objetivos a cumplir a la hora de desarrollar el proyecto:

- **Aceptar múltiples lenguajes de programación**, siendo necesario que el proyecto pueda aceptar múltiples lenguajes de programación para que sea más flexible a la hora de poder usarse en diferentes asignaturas y grados, independientemente del lenguaje.
- **Poder verificar si la solución es la correcta**, de esta manera el alumnado podría saber si la solución planteada para el problema es válida o no y en el caso de que no lo sea, tener una retroalimentación con las diferencias respecto a la solución correcta.

- **Facilitar la creación de pruebas al profesorado** mediante una interfaz sencilla e intuitiva para poder centrarse más en la enseñanza del lenguaje que en el aprendizaje de la plataforma a la hora de la creación de tests.
- **Ejecutar el código de manera segura y sencilla**, con el fin de evitar filtraciones de datos o problemas de seguridad con la aplicación, además de facilitar la ejecución del código al alumnado sin necesidad de tener instalado nada en su máquina.
- **Registrar el progreso del alumnado** a la hora de intentar aprobar los diversos tests que existan, con el fin de poder analizar posteriormente el rendimiento y dificultades que puedan existir.
- **Permitir la configuración de las Suites de una manera más profunda** como puede ser el tiempo máximo de ejecución o la memoria máxima, entre otros, para poder ajustar las necesidades de la prueba lo máximo posible por parte del profesorado.
- **Poder evaluar partes específicas del código** al mismo tiempo que el código general y así poder aislar tests relacionados con partes concretas del código, como pueden ser las funciones.

1.3 Descripción del proyecto

El proyecto consiste en la creación de una plataforma web denominada UniTestCode creada para facilitar la evaluación del código del alumno como profesor. En la plataforma, tendremos principalmente dos tipos de usuarios: **Alumno y Profesor**.

El profesor podrá crear lo que llamaremos *Suites*, donde cada Suite tendrá un código del profesor, el cual se utilizará para generar la solución correcta y un conjunto de **Tests** diseñados por ellos mismos para que el alumno los supere (*más adelante se entrará en detalle de cómo funciona*).

Estos Tests estarán divididos en dos secciones:

- **Tests Generales:** Estos tests son generados mediante el código de la Suite y se ejecutarán independientemente del lenguaje de programación que escoja el alumno dentro de los permitidos por el profesor.
- **Tests Específicos:** Estos al contrario que los generales, son unos tests más específicos pensados para poner a prueba funciones o trozos de código del proporcionado por el alumno.

En la ejecución del código, cada test se diferenciará del resto además de por su título y descripción, por las entradas y argumentos que reciba el programa. En el test, se almacenará la salida que devuelva la ejecución del código del profesor con las entradas y argumentos del test para su posterior comparación con la ejecución por parte del alumno.

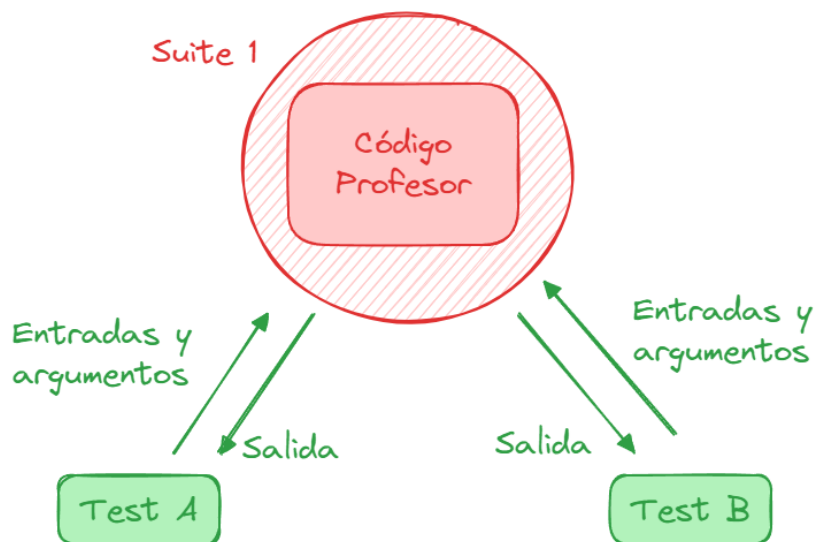


Figura 1.1 Creación de tests

Por otro lado, el Alumno tendrá a su disposición una serie de Suites creada por los profesores donde deberá subir su código, el cual se ejecutará contra los diferentes tests.

A la hora de comprobar si el test ejecutado con el código del alumno es válido, se observará la diferencia de la salida generada con el código del alumno y la salida almacenada en el test. Si no existe diferencia entre ambas salidas, se da por correcto el test. En el caso de que no, se da por fallido.

Factorial de 2	
<p>Output</p> <p>Introduce un número: 2 El factorial de este número es: 2</p>	<p>Expected Output</p> <p>Introduce un número: 2 El factorial de este número es: 2</p>
Factorial de 3	
<p>Output</p> <p>Introduce un número: 3 El factorial de este número es: 3</p>	<p>Expected Output</p> <p>Introduce un número: 3 El factorial de este número es: 6</p>

Figura 1.2 Ejemplo ejecución de tests

Introduce un número: 3
El factorial de este número es: 36

Close

Figura 1.3 Ejemplo diferencia de salidas

1.4 Estado del arte

Actualmente existen varias plataformas webs para resolver ejercicios o problemas de código de manera online con tests. Entre estas plataformas, destaca LeetCode [1], siendo la más popular y conocida que existe, donde mucha gente entra a mejorar su capacidad de programación gracias a los ejercicios que ofrece la página.

#	Title	Solution	Acceptance	Difficulty	Frequency
1712	Ways to Split Array Into Three Subarrays		29.8%	Medium	
1710	Maximum Units on a Truck		70.6%	Easy	
1705	Maximum Number of Eaten Apples		42.0%	Medium	
1704	Determine if String Halves Are Alike		77.6%	Easy	
1696	Jump Game VI		53.8%	Medium	
1695	Maximum Erasure Value		49.6%	Medium	
1690	Stone Game VII		47.6%	Medium	
1689	Partitioning Into Minimum Number Of Deci-Binary Numbers		87.4%	Medium	

Figura 1.4 Página de LeetCode

Como esta existen muchas más, como pueden ser Exercism [2], HackerRank [3], siendo la problemática de todas estas el aprendizaje de la plataforma y el no tener la opción de poder crear tus tests de manera personalizada con tanta libertad.

Por último, hay que destacar un trabajo llamado CheckMyCode [4], siendo este un trabajo muy reciente (2020), donde también trata de simplificar la tarea de evaluación de código del alumno, siendo un proyecto basado en PHP y con la diferencia de estar orientado solamente a Java y no tener un sistema de tests como tal.

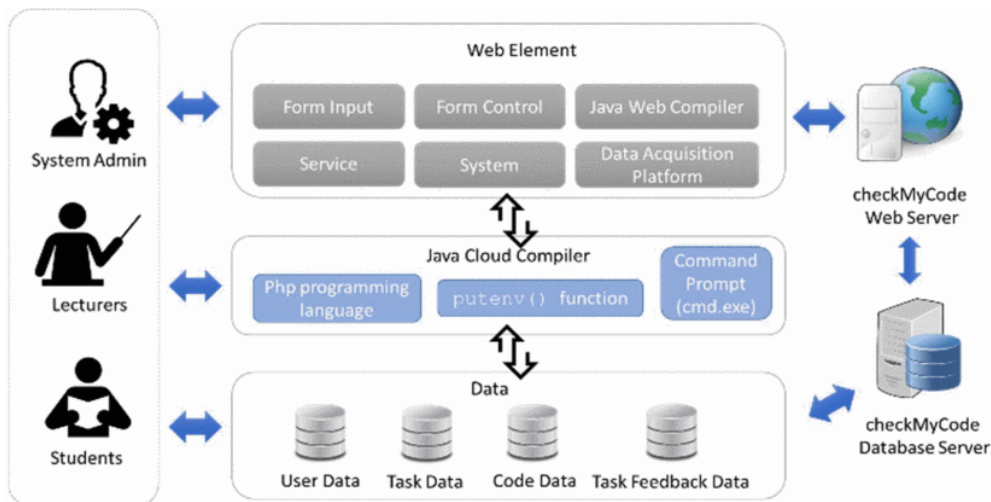


Figura 1.5 Diseño CheckMyCode

Capítulo 2

Lenguajes y tecnologías utilizadas

En este capítulo se hablará de las tecnologías y herramientas utilizadas y el por qué se ha optado por elegir estas entre otras disponibles.

Para lograr cumplir con los objetivos que requiere este proyecto, se han utilizado múltiples herramientas y tecnologías, teniendo muchos detalles en mente como puede ser el rendimiento de la plataforma, la interfaz o la comunicación entre el servidor y el usuario.

2.1 Lógica de la aplicación (*Backend*)

2.1.1 Node y Express

Para el servidor se ha optado por utilizar Node [5] como entorno de ejecución debido a varias razones. Entre ellos la gran cantidad de paquetes que existen para Node, los cuales simplificarían el proceso de creación del proyecto, además de la eficiencia de este y la utilización del mismo lenguaje tanto en el frontend como en el backend, ayudando a la coherencia del código y simplificando el desarrollo.

Además, para el desarrollo de la API se ha optado por utilizar el framework ExpressJS [6], siendo el framework más popular para la realización de APIs con Node y ofreciendo todo lo necesario para la realización del proyecto. Este entre otras cosas ha sido elegido por la simplicidad, pudiendo así enfocarse más en el desarrollo de la aplicación, además de la gran comunidad que lo respalda y los paquetes que se crean entorno a ExpressJS.

2.1.2 TypeScript

Como lenguaje de programación se ha optado por TypeScript [7] gracias a sus ventajas respecto a JavaScript, siendo este un superconjunto del lenguaje, creado por Microsoft para combatir las limitaciones que presentaba JavaScript en proyectos más grandes y complejos.

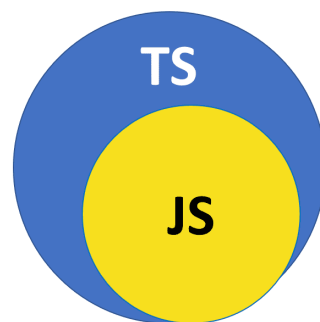


Figura 2.1 Relación TypeScript y JavaScript

Entre sus múltiples ventajas, las que más llamaron la atención a la hora de escogerlo frente a JavaScript son el tipado y las interfaces incluye, ofreciendo más seguridad y robustez

a la hora de desarrollar el código, además de facilitar la escalabilidad a lo largo del tiempo y la legibilidad de este.

2.1.3 Socket.IO

Para la comunicación entre el servidor y el usuario durante la ejecución del código se plantearon varias ideas. En el proceso de investigación, se encontró como otras páginas utilizaban lo que se conoce como Long Polling [8], donde una vez se ejecutan los tests, el cliente va enviando consultas al servidor para actualizar el estado.

Esto es una manera de resolver el problema con este tipo de aplicaciones para poder ver en “tiempo real” el resultado de los tests a medida que se van ejecutando, tratando de simular una comunicación bidireccional. Tras investigar un poco, se decidió utilizar en vez del Long Polling, WebSockets, donde si bien es más difícil de integrar, los beneficios son varios [9]. Entre estos encontramos algunos como:

- **El tipo de conexión**, ya que si bien el Long Polling, simula una comunicación bidireccional, esta no lo es, al contrario que los WebSockets.
- **La latencia**, donde al contrario que en el Long Polling, al ser una conexión bidireccional, el servidor le puede mandar al cliente nada más se termine de resolver un test el resultado sin tener que esperar a que el cliente haga la consulta para actualizar el estado.
- **La compatibilidad es de casi un 100%**, donde si bien uno de los problemas que existía antes a la hora de integrar WebSockets era la poca compatibilidad que había en los navegadores, mientras que hoy en día, en los navegadores de escritorio (*la mayoría del alumnado usará estos*) es del ~99.75%.

Características	WebSockets	Long Polling
Conexión	Bidireccional	Unidireccional
Latencia	Baja	Variable, pero mayor que los WebSockets
Requerimientos	Soporte para WebSockets	Ninguno

Tabla 2.1 WebSockets vs Long Polling

En la integración de los WebSockets se ha optado por utilizar la librería Socket.IO [10], siendo una librería muy completa además de robusta, con características como en el caso de que no sea posible la conexión mediante WebSockets, se utilizará HTTP Polling o si la conexión se pierde, el cliente se intentará conectar de vuelta automáticamente.

2.1.4 Docker

Uno de los mayores problemas de este proyecto es la seguridad relacionada a la hora de ejecutar código no controlado en el servidor. Se ha estado investigando y se ha cambiado varias veces a lo largo del proyecto hasta dar con la solución en la cual se entrará más en profundidad más adelante, aunque finalmente, se ha terminado optando por utilizar Docker [11] para esta tarea.

La razón por la que se ha usado Docker es que, al ejecutar código ajeno, necesitamos un entorno de ejecución seguro donde si se insertase código maligno y explotara el entorno, a la aplicación no le ocurra nada y pudiera funcionar con total normalidad. Otra de las decisiones que impulsó al uso de Docker en este proyecto es que, al ejecutarse código ajeno, si no existe un entorno seguro para ejecutarse, se podría llegar a filtrar mucha información comprometida del servidor como lo pueden ser credenciales entre otros, resolviendo así este problema.

2.2 Interfaz de la aplicación (*Frontend*)

2.2.1 NextJS

Para el apartado de la interfaz de la aplicación, primero se comenzó a realizar mediante el framework AstroJS [12] debido a su rapidez comparado con el resto de frameworks, aunque lamentablemente se dejó de utilizar ya que era un framework bastante reciente, con lo cual había bastantes problemas y ciertas incompatibilidades que retrasarían el desarrollo del proyecto.

Por ello, entre el resto de frameworks, se ha decidido usar el framework de código abierto NextJS [13] por varios motivos. Entre estos destacan algunos como las facilidades que ofrece en el desarrollo gracias a su modularización y enrutamiento, la gran comunidad que le respalda o el gran soporte que tiene para integrar diferentes APIs, ya sean de autenticación como pueden ser de WebSockets, siendo el creador de NextJS, el mismo de Socket.IO.

2.2.2 React

NextJS para la generación de sus páginas utiliza el framework de React [14] siendo React una biblioteca de JavaScript de código abierto desarrollada por Facebook para construir interfaces. React está basado en la modularización de los componentes de una página web, por lo que ayuda a realizar código más limpio además de facilitar la escalabilidad de los proyectos.

Si bien React se desarrolla mediante JavaScript, NextJS permite el desarrollo con TypeScript, lo cual hace el código mucho más robusto gracias a sus interfaces, siendo esta la elección tomada a la hora de desarrollar el código en React.

2.2.3 TailwindCSS

Para los estilos de la página se ha decidido utilizar el framework de TailwindCSS [15], ganando cada vez más y más relevancia, ya que al contrario que otros frameworks como Bootstrap, TailwindCSS no ofrece clases predefinidas, sino que facilita el desarrollo de los estilos de componentes, complementándose de manera muy adecuada con frameworks como React.

2.2.4 ShadcnUI

En este proyecto, para agilizar el desarrollo y poder centrarse más en la lógica y funcionamiento de la plataforma, se optó por utilizar la famosa librería de componentes ShadcnUI [16]. ShadcnUI es una de las librerías más populares del último año de todo GitHub, proporcionando una base de componentes muy personalizables con una estructura muy sólida, además de una coherencia en todo el ecosistema, haciendo el código mucho más

sólido y escalable a lo largo del tiempo, proporcionando componentes como botones, acordeones o desplegados, entre muchos otros.

2.3 Base de Datos

2.3.1 Supabase

Como base de datos, se ha optado por utilizar el framework Supabase [17]. En un principio se pensó en utilizar directamente PostgreSQL [18] o MySQL [19], pero en el desarrollo del proyecto, se quiso implementar la autenticación mediante GitHub e investigando como implementarlo, se encontraron múltiples frameworks y entre ellos estaba Supabase.

Supabase es una alternativa de código libre de Firebase [20] basado en PostgreSQL bastante moderno y actualizado, contando además con una gran comunidad. Se ha decidido utilizar Supabase por varios motivos, siendo el principal el ser de código abierto, además de su interfaz amigable para el desarrollo y las utilidades que ofrece para simplificar cosas como son la autenticación, entre otras.

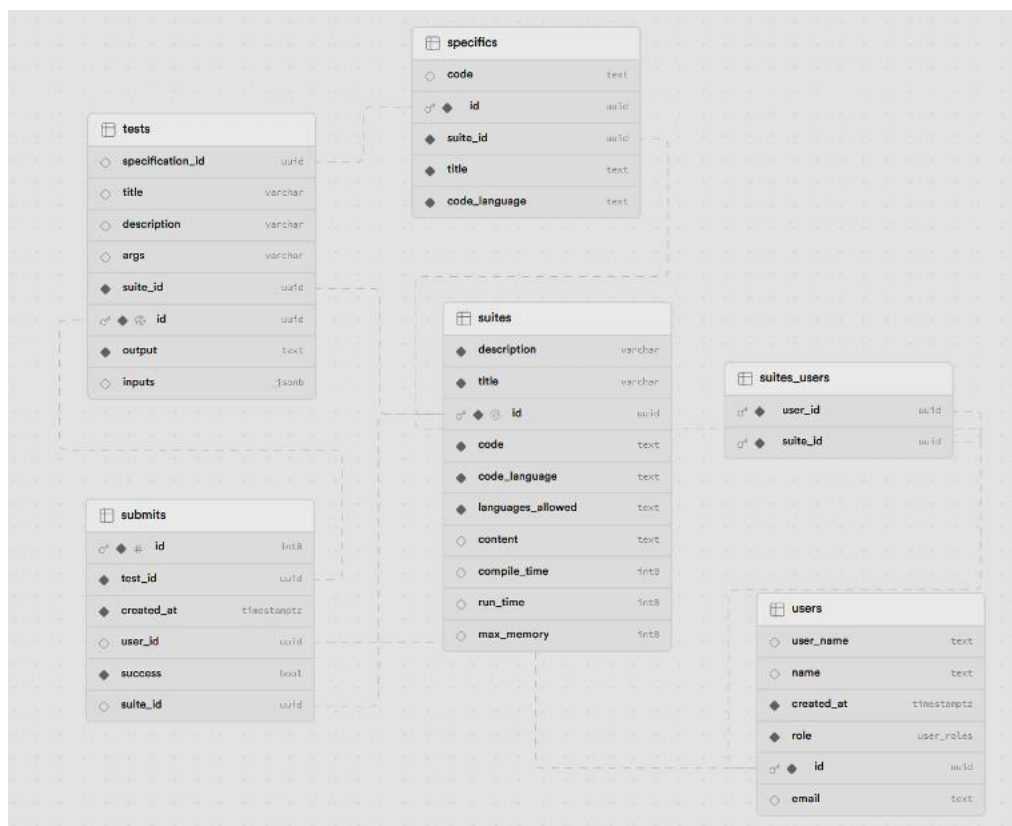


Figura 2.2 Base de datos en Supabase

Capítulo 3

Desarrollo del proyecto

En este capítulo se comentará como se han desarrollado e implementado las funcionalidades del proyecto para poder cumplir con los objetivos. El repositorio del proyecto se puede encontrar en la bibliografía [21]. Las implementaciones realizadas a lo largo del proyecto se pueden dividir en:

- Autenticación del usuario y distinción entre Profesor y Alumno.
- Ejecución de código desconocido de manera segura.
- Creación de una Suite
- Creación de Tests Generales
- Creación de Especificaciones y Tests Específicos
- Ejecución de Tests

3.1 Autenticación y Usuarios

Para la autenticación de los usuarios se ha hecho uso de las herramientas que ofrece Supabase. Gracias a Supabase, además de poder iniciar sesión mediante el correo, también se podrá hacer mediante GitHub.

Para ello, se ha creado un apartado para iniciar sesión y registrarse en la plataforma web, donde el usuario podrá registrarse de la manera más cómoda que le resulte.

```
export const signUpWithEmail = async (
  supabase: SupabaseClient,
  { email, password, data }: SignUpWithEmailInput
): Promise<string | undefined> => {
  const { error } = await supabase.auth.signUp({
    email,
    password,
    options: {
      data,
      emailRedirectTo: '/'
    }
  })
  if (error !== null) return error.message
}
```

Código 3.1 Ejemplo registro con Supabase

Una vez registrado, por defecto el usuario tendrá el rol de alumno, con lo que solo podrá entrar a Suites para poner su código a prueba, además de una zona denominada

“Playground” donde el alumnado podrá probar su código con los argumentos y entradas que él quiera con el fin de practicar con más libertad y no solo con los ejercicios del profesorado. En cambio, el profesorado tendrá acceso total a toda la plataforma sin restricciones.

Para evitar filtraciones del código del profesorado y que los alumnos consigan el código correcto, desde la base de datos mediante cláusulas se ha controlado quien tiene acceso a ver el código y quien no además de comprobarlo también en el backend, forzando a que solamente los profesores puedan ver el código correcto. Esta dinámica se ha aplicado a todos los procesos en los que solamente el profesor puede realizar acciones y así tener dos capas de seguridad.

```
CASE
  WHEN (auth.uid() IN (
    SELECT users.id FROM users
    WHERE users.role = 'teacher'::user_roles )
  ) THEN suites.code
  ELSE NULL::text
END AS code,
CASE
  WHEN (auth.uid() IN (
    SELECT users.id FROM users
    WHERE users.role = 'teacher'::user_roles )
  ) THEN suites.code_language
  ELSE NULL::text
END AS code_language
```

Código 3.2 Medida de seguridad en BDD

```
const supabase = getSupabase({ req })
const userId = (await supabase.auth.getUser()).data.user?.id

if (!userId) return res.status(409).json({ msg: OTHER_ERRORS.ONLY_TEACHERS })
const { data: users } = await supabase
  .from('users')
  .select('role')
  .eq('id', userId)

const { role } = users?.length === 1 ? users[0] : { role: 'student' };
if (role !== 'teacher') return res.status(400).json({ msg: OTHER_ERRORS.ONLY_TEACHERS })

req.body.userId = userId
next()
```

Código 3.3 Medida de seguridad en Backend

3.2 Ejecución de código desconocido

Para la ejecución de código desconocido en un principio se iba a realizar en el sistema del propio servidor con un usuario que tuviera pocos privilegios. Al comenzar a desarrollar el

proyecto y observar las opciones que pueden existir a la hora de la creación de tests, nos dimos cuenta rápidamente que escribiendo cierto código o pasando ciertos argumentos a la hora de la ejecución, se podría saltar de alguna manera la seguridad y obteniendo datos sensibles como pueden ser credenciales de la base de datos entre otros.

Investigando de cómo otras páginas similares ejecutaban el código, se llegó a la conclusión que se ejecutaban en lo que se conoce como **sandbox**, donde un sandbox no es otra cosa que un entorno controlado y seguro para el usuario en el que se pueda ejecutar código malicioso y no ocurra nada más allá de que el sandbox deje de funcionar. Entre las opciones que se investigaron para implementar un sandbox en el que ejecutar código, se dieron con varias.

Primero, se dió con la API de Judge [22, 21], lo cual era perfecto para el proyecto ya que mediante su API se podría ejecutar el código de manera remota sin preocuparme por la seguridad. Los principales motivos por los que no se llegó a escoger esta herramienta fueron dos:

- **El precio:** Si bien existe un plan gratuito a la hora de utilizar esta API, solamente permite 50 ejecuciones diarias de manera gratuita, donde si en cada Suite hay varios tests y para cada test necesitamos una ejecución, no nos daría para nada.
- **Los lenguajes disponibles:** Si bien dispone de una gran selección de lenguajes compatibles, nos limita a los que están disponibles y a la hora de querer crear tests para lenguajes que no estén en la lista, no podríamos, como pueden ser lenguajes como Prolog o MIPS, los cuales se utilizan en la universidad y no se encuentran soportados.

Tras seguir investigando, se encontró el siguiente artículo [23] que explicaba cómo crear un entorno seguro con Docker. Se optó por realizar el sandbox de esta manera por la libertad y el control que otorgaba el realizarlo así en vez de utilizar una API externa.

Para ello, se creó una imagen basada en Linux con lo necesario para ejecutar los lenguajes que queramos soportar.

```
FROM ubuntu:latest

RUN apt-get update && apt-get install -y \
    build-essential python3 \
    python3-pip nodejs \
    npm spim

WORKDIR /app

CMD ["bash"]
```

Código 3.4 Imagen Docker

Además, a la hora de ejecutar funcionalidades en la instancia de Docker, se han limitado recursos como el acceso a internet, la memoria máxima permitida o la cantidad de CPU que puede utilizar:

```

const args = ['run', '--cpus', maxCpu,
  '--network', 'none',
  '--memory', `${maxMemory}m`,
  '--name', name,
  '-d', '-w', '/app', '-t', image
];
const docker = spawn('docker', args);

```

Código 3.5 Opciones Inicialización Docker

Una vez creada la imagen de Docker, se comenzará la ejecución del código:

Primero, se comprobará si existe un contenedor activo con el ID del usuario que ejecute el código, donde si existe, simplemente se obtiene el ID del contenedor y en el caso de que no exista, se inicia uno y se obtiene el ID del contenedor.

Después, se tratará de ejecutar el código como se especifica en el siguiente diagrama:

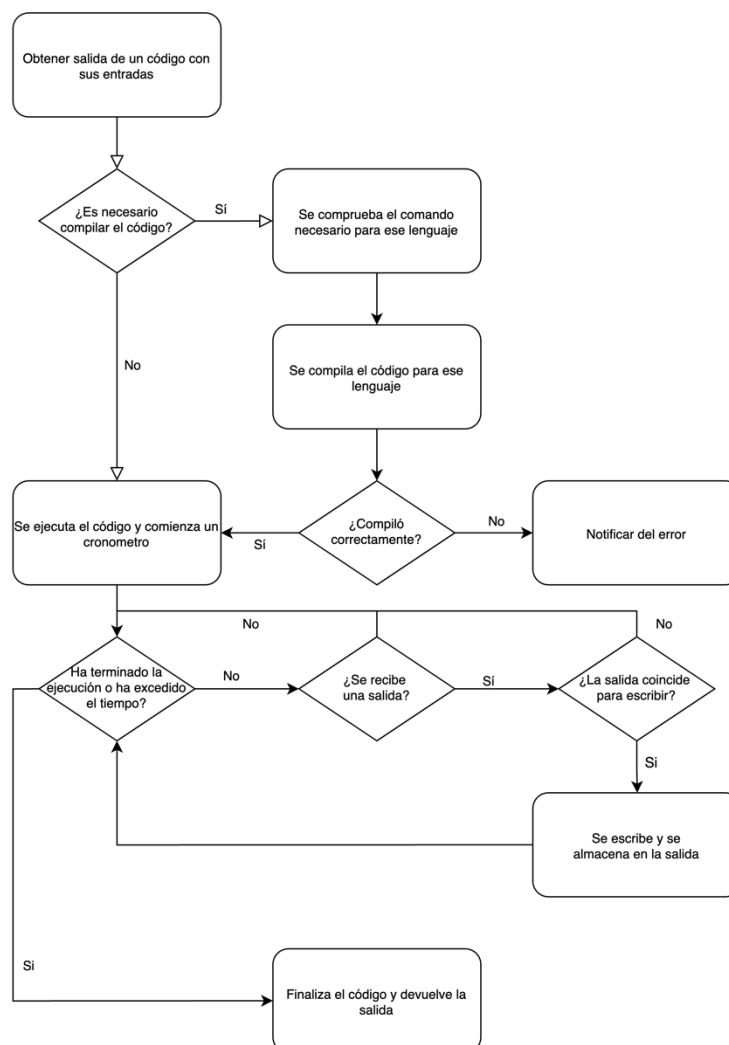


Figura 3.1 Diagrama ejecución del código

Tanto para la compilación como ejecución del código, lo haremos en la instancia de Docker para que resulte más seguro mediante procesos y nos comunicaremos con esta instancia mediante procesos con Node utilizando su librería `child_process` [24]. Para compilar el código lo primero que haremos es comprobar si el lenguaje del código está incluido en una lista de lenguajes compilados. En el caso de que esté, se obtiene el comando para compilar el lenguaje y se compila.

```
const compileCPP = ({ inputFile, executableFile }: CompileCommandInput) => {
  return {
    command: 'g++',
    args: ['-x', 'c++', inputFile, '-o', executableFile]
  }
}

const compileCommands: CompileCommand = Object.freeze({
  cpp: compileCPP,
  c: compileC
  ...
})

const executable = crypto.randomUUID()
const compileOptions = compileCommands[language]({ inputFile: filename,
executableFile: executable})
const args = ['docker', 'exec', containerId, compileOptions.command]
if (compileOptions.args) args.push(...compileOptions.args)
```

A la hora de ejecutar el código, iremos almacenando la salida en una variable y una vez finalice o se exceda del tiempo máximo de ejecución, devolveremos la salida de la ejecución.

Para interrumpir la ejecución cuando ha excedido el tiempo máximo, se realiza mediante la función `timeout` que incluye Linux, donde le pasamos el tiempo máximo y el proceso que queremos ejecutar. A la hora de determinar si ha terminado correctamente o ha terminado por exceder el tiempo máximo u otra razón, lo determinaremos por el código que devuelve al finalizar la ejecución.

```
runner.on('exit', (code) => {
  if (code === 124) resolve({ output: `${output}\nMaximum time expired`, code: -1 })
  if (code === 137) resolve({ output: `${output}\nMemory limit exceeded`, code: -1 })
  resolve({ output, code: (code ?? -1) })
})

runner.on('error', (err) => {
  reject({ output: `${output}\nError: ${err.message}`, code: -1 })
})
```

Código 3.6 Obtención de la salida y por qué finalizó la ejecución

Para saber cuándo es necesario escribir una entrada, se ha creado la estructura de datos *Input*, que no tiene más que dos parámetros:

- **has:** Contiene el texto que ha de incluir la salida para escribir un valor
- **value:** Contiene el valor que se escribirá

Un ejemplo sería la ejecución de un programa que reciba dos números a sumar:

```
#include <iostream>

int main () {
    int a, b;
    std::cout << "Introduce un número: ";
    std::cin >> a;
    std::cout << "Se ha introducido el " << a << std::endl;
    std::cout << "Escoge otro número: ";
    std::cin >> b;
    std::cout << "Se ha introducido el " << b << std::endl;
    std::cout << a << " + " << b << " = " << a + b << std::endl;
    return 0;
}
```

Figura 3.2 Ejemplo código suma

```
{
  "has": "Introduce un número:",
  "value": "3"
},
{
  "has": "número:",
  "value": "5"
}
```

Código 3.7 Ejemplo Input

Output

```
Introduce un número: 3
Se ha introducido el 3
Escoge otro número: 5
Se ha introducido el 5
3 + 5 = 8
```

Código 3.8 Ejemplo Salida

Por último, en las últimas etapas del proyecto nos dimos cuenta de que puede ser que el alumno con mala intención o por un fallo en el código, se encontrara en un bucle infinito, el cual no habría problema gracias al tiempo máximo que hemos introducido, pero si en cada iteración se imprimiera por pantalla un texto gigante, podría colapsar el servidor y el cliente al mandar una salida de, por ejemplo, 18.000.000 caracteres de longitud como en una prueba que se realizó. Para mitigar esto, se investigó como lo realizaban otros proyectos y se optó por limitar el texto máximo que puede imprimir el programa a 7500, ya que es muy poco

probable que este valor se sobrepase en la ejecución de algún test y así evitamos que el servidor colapse por falta de recursos.

3.3 Creación de una Suite

Una Suite no es otra cosa que un conjunto de tests que tendrán una salida correcta y será la utilizada para comprobar si el alumno pasa el test o lo falla. En el caso de los tests generales, la salida se generará mediante el código de la Suite, mientras que los tests específicos se generarán mediante el código de la especificación dentro de la Suite.

A la hora de crear la Suite, existirán múltiples opciones:

- **code:** Código de la suite para los tests generales
- **title:** Título de la Suite
- **description:** Descripción de la Suite
- **content:** Explicación de la Suite compatible con Markdown
- **languages_allowed:** Lenguajes disponibles para el alumnado a la hora de ejecutar los tests
- **compile_time:** Tiempo máximo para la compilación del código
- **run_time:** Tiempo máximo para la ejecución de cada test
- **max_memory:** Máxima memoria ram a usar por usuario en la suite.

La creación de una Suite es bastante sencilla, ya que solo almacenaremos la información relacionada a esta y no se crearán tests ni especificaciones, ya que esto se creará en el panel de edición de la Suite.

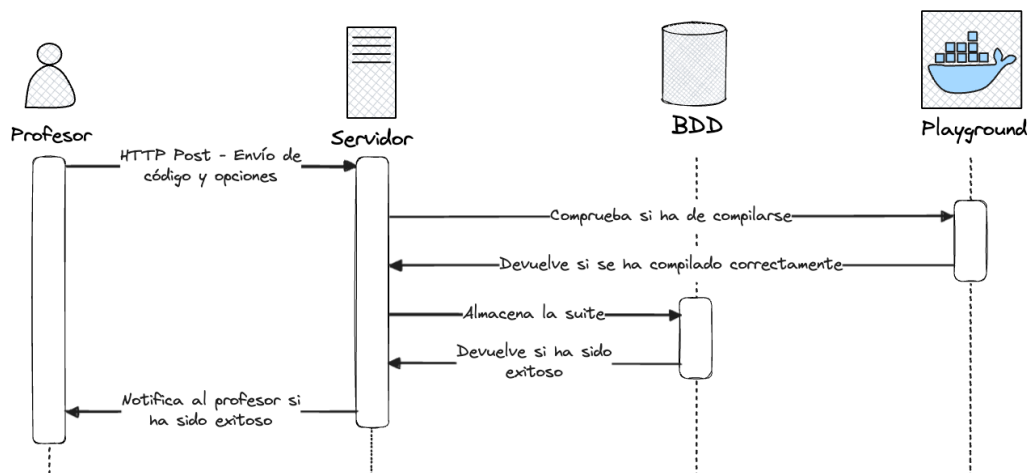


Figura 3.3 Diagrama creación de una Suite

En la creación, el cliente enviará una petición POST a la API con todas las opciones. El servidor comprobará en el caso de que sea un lenguaje compilado si compilar correctamente. Una vez comprobado, se almacena toda la información relacionada a la Suite en la base de datos para su posterior uso y se le informa al profesor de que la creación ha sido exitosa o no y si se ha guardado correctamente la información en la base de datos.

```

const supabase = getSupabase({ req })
const containerId = await runDockerInstance({ user.id, image })

let filename: string = ''
try {
  filename = await compileIfNeeded({ containerId, code, codeLanguage })
} catch (error) {
  return res.status(400).json({ msg: OTHER_ERRORS.COMPILE_ERROR })
}

const suiteId = crypto.randomUUID()
const { error } = await supabase
  .from('suites')
  .insert([[{id, title, description, code, ...}]]
if (error !== null) return res.status(400).json({ msg: SUITE_RESPONSES.NEW_ERROR })
return res.status(200).json({ msg: SUITE_RESPONSES.NEW_SUCCESS, suiteId })

```

Código 3.9 Creación de Suite (simplificado)

3.4 Creación de Tests Generales

Una vez hemos creado la Suite donde residirán los tests, se comenzará a explicar cómo se generan los tests generales y luego los específicos, ya que son muy parecidos entre sí.

Como se explicó anteriormente, los tests generales no son nada más que una estructura de datos que almacenará varios valores, siendo los más importantes los inputs, los argumentos y la salida.

```

{
  id: 1,
  title: "Factorial de 2",
  description: "Calcular el factorial de 2",
  input: [{
    has: "Introduce un número:",
    value: 2
  }],
  args: "",
  output: `Introduce un número: 2
          El factorial de este número es: 2`
}

```

Código 3.10 Ejemplo estructura Test

A la hora de crear el test al igual que la creación de la Suite, es bastante sencillo, siendo lo que ha resultado más complicado la ejecución del código, la cual se explicó anteriormente.

Primero, el profesor enviará el título, descripción, entradas, argumentos del test e identificador de la suite al servidor. La salida no debe especificarla ya que esta se obtendrá de la ejecución del código con las entradas y argumentos enviados. Después, se obtendrá del servidor tanto el código como el lenguaje en el que está hecho para su ejecución.


```

const { data: suites } = await supabaseMaster
  .from('suites')
  .select('code, code_language')
  .eq('id', suiteId)

if (suites === null || suites.length === 0) {
  return res.status(400).json({ msg: 'Suite not found!' })
}
code = suites[0].code
language = suites[0].code_language

```

Código 3.11 Obtención del código de la Suite

Una vez obtenido el código de la Suite y su lenguaje, ejecutaremos el código con los argumentos e Inputs del test y obtendremos su salida. Una vez se haya completado la ejecución, se almacenará la información del Test con su salida correspondiente.

```

const { output } = await run({ containerId, filename, args, inputs, language })
const { error } = await supabaseMaster
  .from('tests')
  .insert([
    {
      title,
      description,
      args,
      inputs,
      output,
      ...
    }
  ])
if (error !== null) {
  return res.status(400).json({ msg: TEST_RESPONSES.NEW_ERROR })
}
return res.status(200).json({ msg: TEST_RESPONSES.NEW_SUCCESS })

```

Código 3.12 Ejecución y almacenamiento del Test

3.5 Creación de Especificaciones y Tests Específicos

Las especificaciones no son otra cosa que una manera de poder evaluar una zona del código en específico sin tener que hacer una Suite solamente para esta.

Un ejemplo de utilización de especificaciones en una Suite podría ser un ejercicio en el que se pide realizar un código que pretenda resolver funciones matemáticas, donde si bien tenemos los tests para probar el código general, podemos querer tener tests más centrados en analizar esas funciones más en detalle. Los siguientes dos códigos representan el ejemplo, donde tenemos un código general que podría tener varias funciones matemáticas y el segundo donde se analiza una de esta más a fondo.

```

int pow(int n, int pow) {
    ...
}

int factorial(int n) {
    ...
}

int main(int argc, char* argv[]) {
    ...
}

```

Código 3.13 Ejemplo Especificación Código General

```

int factorial(int n) {
    ...
}

int main(int argc, char* argv[]) {
    int a;
    std::cout << "Introduce un numero: ";
    std::cin >> a;
    std::cout << "Factorial of " << a << " is: " << factorial(a) << std::endl;
    std::cout << "Factorial of " << a - 1 << " is: " << factorial(a - 1) << std::endl;
    std::cout << "Factorial of " << a - 2 << " is: " << factorial(a - 2) << std::endl;
    std::cout << "Factorial of " << a + 1 << " is: " << factorial(a + 1) << std::endl;
    std::cout << "Factorial of " << a + 2 << " is: " << factorial(a + 2) << std::endl;
    return 0;
}

```

Código 3.14 Ejemplo Especificación Código Factorial

Una vez explicadas las especificaciones, una cosa que debemos tener muy en cuenta a la hora de la implementación es que se ejecute con el mismo código que se ejecuta el código general, ya que, si no, el alumno podría crear un código para superar solamente los tests específicos y otro código para superar los tests generales.

En el proceso de desarrollar las especificaciones, se optó por que, dentro de cada Suite, existan “Especificaciones”, los cuales son códigos como podría ser el código general, pero con la diferencia de que estas especificaciones tendrán una etiqueta con la que envolver el trozo de código específico a evaluar.

```
Factorial

Bound your code between this tags

Start tag
!#@% START-UniTestCode:c137c4be-3bdf-41ba-8018-d63c1adc0bda-START !#@%
Start Bound

End tag
!#@% END-UniTestCode:c137c4be-3bdf-41ba-8018-d63c1adc0bda-END !#@%
End Bound

Load File Save file C++

1 !#@% START-UniTestCode:c137c4be-3bdf-41ba-8018-d63c1adc0bda-START !#@%
2 int factorial(int n) {
3     int result = 1;
4     for (int i = 1; i ≤ n; i++) {
5         result *= i;
6     }
7     return result;
8 }
9 !#@% END-UniTestCode:c137c4be-3bdf-41ba-8018-d63c1adc0bda-END !#@%
10
11 int main(int argc, char* argv[]) {
12     int a;
13     std::cout << "Introduce un numero: ";
```

Figura 3.4 Ejemplo Especificación Factorial

A la hora de la ejecución de los tests específicos por parte del alumno, este tendrá que envolver el trozo especificado con la etiqueta de la especificación. Al momento de ejecutarse el código, se intercambiará la zona especificada por el alumno con la zona especificada por el profesor del programa original, generando la salida con la fusión del programa original y el trozo de código del alumno. Para que esto funcione, las especificaciones tendrán su lenguaje y estos se ejecutarán solamente si el lenguaje del código del alumno coincide con el de la especificación.

La creación de la Especificación es igual a la de una Suite, solamente que este se almacena en una estructura de datos con los siguientes valores:

- **id:** Identificador de la especificación
- **code:** Código de la especificación
- **language:** Lenguaje del código de la especificación
- **title:** Título de la especificación
- **suite_id:** Suite a la que pertenece la especificación

A la hora de la creación de tests específicos, la única diferencia que se hace respecto a los tests generales es que, a la hora de la ejecución del código, este se parsea para eliminar la etiqueta que envuelve la zona a evaluar. Para que este proceso sea más fácil de realizar y no pueda dar fallo a la hora de confundirse con el código del usuario, se ha creado la etiqueta que envuelve el código de la siguiente manera.

```

// !#@% START-UniTestCode:{id}-START !#@%
{ TROZO DE CODIGO }
// !#@% END-UniTestCode:{id}-END !#@%

!#@% START-UniTestCode:c137c4be-3bdf-41ba-8018-d63c1adc0bda-START !#@%
{ TROZO DE CODIGO }
!#@% END-UniTestCode:c137c4be-3bdf-41ba-8018-d63c1adc0bda-END !#@%

```

Código 3.15 Ejemplo Etiqueta Especificación

Siendo de esta manera mucho más sencillo de remover del código, ya que solo habrá que pasarle a una función el código y el identificador de la especificación y mediante Regex se podrá obtener la zona deseada. También tener en cuenta que es prácticamente imposible que el código del usuario tenga una cadena que contenga esta etiqueta, ya que el identificador está basado en UUIDv4 y la probabilidad de que se generen dos iguales es casi nula [25].

```

const startText = (specificId: string): string => {
  return `!#@% START-UniTestCode:${specificId}-START !#@%`
}
const endText = (specificId: string): string => {
  return `!#@% END-UniTestCode:${specificId}-END !#@%`
}

export const removeSpecificFromCode = (
  { code, specificId }: { code: string, specificId: string }
): string => {
  const start = startText(specificId)
  const end = endText(specificId)
  code = code.replace(new RegExp(end), '')
  code = code.replace(new RegExp(start), '')
  return code
}

```

Código 3.16 Obtener código sin etiqueta

3.6 Ejecución de Tests

En la ejecución de tests, el alumno entrará a la Suite, donde se mostrará el título, descripción y contenido de esta, además de un editor de código para que pueda subir el código correspondiente. En el caso de que existan especificaciones para el lenguaje, la plataforma mostrará el título de esta.

A la hora de ejecutar los tests por parte del alumno, en vez de realizar una conexión unidireccional con HTTP como en el resto de la página, se utilizará una comunicación bidireccional mediante WebSockets por las razones técnicas descritas en el apartado de herramientas y tecnologías (2.1.3) y para que el usuario pueda tener actualización del estado de los tests en tiempo real. Los pasos a seguir para desarrollar esta funcionalidad fueron los siguientes:

Primero, el usuario mandará el código, su lenguaje y el identificador de la Suite en la que se encuentra al servidor. Una vez el servidor reciba esta información, este obtendrá de la suite el tiempo máximo de compilación, de ejecución y el máximo de memoria ram permitida por test.

```
const { data: suiteData } = await supabaseMaster.from('suites')
  .select('*')
  .eq('id', suiteId)

if (!suiteData) return sendTestErrors(SUITE_RESPONSES.NO_SUITES_FOUND)
const compileTime = suiteData[0].compile_time
const runTime = suiteData[0].run_time
const maxMemory = suiteData[0].max_memory
```

Código 3.17 Obtener opciones de compilación/ejecución de la BDD

Después, obtendrá todos los tests para esa Suite de la base de datos, tanto los generales como los específicos en los que el lenguaje del código del alumno coincida con el código de la especificación. Estos estarán agrupados en un objeto por el identificador de su especificación o con el atributo “general” si no pertenecen a ninguna especificación.

Posteriormente, obtendremos el código de las especificaciones junto a sus identificadores y el código general. En el código de las especificaciones, reemplazaremos el trozo a evaluar del alumno por el trozo envuelto del código de la especificación. Para ello, primero obtendremos del código del alumno, los trozos de código para cada especificación. Esto lo haremos mediante una función que irá iterando por cada identificador de la especificación y devolviendo un objeto con una propiedad y un valor, donde la propiedad contendrá el identificador de la especificación y como valor tendrá la zona del código del alumno para esa especificación. Esto lo almacenaremos en una variable `specificCodes`.

```
export const parseCode = ({ code, specifics }: ParseCodeInput): ParseCodeGrouped => {
  const result: ParseCodeGrouped = {}
  for (const specificId of specifics) {
    const start = startText(specificId)
    const end = endText(specificId)
    const regex = new RegExp(`${start}([\\s\\S]*?)${end}`, 'g')
    const match = regex.exec(code)
    if (!match) continue
    const specific = removeAllSpecifics(match[1], specifics)
    result[specificId] = specific
  }

  return result
}
```

Código 3.18 Obtener zonas específicas del alumno

Una vez hecho esto, se le comunicará al usuario mediante sockets que los tests comenzarán a ejecutarse. A la hora de comenzar a enviar el resultado de los tests al alumno, la estructura de datos que se seguirá es la siguiente:

- **id:** Identificador del test
- **output:** Salida generada con el código del alumno
- **expectedOutput:** Salida esperada por parte del test
- **passed:** Valor booleano que indica el test es válido o no.

En la ejecución, se irá iterando por cada grupo de tests donde lo primero que se comprobará es a la hora de ejecutarse un grupo de tests específicos, si el código del alumno contiene las etiquetas para su ejecución, en el caso de que no, simplemente se le informará al alumno en el resultado del tests que las etiquetas de la especificación no se encuentran en el código.

```

for(const [specificId, tests] of Object.entries(groupedTests)) {
  if (specificId !== 'general' && specificsCodes[specificId] === undefined) {
    for (const { id, output } of tests) {
      socket.emit(
        TEST_CHANNELS.FINISHED,
        { id, output: 'Specific bound not found.', expectedOutput, passed: false }
      )
    }
    continue
  }
  ...
}

```

Código 3.19 Especificación no encontrada

Luego, si es el grupo de tests generales o la etiqueta de la especificación se encuentra en el código del alumno, se irán ejecutando los tests uno a uno de manera síncrona. Inicialmente se pensó en realizarlo de manera asíncrona para mejorar la rapidez de ejecución de todos los tests. Sin embargo, durante las pruebas nos dimos cuenta de que en una suite con varios tests complejos, si hubiera varios usuarios ejecutándolos a la vez, podrían colapsar el servidor si este no es muy potente, debido principalmente a una limitación de hardware.

A la hora de la ejecución de los tests, se ejecuta el código como se ha explicado anteriormente, con el código del alumno en caso de que sea un test general o con el código del profesor y la zona a evaluar del alumno, en el caso de que sea un test específico. En el caso de que falle la compilación del código, se enviará para todos los tests relacionados con esa parte el mensaje de que ha habido un error en la compilación del código.

Para la comprobación de las salidas y si es el mismo del profesor, ambas se normalizan eliminando espacios en blanco al principio y final de cada línea para su comparación.

```

let { inputs, args, id } = test
const expectedOutput = test.output

run({ containerId, inputs, args, filename, language, timeout })
  .then(({ output }) => {
    const passed = compareOutput({ output, expectedOutput })
    const testResponse: GeneralTestResponse = { id, output, expectedOutput, passed }
    resolve(testResponse)
  })
  .catch(({ output }) => {
    console.error('Error running test with id: ', id)
    const testResponse: GeneralTestResponse = { id, output, expectedOutput, passed: false }
    resolve(testResponse)
  })

```

Código 3.20 Ejecución de tests y comprobación de salidas

Una vez se han ejecutado todos los tests, se comienzan a subir los resultados a una tabla *submits* que almacenará la hora de ejecución, el alumno y cuales ha acertado y cuales no, de esta manera se podrá analizar esta información para posteriormente identificar debilidades que estén ocurriendo o problemas comunes con el fin de mejorar el aprendizaje del alumno.

3.7 Otros aspectos desarrollados

En este apartado se comentarán más aspectos desarrollados en el proyecto menos relacionados con la lógica de la aplicación y de la ejecución creación de tests.

3.7.1 Suite con múltiples alumnos y profesores

A medida que se fue desarrollando el proyecto y se implementó la base de datos con los usuarios, se pensó la idea de que además de varios alumnos en una misma Suite, pudiese haber varios profesores en esta, pudiendo así gestionar múltiples profesores una misma Suite. Para realizar esto se ha creado en el cliente, específicamente en el panel de edición de una Suite, un botón “Manage Users” en el cual el profesor podrá insertar nuevos usuarios, tanto alumnos como profesores a la suite, además de poder eliminarlos si se quisiera.

Para realizar esto, se ha creado una estructura *suite_users*, para poder unir las tablas de usuarios y suites, ya que la relación de estas es muchos a muchos, pudiendo un usuario tener múltiples Suites y pudiendo haber múltiples usuarios para una sola Suite.

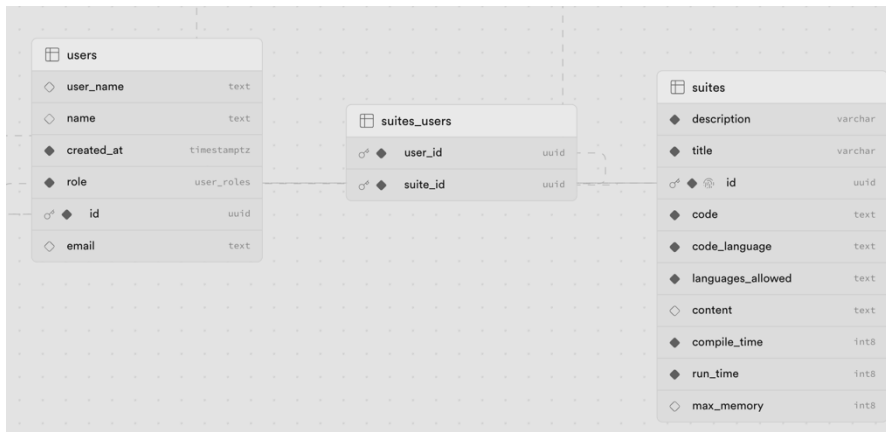


Figura 3.5 Relación Suites y Usuarios

Una vez tenemos la estructura, la forma de añadir el usuario que queremos a una Suite es tan simple como añadir un registro con el usuario y la suite a la que está registrado. El cliente mandará una petición Post al servidor y este comprobará si el usuario existe, en el caso de que exista, se añadirá/eliminará a la Suite.

```

const { suiteId } = req.params
const { userEmail } = req.body
const { data: user, error } = await supabaseMaster
  .from('users')
  .select()
  .eq('email', userEmail)
  .single()

if (error !== null || !user.id) {
  return res.status(400).json({ msg: USER_ERROR.NOT_FOUND })
}

const { error: insertError } = await supabaseMaster
  .from('suites_users')
  .insert([[suite_id: suiteId, user_id: user.id]])

if (insertError !== null) {
  return res.status(400).json({ msg: USER_ERROR.ADD_USER_ERROR })
}

return res.status(200).json(user)

```

Código 3.21 Insertar usuario en una Suite

3.7.2 Patio de juegos o *Playground*

Además de poder ejecutar suites, se ha creado un apartado de “Playground” donde el propio alumno podrá probar su código con sus entradas y argumentos, simplificando así aún más el aprendizaje del lenguaje.

La interfaz es muy sencilla, donde podrá subir el código con su lenguaje, entradas y argumentos y se mostrará la salida obtenida por el código.

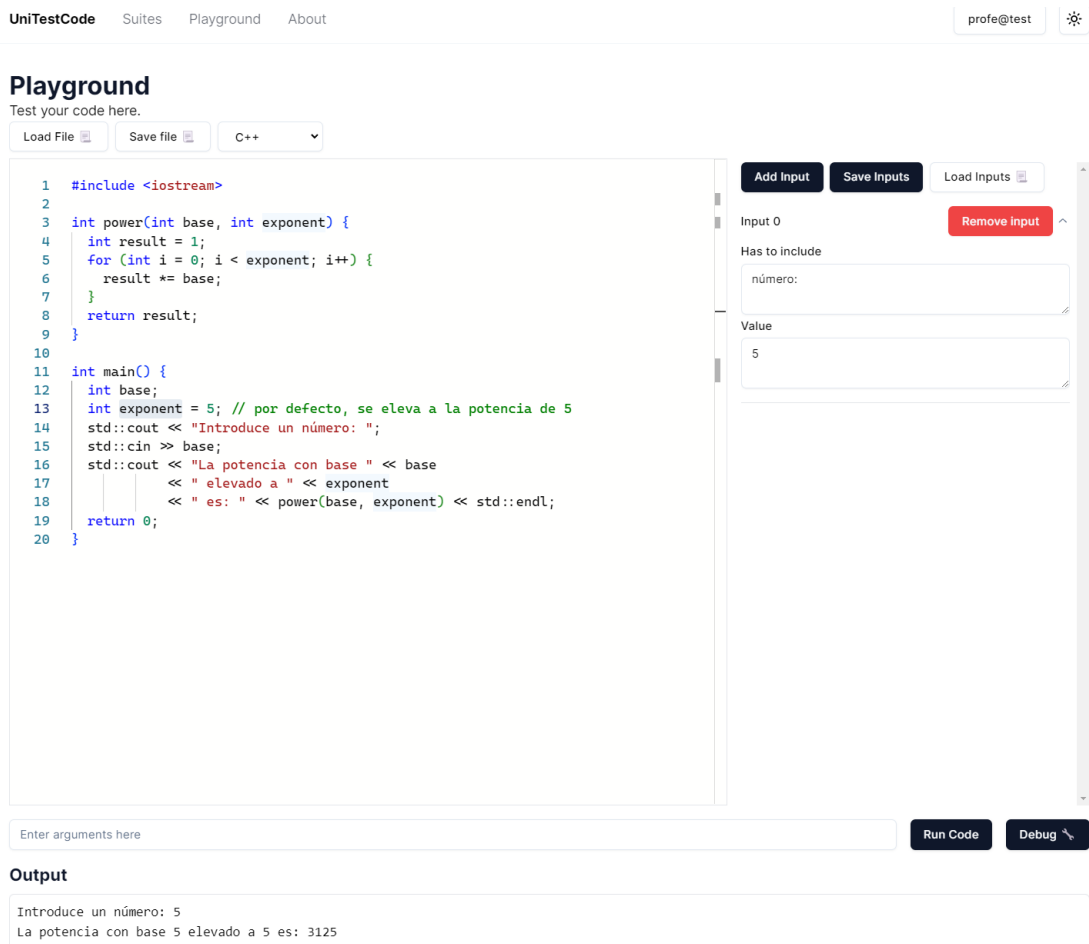


Figura 3.6 Playground

3.7.3 Modo Claro y Oscuro

Al usar tecnologías más modernas, se ha aprovechado TailwindCSS y ShadcnUI para la implementación de temas en toda la plataforma dependiendo de la preferencia del usuario, pudiendo ser un tema oscuro o claro. De esta manera, la plataforma será más accesible ya que el usuario podrá elegir el tema adaptado a las necesidades que tenga en el momento.

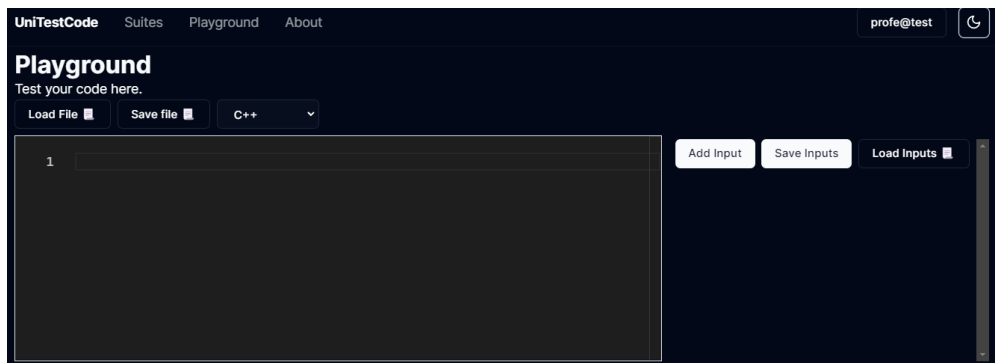


Figura 3.7 Ejemplo modo oscuro

Capítulo 4

Resultados y Guía de uso

En este capítulo se mostrarán los resultados obtenidos con este proyecto y también servirá como guía de uso para saber cómo funciona la plataforma.

4.1 Autenticación

Para autenticarnos, lo primero que debemos hacer es dirigirnos a la esquina derecha superior de la página y hacer clic en el botón “Log In”

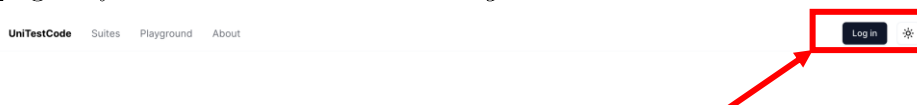


Figura 4.1 Pantalla Principal UniTestCode

Una vez dentro, podremos iniciar sesión escribiendo las credenciales o mediante GitHub con el botón “Sign in with GitHub”. También se podrá registrar haciendo click en el texto “Sign Up”.

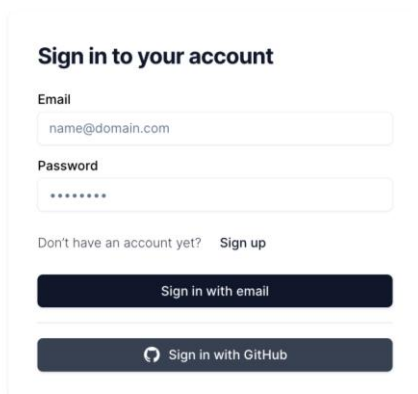
A screenshot of the 'Sign in to your account' form. It has a title 'Sign in to your account'. Below the title are two input fields: 'Email' with the placeholder 'name@domain.com' and 'Password' with a masked password '*****'. Below the password field is a link 'Don't have an account yet? Sign up'. At the bottom, there are two buttons: 'Sign in with email' and 'Sign in with GitHub'.

Figura 4.2 Inicio de Sesión

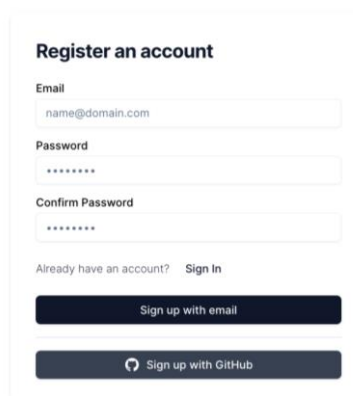
A screenshot of the 'Register an account' form. It has a title 'Register an account'. Below the title are three input fields: 'Email' with the placeholder 'name@domain.com', 'Password' with a masked password '*****', and 'Confirm Password' with a masked password '*****'. Below the confirm password field is a link 'Already have an account? Sign in'. At the bottom, there are two buttons: 'Sign up with email' and 'Sign up with GitHub'.

Figura 4.3 Creación de cuenta

4.2 Creación de Suites

Lo primero que haremos es entrar al panel de control del profesor y haremos clic en el botón de “*Create New Suite*”

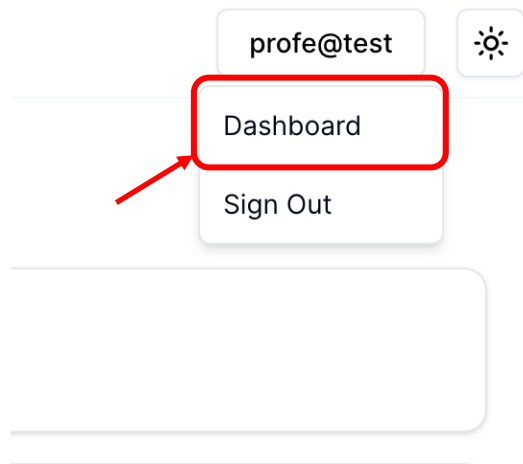


Figura 4.4 Desplegable usuario

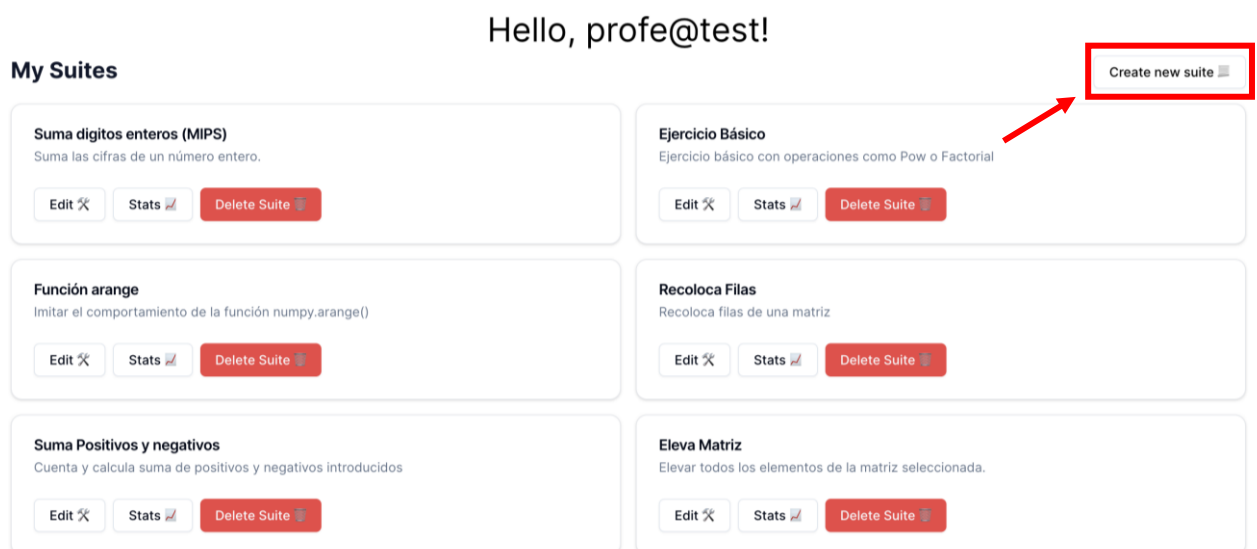



Figura 4.5 Panel de control del profesor

A la hora de la creación de suites, podremos encontrar varias opciones:

UniTestCode Suites Playground About profe@test 

Create new suite

Load File Save file C++

1

Title

Description

Content (Markdown compatible)

Languages (at least one)

C++	<input checked="" type="checkbox"/>
C	<input checked="" type="checkbox"/>
MIPS	<input checked="" type="checkbox"/>
Python	<input checked="" type="checkbox"/>
Javascript	<input checked="" type="checkbox"/>

Maximum Compilation Time (in ms)

Maximum Run Time (in ms)

Maximum Memory (in MB)

© 2024 UniTestCode by [Alexander González Covic](#)

- Código de la suite para los tests generales y el lenguaje del código
- Título de la Suite
- Descripción de la Suite
- Contenido de la Suite
- Lenguajes disponibles para el alumnado
- Tiempo máximo para la compilación del código
- Tiempo máximo para la ejecución de cada test
- Máxima memoria ram a usar por usuario.

Figura 4.6 Opciones de Suite

Una vez seleccionadas las opciones de la suite junto al código, se hará clic al botón “Create new suite” y se almacenará en la base de datos si se ha ejecutado con éxito, en otro caso, se notificará del error.

Para añadir usuarios, tanto alumnos como profesores, habrá que entrar al panel de edición de la Suite y darle al botón “*Manage Users*”

Hello, profe@test!

My Suites

Create new suite

Suma dígitos enteros (MIPS)
Suma las cifras de un número entero.

Ejercicio Básico
Ejercicio básico con operaciones como Pow o Factorial

Función arange
Imitar el comportamiento de la función numpy.arange()

Recoloca Filas
Recoloca filas de una matriz

Suma Positivos y negativos
Cuenta y calcula suma de positivos y negativos introducidos

Eleva Matriz
Eleva todos los elementos de la matriz seleccionada.

Figura 4.7 Ir al panel de edición de la Suite

Suma dígitos enteros (MIPS)

14c7e776-a91d-4b68-b49a-a7b122bba845

Edit Suite Manage Users

Create Test Show Code

General Tests

0 termina la ejecución

Edit Show Output Delete Test

Numeros pequeños

Edit Show Output Delete Test

Specific Tests

Create Specification

Figura 4.8 Ir a la gestión de usuarios

Una vez aquí adentro, se podrá tanto añadir un usuario escribiendo el correo y presionando “Add User” o eliminar uno que se encuentre en la Suite con el botón “remove”

Manage Users

Add new user

alumno2@test

Add User

Sort by email Sort by role

Email	Role	Actions
profe@test	teacher	Remove
alumno@test	student	Remove

Figura 4.9 Panel de gestión de usuarios

4.3 Creación de Tests Generales

Dentro del panel de control del profesor, entraremos a la edición de la suite y haremos clic en el botón de “*Create Test*”

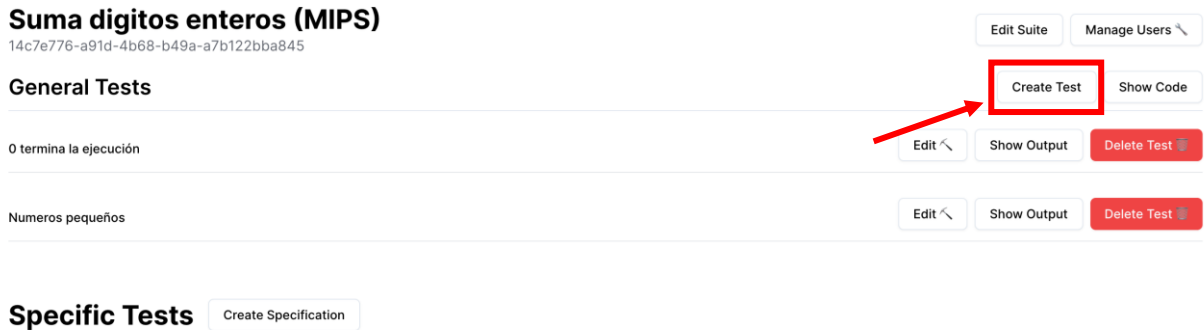


Figura 4.10 Ir al panel de creación de Tests Generales

Una vez dentro, nos encontraremos con varias opciones a la hora de la creación de tests

Create Test

Title

Description

Args

Input 0

- Título
- Descripción
- Argumentos
- Entradas

Figura 4.11 Panel de creación de Tests

Para introducir las entradas, se podrá introducir tanto de manera manual, como cargándolo desde un fichero de configuración:

Input 0

Has to include	Value
<input type="text" value="Introduce un número:"/>	<input type="text" value="4"/>

Figura 4.12 Módulo de entradas

La estructura que deberá tener el fichero de configuración de los inputs ha de ser como el siguiente:

```
[
{
  "has": "Introduce un número:",
  "value": "4"
},
{
  "has": "Introduce otro:",
  "value": "5"
}
]
```

Código 4.1 Ejemplo fichero de configuración entradas

4.4 Creación de Tests Específicos

Para especificar la zona a evaluar por los tests, dentro del panel de edición de la Suite, se hará clic en el apartado “*Create Specification*”

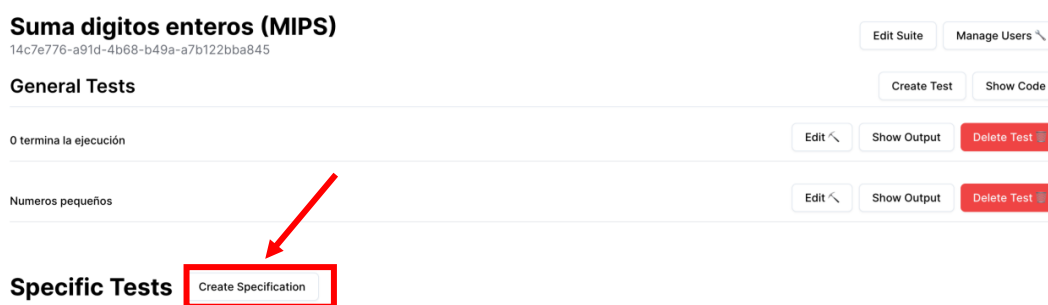


Figura 4.13 Ir a crear una especificación

Una vez en dentro, deberemos poner el código para probar esa zona en específico, con su lenguaje y con la zona a evaluar encapsulada por dos tags, uno de apertura y otro de cierre.

Create Specification

Insert the specification name

Bound your code between this tags

Start tag
!#@% START-UnitTestCode:d2c01880-6120-4dd5-9f18-616639f4f987-START !#@%

Start Bound

End tag
!#@% END-UnitTestCode:d2c01880-6120-4dd5-9f18-616639f4f987-END !#@%

End Bound

Figura 4.14 Menú de creación de especificación

Un ejemplo para la creación de una especificación para una función *Pow* dentro del código sería la siguiente:

```
Load File Save file C++
2
3 !#@% START-UniTestCode:d2c01880-6120-4dd5-9f18-616639f4f987-START !#@%
4 int pow(int n, int pow) {
5     int result = 1;
6     for (int i = 0; i < pow; i++) {
7         result *= n;
8     }
9     return result;
10 }
11 !#@% END-UniTestCode:d2c01880-6120-4dd5-9f18-616639f4f987-END !#@%
12
13 int main(int argc, char* argv[]) {
14     int a;
15     std::cout << "Introduce un numero: ";
16     std::cin >> a;
17     std::cout << "Power of " << a << " to the power of 3 is: " << pow(a, 3) << std::endl;
18     std::cout << "Power of " << a << " to the power of 4 is: " << pow(a, 4) << std::endl;
19     std::cout << "Power of " << a << " to the power of 5 is: " << pow(a, 5) << std::endl;
20     std::cout << "Power of " << a << " to the power of 6 is: " << pow(a, 6) << std::endl;
21     return 0;
22 }
```

Create specification Cancel

Figura 4.15 Ejemplo especificación

Una vez creada la especificación, para crear los tests será de manera similar a los tests generales, solamente que esta vez, le daremos al botón de “*Create Test*” situado al lado de la especificación.

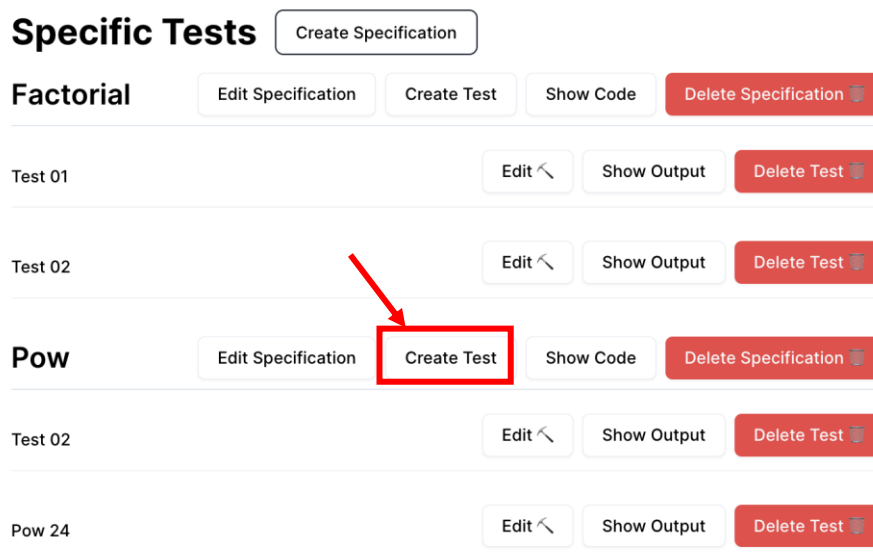


Figura 4.16 Ir al panel de creación de Test Específico

4.5 Gestión de usuarios a una Suite

Dentro del panel de edición de una Suite, daremos clic al botón “*Manage Users*” y añadiremos o eliminaremos al usuario en cuestión.

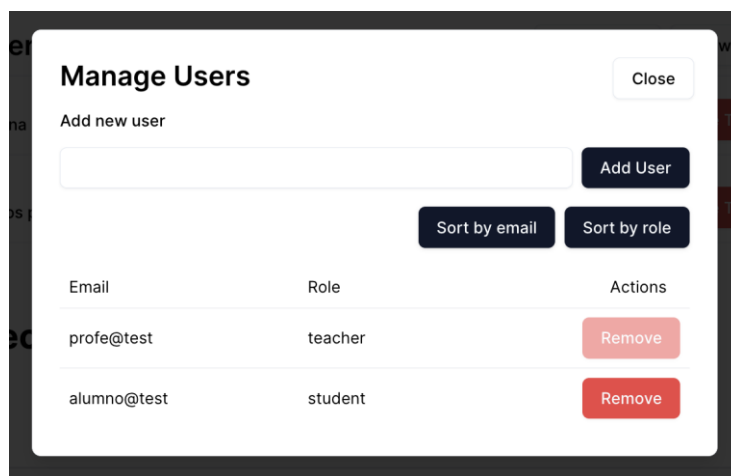
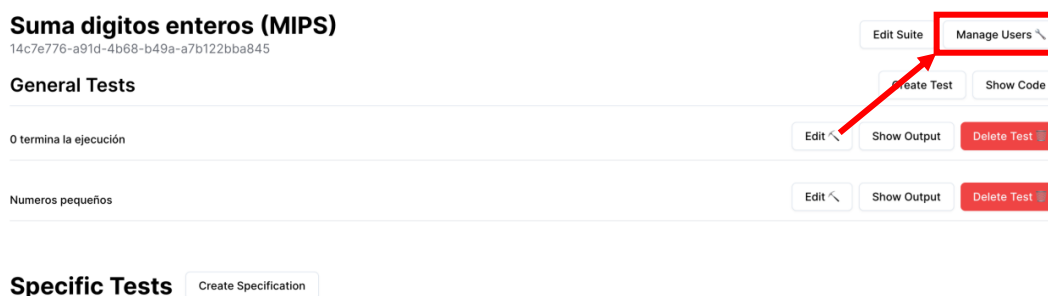


Figura 4.17 Panel de gestión de usuarios

4.6 Ejecución de Tests

Una vez creado la Suite y añadido el alumnado, este podrá acceder a la Suite en cuestión desde el apartado Suites

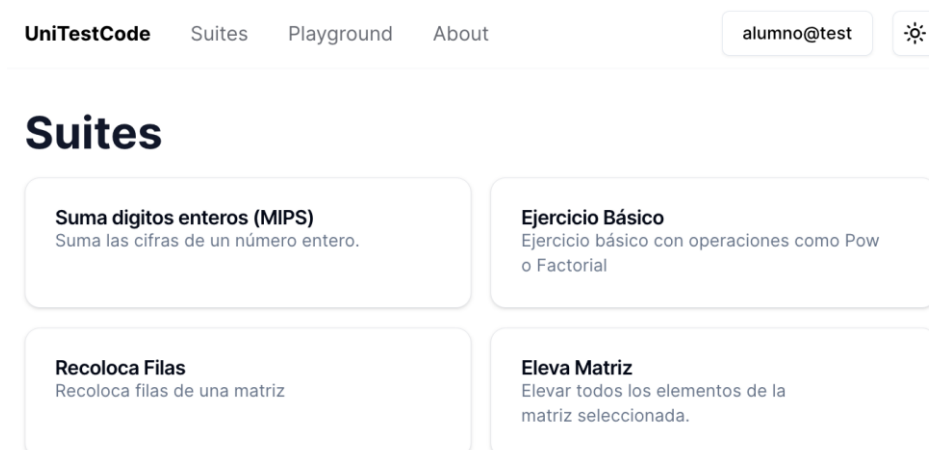


Figura 4.18 Apartado de Suites

Ejercicio Básico

Ejercicio: Funciones Matemáticas

Implementa un programa en C++ que incluya dos funciones matemáticas: una para calcular la potencia de un número y otra para calcular el factorial. A continuación, se proporciona un esbozo del código que deberás completar:

```
#include <iostream>

// Completa la función para calcular la potencia de un número
int pow(...) {
    // TODO: Tu código aquí
}

// Completa la función para calcular el factorial de un número
int factorial(...) {
    // TODO: Tu código aquí
}

int main(int argc, char* argv[]) {
    int a;

    // Solicita al usuario que introduzca un número
    std::cout << "Introduce un numero: ";
    std::cin >> a;
```

Figura 4.19 Ejemplo dentro de una Suite

Una vez el alumno esté dentro de la Suite, podrá subir el código y ejecutar los tests. A medida que se van ejecutando los tests, se mostrará si se ha ejecutado correctamente o no, además de la salida, la salida esperada y un botón para ver las diferencias entre ambos.

The screenshot shows a test execution interface. At the top, there is a test labeled 'Numero 5' with a green checkmark icon and a dropdown arrow. Below it, another test labeled 'Numero 10' is shown with a red 'X' icon and an upward arrow. The 'Numero 10' test is expanded to show a comparison between 'Output' and 'Expected Output'. The 'Output' box contains the text: 'Introduce un numero: 10', 'Power of 10 to the power of 3', and 'Factorial of 10 is: -1'. The 'Expected Output' box contains: 'Introduce un numero: 10', 'Power of 10 to the power of 3', and 'Factorial of 10 is: 3628800'. A 'View Diff' button is positioned between the two boxes.

Figura 4.20 Ejemplo ejecución tests

The screenshot shows a 'General Tests' dialog box. It contains a text area with the following text: 'Introduce un numero: 10', 'Power of 10 to the power of 3 is: 1000', and 'Factorial of 10 is: -13628800'. The value '-13628800' is highlighted in green. Below the text area is a 'Close' button.

Figura 4.21 Ejemplo diferencias de salida

4.7 Obtención de estadísticas

Para obtener las estadísticas del alumnado en cada Suite, se ha de entrar en el apartado de “Stats” de la Suite en específico en el panel de control del profesor. Una vez ahí dentro se podrá ver una vista simplificada de las estadísticas o descargarse los datos en crudo con más información para un mayor análisis.

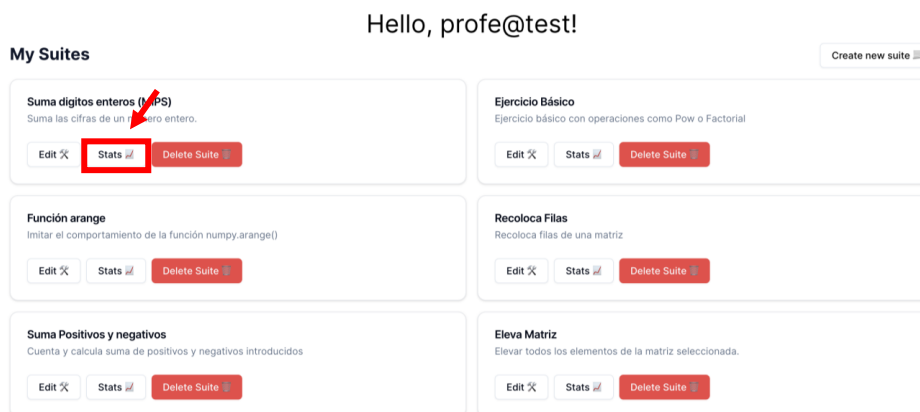
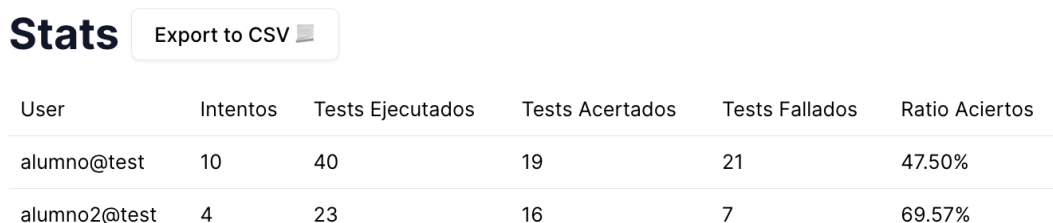


Figura 4.22 Ir al panel de estadísticas de la Suite

The image shows the "Stats" panel with an "Export to CSV" button. Below the button is a table with the following data:

User	Intentos	Tests Ejecutados	Tests Acertados	Tests Fallados	Ratio Aciertos
alumno@test	10	40	19	21	47.50%
alumno2@test	4	23	16	7	69.57%

Figura 4.23 Panel de estadísticas

En el caso de que exportemos los datos a CSV, obtendremos una tabla más detallada donde cada fila será un intento del usuario.

created_at	success	email	test_title	specifics_title	suite_title
"2024-05-05 08:55:36.805319+00"	true	alumno@test	"Numero 10"	null	"Ejercicio Básico"
"2024-05-05 08:55:36.805319+00"	false	alumno@test	"Numero 5"	Pow	"Ejercicio Básico"

Tabla 4.1 Ejemplo fichero estadísticas.csv

Capítulo 5

Conclusiones y líneas futuras

Para concluir, en el proyecto se logró cumplir todos los objetivos propuestos, desde los primeros que se pensaron a la hora de comenzar el proyecto, hasta los que se fueron incluyendo a lo largo del desarrollo. Entre las tareas logradas a lo largo del proyecto, podemos ver las siguientes:

- Se ha implementado un sistema seguro para la ejecución de código no seguro mediante la creación de un *sandbox* personalizado hecho con Docker.
- Se ha diseñado una base de datos para la gestión de usuarios y su autenticación, además de almacenar información relacionada con las suites, tests, especificaciones y la relación con los usuarios.
- Se ha diseñado una interfaz con NextJS, TailwindCSS, ShadcnUI y React para el uso de la plataforma de una manera gráfica y accesible.
- Se ha realizado el backend mediante NodeJS, Express y TypeScript para la creación de la API del proyecto y la gestión de los recursos.
- Se ha implementado Socket.IO como librería de WebSockets para la comunicación entre el cliente y el servidor a la hora de ejecutar los tests y poder notificar en tiempo real al cliente del estado de ejecución.
- Se ha implementado un sistema de tests donde el alumnado pueda subir el código contra los tests y comprobar si la solución es válida y en el caso de que no lo sea, las diferencias a lo esperado.
- Se ha implementado un sistema para poder verificar zonas específicas del código del alumnado mediante especificaciones.
- Se ha implementado el modo oscuro para mejorar la accesibilidad y la visión de la plataforma.

Como líneas futuras para el proyecto, si bien podría estar en producción, hay varias mejoras o cambios que se podrían llegar a hacer en un futuro.

1. **Implementar una forma más visual de especificar los trozos que se evaluarán** en los tests, de modo que, aunque los tags para encapsular las zonas se introduzcan en el código, visualmente no ensucien el código para el usuario, aunque en la práctica se envíe el código "*sucio*" al backend.
2. **Implementar que se puedan reordenar las entradas** a la hora de crear un test o en el *Playground* de una manera más sencilla, como pudiera ser arrastrándolo en vez de tener que eliminar o subir un fichero.

3. **La ejecución de los tests en paralelo**, ya que como bien se comentó anteriormente, se acabó optando por la ejecución de los tests de manera secuencial debido a la falta de recursos y aumentando el tiempo de la ejecución por ello.
4. **Integración de la Inteligencia Artificial** a la hora de poder evaluar otros aspectos del código como pueden ser:
 - **Calidad del código:** Utilizar la IA para analizar el código y proporcionar una puntuación de la calidad del mismo en base a buenas prácticas, legibilidad, etc.
 - **Detectar plagio:** Utilizar la IA para poder detectar el plagio de diferentes códigos.

Capítulo 6

Summary and Conclusions

To conclude, the project successfully achieved all the proposed objectives, from those initially planned at the start of the project to those added throughout its development. The accomplished tasks include:

- A secure system for the execution of non-secure code has been implemented by creating a custom sandbox made with Docker.
- A database has been designed for user management and authentication, in addition to storing information related to suites, tests, specifications and the relationship with users.
- An interface has been designed with NextJS, TailwindCSS, ShadcnUI and React for the use of the platform in a graphical and accessible way.
- The backend has been realized using NodeJS, Express and TypeScript for the creation of the project's API and resource management.
- Socket.IO has been implemented as a WebSockets library for the communication between the client and the server when executing the tests and to be able to notify the client in real time of the execution status.
- A test system has been implemented where the students can upload the code against the tests and check if the solution is valid and in case it is not, the differences to what is expected.
- A system has been implemented to be able to verify specific areas of the student's code by means of specifications.
- Dark mode has been implemented to improve the accessibility and viewability of the platform.

As future lines for the project, although it could be in production, there are several improvements or changes that could be made in the future.

- 1. Implement a more visual way of specifying the chunks to be evaluated in the tests**, so that, although the tags to encapsulate the zones are introduced in the code, visually they do not dirty the code for the user, although in practice the "dirty" code is sent to the backend.
- 2. Implement that the inputs can be reordered** when creating a test or in the *Playground* in a simpler way, such as dragging it instead of having to delete or upload a file.

3. **The execution of the tests in parallel**, since as it was commented previously, the tests were executed sequentially due to the lack of resources and therefore increasing the execution time.
4. **Integration of Artificial Intelligence** in order to evaluate other aspects of the code, such as:
 - **Code quality:** Use AI to analyze the code and provide a code quality score based on best practices, readability, etc.
 - **Plagiarism detection:** Use AI to detect plagiarism of different codes.

Capítulo 7

Presupuesto

Todas las tecnologías utilizadas en este proyecto son código libre y son hospedados todos por el servidor, donde si bien en el caso de la tecnología Supabase, para este proyecto se está utilizando la API, se podría hospedar de manera gratuita en el servidor.

Por ello, para el cálculo del presupuesto, se tendrá en cuenta principalmente las horas de trabajo en cosas como el diseño, prototipos e informe final y de recursos materiales empleados en el proyecto.

La duración del proyecto en una jornada de 40 horas semanales aproximadamente sería de unas 10 semanas: $10 \text{ semanas} * 40 \text{ horas/semana} = 400 \text{ horas}$. Por otra parte, el coste por hora será de unos 12,50 eur/hora.

- **Diseño de la plataforma:** El diseño tarda aproximadamente una semana al ser un diseño simple, pero teniendo varias cosas en cuenta: $40 \text{ horas} * 12,50 \text{ eur/hora} = 500\text{€}$
- **Desarrollo del proyecto:** El desarrollo del proyecto es lo que más tardará, tardando aproximadamente 7 semanas teniendo en cuenta el diseño de la arquitectura y toda su implementación: $7 \text{ semanas} * 40 \text{ horas/semana} * 12,50 \text{ eur/hora} = 3500\text{€}$
- **Pruebas del proyecto:** La duración de las pruebas son de aproximadamente dos semanas para detectar fallos y arreglar posibles errores de seguridad que existan: $2 \text{ semanas} * 40 \text{ horas/semana} * 12,50 \text{ eur/hora} = 1000\text{€}$
- **Servidor:** El coste del servidor dependerá del proveedor, ya que necesitamos que tenga compatibilidad con WebSockets. Para poder cumplir con las especificaciones del proyecto, dependerá de la cantidad de usuarios, pero aproximadamente $\geq 50\text{€}$ mensuales si se utilizan proveedores como AWS o DigitalOcean.

Coste	Tipo	Descripción
500€	Diseño de la plataforma	Diseño de la plataforma web con su interfaz gráfica
3500€	Desarrollo del proyecto	Desarrollo del backend, del frontend y de toda la implementación del proyecto

1000€	Pruebas del proyecto	Pruebas realizadas para comprobar fallos o problemas de seguridad del proyecto.
$\geq 50€$	Servidor	Servidor donde se hospedar� la aplicaci�n.

Tabla 7.1 Presupuesto

El total del proyecto ascender  a 5000€ como coste inicial sin contar el coste mensual del servidor. Si estimamos que la plataforma la utilizar n en diversos grados y asumimos ~400 alumnos, el coste por alumno ser  de aproximadamente $5000€ / 400 = 12,50€$

Bibliografía

- [1] «LeetCode,» [En línea]. Available: leetcode.com.
- [2] «Exercism,» [En línea]. Available: exercism.org.
- [3] «HackerRank,» [En línea]. Available: hackerrank.com.
- [4] A. M. Azahari, A. Ahmad, S. B. Rahayu y M. H. M. Halip, «CheckMyCode: Assignment Submission System with Cloud-Based Java Compiler,» *8th International Conference on Information Technology and Multimedia*, 2020.
- [5] «NodeJS,» [En línea]. Available: <https://nodejs.org/>.
- [6] «ExpressJS,» [En línea]. Available: expressjs.com.
- [7] «TypeScript,» [En línea]. Available: typescriptlang.org.
- [8] «What is long polling?,» PubNub, [En línea]. Available: pubnub.com/guides/long-polling/.
- [9] «Long Polling vs WebSockets,» svix, [En línea]. Available: svix.com/resources/faq/long-polling-vs-websockets/.
- [10] «Socket.IO,» [En línea]. Available: socket.io.
- [11] «Docker,» [En línea]. Available: docker.com.
- [12] «AstroJS,» [En línea]. Available: astro.build.
- [13] «NextJS,» [En línea]. Available: <https://nextjs.org/>.
- [14] «React,» [En línea]. Available: react.dev.
- [15] «TailwindCSS,» [En línea]. Available: tailwindcss.com.
- [16] «ShadcnUI,» [En línea]. Available: ui.shadcn.com.
- [17] «Supabase,» [En línea]. Available: supabase.com.
- [18] «PostgreSQL,» [En línea]. Available: postgresql.org.
- [19] «MySQL,» [En línea]. Available: mysql.com.
- [20] «Firebase,» [En línea]. Available: firebase.google.com.

- [21] A. G. Covic, «UniTestCode,» [En línea]. Available: github.com/Universidad-de-La-Laguna/TFG-Alexander-Testeos.
- [22] «Judge0 CE,» [En línea]. Available: ce.judge0.com.
- [23] N. PrasannaHN, «Building a secure/sandboxed environment for executing untrusted code,» [En línea]. Available: dev.to/narasimha1997/building-a-secure-sandboxed-environment-for-executing-untrusted-code-7e8.
- [24] «NodeJS Child Process,» [En línea]. Available: nodejs.org/api/child_process.
- [25] J. Hall, «What are the odds?,» [En línea]. Available: jhall.io/archive/2021/05/19/what-are-the-odds/.