Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Plataforma de evaluación automática para programación

Automatic Grading Platform for Programming

José Orlando Nina Orellana

La Laguna, *11* de *Julio* de 2024

D. **Alberto Hamilton Castro**, con N.I.F. 43.773.884-P profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor.

D. **Pedro Antonio Toledo Delgado**, con N.I.F. 45.725.874-B profesor Contratado Doctor adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutor.

C E R T I F I C A N

Que la presente memoria titulada:

“Plataforma de evaluación automática para programación”

ha sido realizada bajo su dirección por D. **José Orlando Nina Orellana**,

con N.I.F. 42.281.439-X.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 11 de julio de 2024.

Agradecimientos

Doy las gracias a mis tutores Alberto Hamilton Castro y Pedro Antonio Toledo Delgado por la oportunidad y por su ayuda durante la realización del proyecto.

Agradecer a mi familia por el apoyo que me han brindado durante esta etapa universitaria, especialmente a mi hermano, que me ayudó a decidirme por esta carrera.

También quiero agradecer a mi amigo José por acompañarme durante esas interminables clases y sacarme unas risas que hicieron todo más llevadero.

A mi pareja, gracias por darme este último empujón necesario.

Mención especial también a Alejandro Miguel Taboada Sánchez, cuyos vídeos de youtube me ayudaron al inicio de la carrera. Siempre recordaré su mítica frase: “Si puedes imaginarlo, puedes programarlo”.

Licencia

Resumen

Este proyecto busca automatizar el proceso de evaluación de ejercicios de programación, beneficiando al profesorado y al alumnado reduciendo tiempos, proporcionando retroalimentación más frecuente y permitiendo nuevas metodologías de enseñanza de la programación. La evaluación manual de código puede llegar a requerir gran cantidad de tiempo, errores humanos y demora en su entrega. Se pretende alcanzar este objetivo utilizando herramientas ya existentes como solución parcial al problema, adaptándolas para que se puedan comunicar y de esta forma crear un sistema completo. Se utiliza PrairieLearn para la administración de cursos y tareas, además se personaliza para realizar una evaluación automática, lanzando tests sobre código enviados por alumnos. Como generador de tests, se usa UTBotCpp, que utiliza la ejecución simbólica para generar tests unitarios para C/C++. Ambas herramientas se unen mediante una API haciendo posible su comunicación. Además, se probaron con el propósito de generación de tests sistemas de IA generativa.

Palabras clave: programación, test unitarios, evaluación, Docker, PrairieLearn, UTBotCpp.

Abstract

This project seeks to automate the evaluation process of programming exercises, benefiting teachers and students by reducing time, providing more frequent feedback and enabling new programming teaching methodologies. Manual code evaluation can be time-consuming, require human error and delay delivery. The goal is to achieve this objective by using existing tools as a partial solution to the problem, adapting them so that they can communicate and thus create a complete system. PrairieLearn is used for the administration of courses and assignments, and it is also customized to perform an automatic evaluation, launching tests on code submitted by students. UTBotCpp, which uses symbolic execution to generate unit tests for C/C++, is used as a test generator. Both tools are linked through an API making their communication possible. In addition, generative AI systems were tested for the purpose of test generation.

Keywords: programming, unit testing, evaluation, Docker, PrairieLearn, UTBotCpp.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En un mundo cada vez más tecnológico, la programación se ha convertido en una habilidad esencial en el ámbito educativo. Independientemente del campo de interés, aprender a programar puede proporcionar nuevas oportunidades. En muchas ocasiones se afirma que la mejor manera de aprender algo es a través de prueba y error, y la programación no es una excepción. Aplicando los conocimientos a la práctica mediante la resolución de problemas prácticos o ejercicios es una de las mejores formas.

En la docencia evaluar el código hecho por estudiantes puede ser una tarea compleja y tediosa para el profesorado. Esta labor puede llegar a conllevar una cantidad de tiempo significativa, fallos humanos que puedan suponer una evaluación incorrecta, y una demora en recibir una retroalimentación que impida a los alumnos corregir sus errores [1]. Esto podría obstaculizar el progreso de los estudiantes y reducir su motivación. Además, añadiendo una retroalimentación automática, podría dar al profesorado la oportunidad de enfocarse en otros aspectos de la enseñanza.

Actualmente existen varias plataformas como Exercism, freeCodeCamp o Codecademy, que ayudan aprender a programar o preparar pruebas técnicas de cara a encontrar trabajo. Exercim está diseñada para mejorar habilidades de programación a través de ejercicios prácticos y mentorías. FreeCodeCamp es una organización educativa sin ánimo de lucro que proporciona contenido gratuito en forma de artículos y vídeo tutoriales en su canal de youtube. Codeacademy es una plataforma interactiva que ofrece clases de pago que incluyen vídeos y proyectos prácticos basados en casos reales.

Pocas de estas plataformas proporcionan planes para crear clases online propias destinadas a la docencia, y las que proporcionan no contemplan una funcionalidad para la generación de pruebas automáticas.

1.1 Objetivos

El proyecto tiene dos objetivos principales. Primero, proporcionar a los alumnos un medio por el cual puedan recibir retroalimentación sobre su código para saber si se ajusta correctamente con los requisitos del ejercicio y puedan corregir posibles errores. Segundo, automatizar para el profesorado todo este proceso de evaluación, enfocados en la creación de pruebas y la corrección de código. Para alcanzar estos objetivos, se han definido los siguientes objetivos específicos:

1. Ofrecer una interfaz amigable para que los alumnos puedan subir su código.
2. Automatizar la evaluación de código de los alumnos. Utilizar un sistema que ejecute las pruebas en el código de los alumnos.
3. Proporcionar retroalimentación a los alumnos en función de las pruebas realizadas. Incluir un módulo capaz de dar una puntuación en tiempo real.
4. Proporcionar una herramienta para la generación automática de pruebas. Implementar un sistema capaz de analizar la solución de un ejercicio y generar pruebas.

Capítulo 2

Herramientas

2.1 Búsqueda de herramientas y selección

En esta fase, se realizó una exploración y evaluación de diversas herramientas tecnológicas podrían utilizarse para lograr los objetivos del proyecto. Se definió unos criterios para ayudar a realizar un filtrado:

- **Gratuito:** determinar el uso de la herramienta implica algún coste económico.
- **Documentación:** verificar que exista información y recursos disponibles que faciliten el uso de la herramienta tanto para los desarrolladores como para los alumnos. Se hizo en una escala del 1 al 10 en función del número de resultados por internet, siendo 1 menos de 100 resultados y 10 si llega a alcanzar más de 100.000.
- **Actualizaciones:** evaluar si la herramienta está recibiendo actualizaciones de forma activa. Esto significa un soporte continuo, mejora y corrección de errores.
- **Código abierto:** comprobar si la herramienta es de código abierto, lo que permite una personalización y facilita la contribución de la comunidad para su mejora.
- **Experiencia previa**: se refiere a que la herramienta está desarrollada en lenguajes en los cuales tengamos algún conocimiento con anterioridad.

Se realizó una búsqueda en internet, lo cual resultó en la siguiente lista de herramientas:

- **API Sanity Checker:** un generador automático de pruebas unitarias básicas para librerías en C/C++.
- **Splint:** una herramienta para comprobar estáticamente programas en C en busca de vulnerabilidades de seguridad y errores de codificación.
- **UTBotCpp:** herramienta diseñada para generar pruebas para C/C++

automáticamente buscando todas las rutas de ejecución que puede tomar un programa.

- **AutoGradr:** es una plataforma con IDE web que automatiza la calificación de cursos de ciencias de la computación.
- **PrairieLearn:** un sistema de aprendizaje en línea basado en problemas para crear tareas y exámenes. Permite gestionar tareas como el dibujo gráfico, el álgebra simbólica y la compilación y ejecución de código del alumno.

El la tabla 2.1 se incluye una lista con las herramientas encontradas y comparándolas en una tabla según los criterios declarados antes:

Tabla 2.1: Comparación de herramientas

Herramienta	Gratuito	Documentación (1-10)	Última actualización	Código Abierto	Experiencia previa
API Sanity Checker	Sí	3	08-09-2015	Sí	Sí
Splint	Sí	2	14-11-2010	Sí	Sí
UTBotCpp	Sí	6	26-03-2024	Sí	Sí
AutoGradr	No	7	-	No	-
PrairieLearn	Sí	9	15-05-2024	Sí	No

Además de este filtrado, cada una de la herramientas se descargaron y se intentó ejecutar siguiendo los pasos de la documentación. Splint no se pudo ejecutar, ya que dependendía de módulos externos que ya no daban soporte. En cuanto a AutoGradr, al no ser un herramienta gratis, se tuvo que enviar un correo para ponerse contacto y no se obtuvo respuesta. Con respecto a las demás, sí se pudieron ejecutar y realizar algunas pruebas.

Tras las búsqueda y en función de las pruebas realizadas, se eligió PrairieLearn cómo portal para los alumnos por la extensa documentación en su página oficial y sus

tutoriales en youtube, así como por las actualizaciones activas y la capacidad de personalización. UTBotCpp se seleccionó como generador de tests por experiencia previa con el lenguaje de programación, ser de código abierto y tener actualizaciones frecuentes.

2.2 UTBotCpp

UtbotCpp [2] es una herramienta diseñada para la generación automática de pruebas unitarias para programas en C/C++. Puede utilizarse como una extensión de entornos de desarrollo como Visual Studio Code o CLion. Funciona como un envoltorio sobre el motor de ejecución simbólica KLEE.

La ejecución simbólica [3] consiste en analizar un programa en busca de obtener todos los caminos de ejecución que puede tomar el programa. Esto se consigue emulando su ejecución y utilizando símbolos en lugar de valores concretos como entradas.

UtbotCpp automatiza todo el proceso de preparación del entorno y generación de pruebas que requiere KLEE. Simplifica la creación de casos de prueba en formato Google Test, incluyendo las entradas.

2.2.1 Ejemplo

Usando la versión de demostración disponible en su página oficial, se pasó el código presente en la figura 2.1. Este tiene tres posibles caminos de ejecución: cuando `num` es menor que diez, cuando está entre diez y quince, o cuando es mayor que quince.

```
int check_num(int num) {
    if (num > 15) return 1;
    if (num > 10) return 2;
    return -1;
}
```

Figura 2.1: Ejemplo de código para probar la generación de tests

Como resultado dio el siguiente archivo como se ve en la figura 2.2, con los siguientes tests probando los tres caminos nombrados anteriormente:

```
TEST(regression, check_num_test_1)
{
    int actual = check_num(536870912);
    EXPECT_EQ(1, actual);
}

TEST(regression, check_num_test_2)
{
    int actual = check_num(11);
    EXPECT_EQ(2, actual);
}

TEST(regression, check_num_test_3)
{
    int actual = check_num(0);
    EXPECT_EQ(-1, actual);
}
```

Figura 2.2: Tests de ejemplos generados

2.3 PrairieLearn

PrairieLearn [4] es un software de código abierto para mejorar experiencias de aprendizaje y evaluaciones a los estudiantes. Esta herramienta, fue desarrollada hace diez años en la Universidad de Illinois, tiene como objetivo liberar carga de trabajo al profesorado y facilitar el aprendizaje al alumnado, mediante la automatización de las tareas rutinarias de calificación y administración.

En cuanto a las tecnologías que utilizan, se estandarizan en JavaScript/TypeScript (Node.js) y SQL (PostgresSQL) como lenguajes de implementación, intentando reducir el número de librerías o frameworks.

2.3.1 Descripción de la interfaz de usuario

A continuación, en las figuras 2.3 y 2.4, se pueden observar las principales

pantallas en función del rol del usuario, ya sea alumno o profesor. Después de iniciar sesión, ambos usuarios tienen una pantalla de inicio similar.

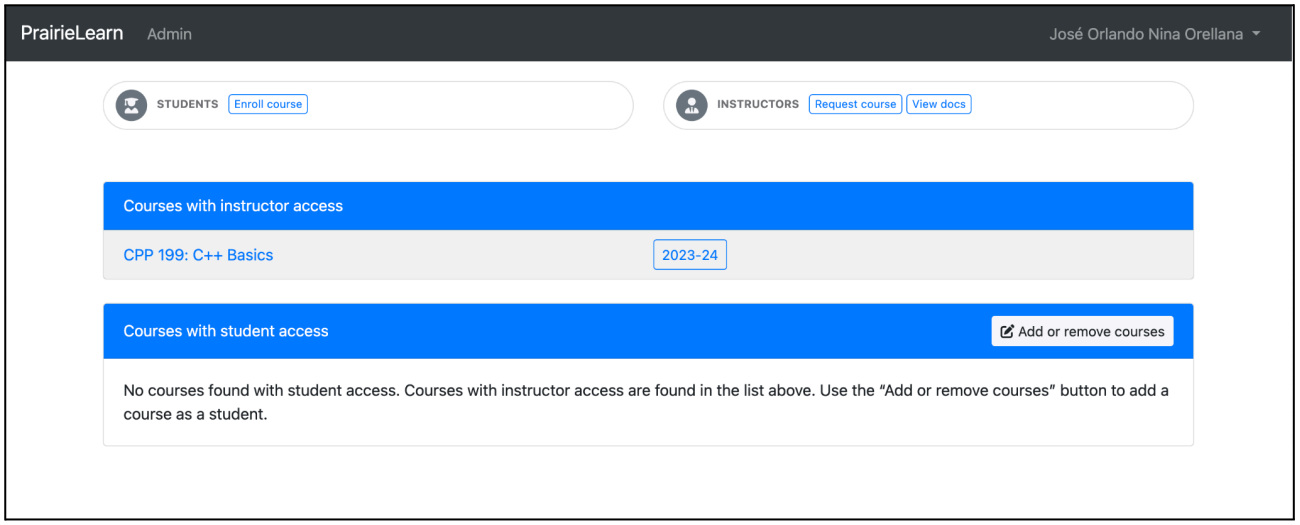

Figura 2.3: Pantalla de inicio de profesores

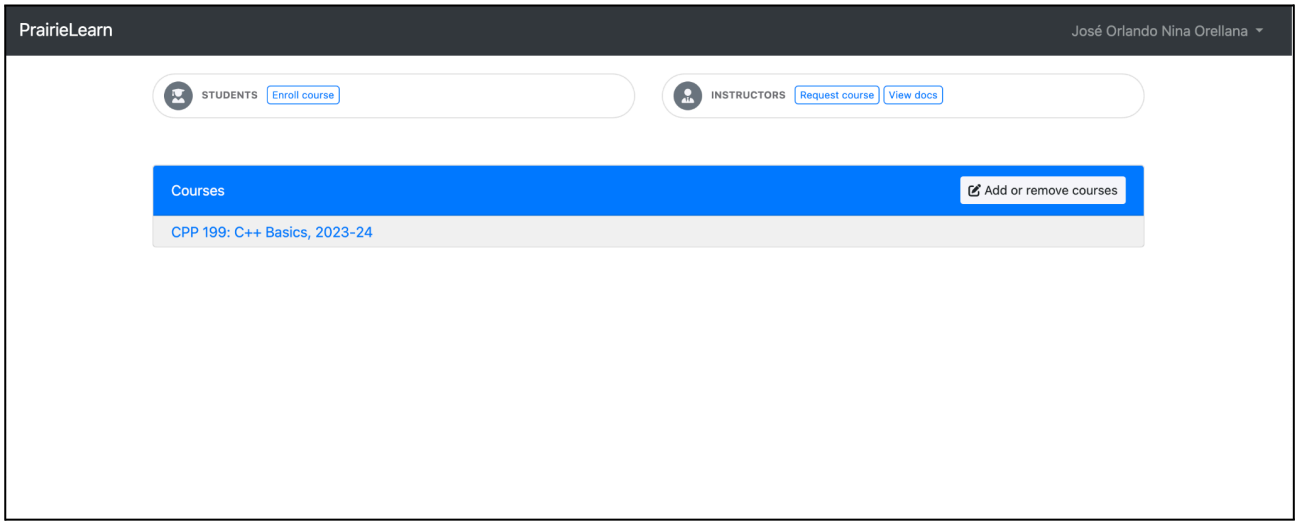

Figura 2.4: Pantalla de inicio de alumnos

La única diferencia es que al profesor le aparece una lista de cursos en los cuales tiene permisos de instructor para realizar modificaciones.

Al seleccionar el curso, ambos usuarios tienen pantallas diferentes, como se pueden ver en las figuras 2.5 y 2.6. El profesor podrá ver todas las posibles configuraciones que puede llegar a tener el curso. En cambio, el alumno verá una serie de preguntas asociadas al curso disponibles para contestar.

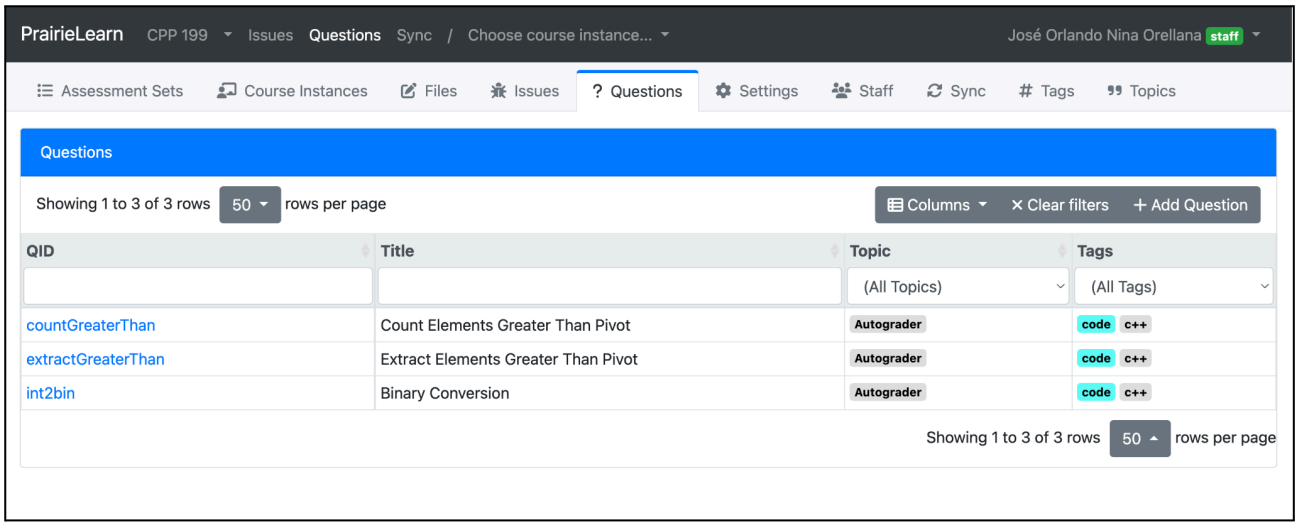

Figura 2.5: Pantalla de curso de profesores

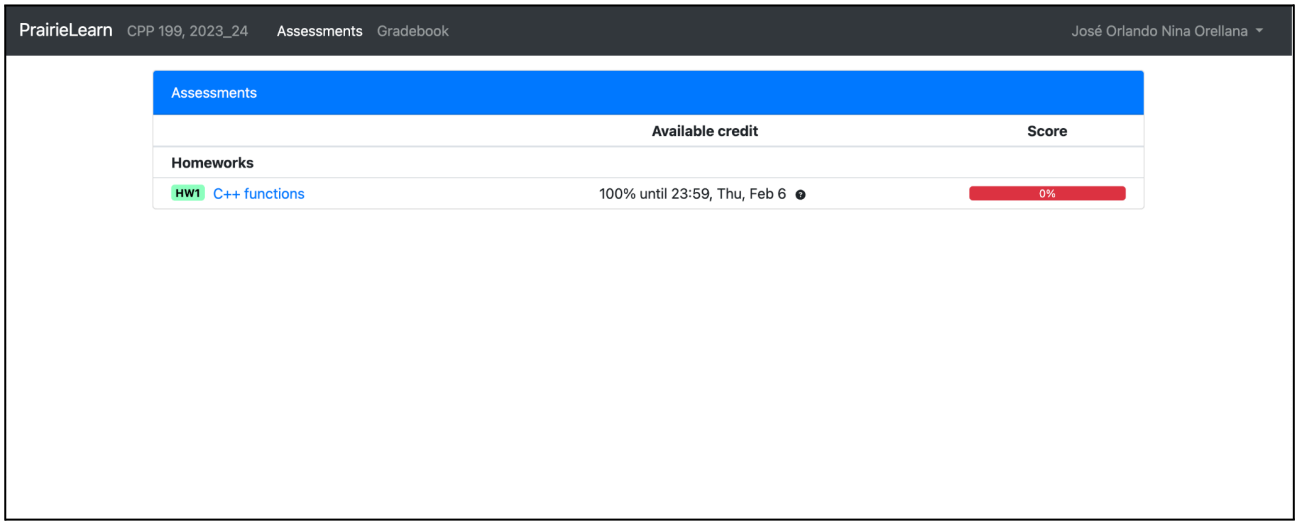

Figura 2.6: Pantalla de curso de alumnos

Capítulo 3

Tecnologías

Durante el desarrollo de este proyecto se utilizarán diversas tecnologías por su compatibilidad y con las herramientas seleccionadas anteriormente. A continuación un listado con las principales tecnologías.

3.1 Docker

Docker tiene la capacidad de crear contenedores aislados que permite añadir seguridad cuando se ejecute código de terceros que podría ser malicioso, protegiendo el sistema de posibles vulnerabilidades.

3.1.1 ¿Qué es Docker?

Docker [5] es una plataforma de código abierto que permite empaquetar y desplegar una aplicación en un entorno aislado llamado contenedores, donde se instalan todas las dependencias necesarias para que se ejecute el código correctamente. De esta manera, sin importar el entorno de desarrollo, cualquier persona obtendrá el mismo comportamiento del software.

3.1.2 ¿Cómo crear contenedores?

La creación de contenedores puede realizarse de varias formas:

- **A partir de un Dockerfile:** se puede lograr creando un Dockerfile, un archivo de texto que contiene todas las instrucciones necesarias para construir una imagen de Docker. Una imagen es como una plantilla de solo lectura que contiene el código

fuente de la aplicación y todas las dependencias para que se ejecute. A partir de esta se pueden instanciar contenedores.

- **Usando imágenes ya existentes:** es posible utilizar imágenes preexistentes que se pueden descargar de sitios web como Docker Hub [6], un repositorio público de imágenes.
- **Personalizar imágenes mediante un Dockerfile:** en lugar de crear un imágen desde cero, también existe la posibilidad de tomar como base una imagen y añadirle nuevas configuraciones por encima.

3.1.3 Docker compose

Docker Compose es una extensión de Docker para definir y ejecutar aplicaciones multi-contenedor. La configuración se describe en un archivo YAML, donde se puede especificar: versión, servicios, redes y volúmenes. Luego, con un solo comando se crean y ponen en funcionamiento todos los contenedores definidos en el archivo de configuración.

3.2 Node.js

Node.js es un entorno de ejecución para ejecutar código en JavaScript fuera del navegador. Es de código abierto y multiplataforma, lo que permite su uso en diferentes sistemas operativos.

Durante el proyecto se usó para crear la API generadora de tests. Una API se define como un intermediario entre dos aplicaciones permitiendo comunicarse entre sí para intercambiar datos, características y funcionalidades.

3.3 PostgreSQL

PostgreSQL [7], o también conocido como Postgres, es un sistema de base de datos de código abierto. A diferencia de otros RDMBS (sistemas de gestión de bases de datos relacionales), admite tanto tipos de datos no relacionales como relacionales.

3.4 Nginx

Nginx es un software de código abierto para servidores web, proxy inverso, almacenamiento en caché, equilibrio de carga, streaming multimedia, entre otras funciones. Principalmente se utilizará como proxy inverso que actúa como intermediario entre el cliente y los servidores web, reenviando las peticiones a los servidores y devolviendo la respuesta a los clientes.

3.5 Google Test Framework

Google Test Framework, también conocido como GTest, es un framework de pruebas unitarias para C++ desarrollado por Google. Proporciona un amplio conjunto de herramientas y funciones para escribir y ejecutar pruebas unitarias de código C++.

3.5.1 Ejemplo de test

Como se ve en la figura 3.1, un ejemplo de tests para verificar el funcionamiento de la función `add`.

```
TEST(MyTestSuite, MyTestCase) {
    ASSERT_EQ(2, add(1, 1));
    ASSERT_EQ(5, add(3, 2));
    ASSERT_EQ(10, add(7, 3));
}
```

Figura 3.1: Ejemplo de test unitario

En este ejemplo, se puede ver como se define un test utilizando el macro `TEST` proporcionado por Google Test. Se define el nombre del conjunto de pruebas `MyTestSuite` y el nombre de la prueba `MyTestCase`. Dentro del cuerpo de la prueba, se definen una serie de afirmaciones que la función `add` debería pasar.

Capítulo 4

Solución global

La solución propuesta, como se ha adelantado anteriormente, estará compuesta por distintos módulos conectados entre sí. Para clarificar su funcionamiento desde el punto de vista de los distintos usuarios que interactúan con la herramienta, se han elaborado diagramas de caso de uso que se presentan en la figura 4.1.

4.1 Diagrama casos de uso

Los diagramas de casos de uso permiten visualizar las interacciones entre los usuarios y el sistema, en este caso, la plataforma de evaluación.

Figura 4.1: Diagrama casos de uso

En la figura 4.1 se muestra a los actores junto a una serie de casos de uso. Hay dos principales actores:

- Alumno: usuario que representa a los estudiantes que utilizan la plataforma para inscribirse a cursos, resolver preguntas y ver sus calificaciones.
- Profesor: usuario encargado de crear cursos, asignar preguntas y calificar respuestas.

En cuanto a los casos de uso, como se ve en el diagrama, se pueden relacionar utilizando las anotaciones <<extend>> o <<include>>. La primera [8] se refiere a un caso de uso opcional después del caso de uso base. Se representa con una flecha apuntando al caso base con su correspondiente anotación. La segunda anotación se refiere a un caso de uso obligatorio, es decir, para realizar un caso de uso es necesario ejecutar el otro. Se representa con una flecha con el caso de uso a incluir. A continuación, se listan los casos de uso que aparecen en la figura 4.1:

- En cuanto a los alumnos:
    - Inscribirse en un curso: el alumno puede inscribirse en un curso.
    - Responder preguntas: el alumno puede responder a las preguntas de un curso en el que está inscrito.
    - Ver calificación: el alumno puede visualizar la calificaciones de las preguntas. Este caso depende del caso “Calificar una pregunta”, ya que el alumno solo puede ver la calificación una vez ha sido calificada.
- En cuanto al profesorado:
    - Crear un curso: el profesor puede diseñar un nuevo curso.
    - Crear instancia de curso: el profesor puede crear una instancia de un curso. Este caso de uso depende del caso “Crear un curso”, ya que las instancias se hacen a partir de un curso ya existente. Además, puede incluir el caso de uso de “Asignar preguntas”, puesto que también se pueden añadir preguntas a la instancia.
    - Crear preguntas: el profesor puede crear preguntas. Es opcional incluir “Crear pruebas” al crear una pregunta, de forma que se creen una conjunto de pruebas para calificar la pregunta. A su vez se subdivide en “Crear pruebas manualmente” o “Crear pruebas automáticamente”.
    - Calificar una pregunta: el profesor puede asignar una calificación a una

respuesta de una pregunta. Este caso de uso se subdivide en “Calificar manualmente ” o “Calificar automáticamente”.

4.2 Arquitectura de la solución

En la figura 4.2 se muestra la estructura de la solución global compuesto por dos contenedores principales con líneas continuas en la parte superior: PrairieLearn, que administra los estudiantes y gestiona las tareas de programación, y la API de generación de tests, encargada de generar los tests para la evaluación. También se incluyen dos contenedores que son generados y destruidos dinámicamente representados con líneas discontinuas y ubicados en la parte inferior: el contenedor de la izquierda se utiliza para realizar la calificación de una pregunta, y el contenedor de la derecha para generar automáticamente tests unitarios basados en el código de solución.

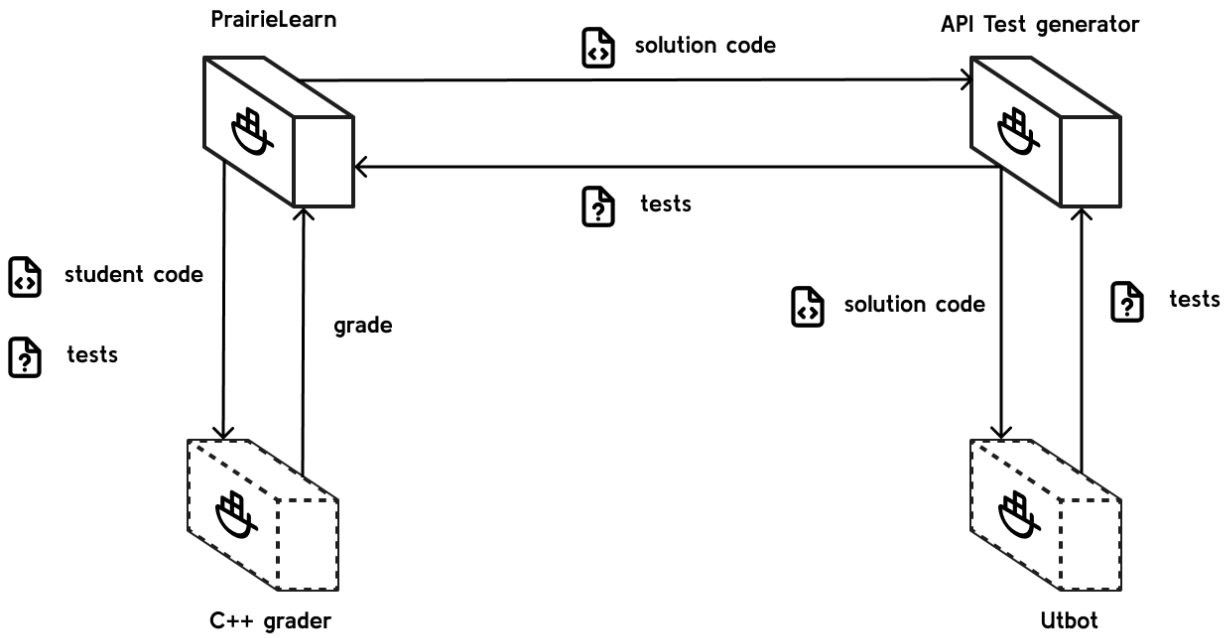

Figura 4.2: Esquema de la arquitectura de la solución

El sistema consta de 2 principales funcionalidades:

1. Creación de tests:

a. El contenedor de PrairieLearn envía una petición para crear tests, adjuntando la solución de una pregunta.
b. Cuando el servidor recibe esta petición, crea un contenedor temporal instanciado a partir de una imagen, que contiene la herramienta UTBotCpp y mediante un volumen se pasa la solución.
c. Tras la creación del contenedor se lanza un script y se generan los tests que luego se devuelven al PrairieLearn como respuesta.

2. Calificación de una pregunta:
    a. Cuando el alumno solicita una calificación, se crea un contenedor temporal, que tiene instalado el compilador de C++ y la librería de Google Test para que pueda lanzar los tests. Gracias a un volumen se pasa al contenedor el código del alumno y los tests.
    b. Mediante un script, se ejecutan los tests y se genera una calificación basada en los tests superados, que luego se muestra al alumno.

Capítulo 5

Desarrollo

5.1 Generador de pruebas

5.1.1 Imagen de Docker del generador de pruebas

La instalación de UTBotCpp está oficialmente diseñada como una extensión de Visual Studio Code o CLion, y no para ser utilizada como una librería. Su instalación habitualmente consiste en la descarga de un archivo zip que es reconocido como extensión de estas herramientas. Sin embargo, en este caso, se debe adaptar el código empaquetado originalmente para utilizar UTBotCpp bien como una librería o como un programa autónomo capaz de generar tests a demanda.

```
FROM ghcr.io/unittestbot/utbotcpp/base_env:2024.03.1
RUN git clone https://github.com/UnitTestBot/UTBotCpp/
RUN /bin/bash -c "source docker/building_dependencies/runtime_env.sh"
RUN git submodule update --init --recursive
RUN ./build.sh
```

Figura 5.1: Dockerfile del generador de pruebas

Se ha creado, en cambio, una imagen de Docker que contenía la herramienta. Para generar la imagen, se creó un archivo Dockerfile, como se puede ver en la figura 5.1, que utilizaba como base una imagen que contenía todas las dependencias necesarias para desplegar UTBotCpp. Luego, se clonó el repositorio de GitHub de la herramienta y ejecutando un script del repositorio, se construyó la herramienta.

5.1.2 Script para generar pruebas

Se ha generado un script para generar pruebas, puesto que la herramienta carecía de esta posibilidad al estar diseñada para ser utilizada a través de una interfaz gráfica.

```
#!/bin/bash
source $UTBOT_RUNTIME_ENV
mkdir -p $PROJECT_PATH/build
cd $PROJECT_PATH/build
$UTBOT_INSTALL_DIR/bin/cmake ..
$UTBOT_ALL/server-install/utbot generate --project-path $PROJECT_PATH file
--file-path $FILE_PATH
```

Figura 5.2: Script para la generación de tests

En la figura 5.2, se detalla el contenido del script de bash que tiene los siguientes pasos:

1. Se ejecuta un script que establece una serie de variables de entorno requeridas para UTBotCpp.
2. Se crea un directorio para almacenar los archivos generados y se cambia a ese directorio.
3. Se ejecuta el CMake para configurar el proyecto.
4. Se llama a la herramienta UTBotCpp con la opción de generar, pasando como argumento la ruta del archivo a partir del cual se quieren generar los tests.

5.1.3 API

Para conectar Prairielearn con la herramienta de generación de pruebas, se desarrolló una API. Esta API es capaz de recibir código en C++, generar automáticamente tests unitarios a partir de dicho código, y devolverlos al cliente.

La API tiene un único punto de acceso (entrypoint) en la ruta `/generate`, que es controlado por la función `generate` que se encarga de la creación de los pruebas unitarias.

```
export const generate = async (req, res) => {
  try {
    await cleanProject()
    const code = req.body.code
    await fs.writeFile(FUNCTION_PATH, code)
    await generateTests()
    const test = await fs.readFile(FUNCTION_TEST_PATH)
    res.status(200).send(test)
  } catch (error) {
    console.error(error)
    res.status(500).send(error)
  }
}
```

Figura 5.3: Función encargada de procesar la petición

```
const generateTests = async () => {
  const docker = new Docker()
  const container = await docker.createContainer({
    Image: IMAGE_NAME,
    HostConfig: {
      Binds: [`${PROJECT_PATH}:${PROJECT_PATH}`]
    },
    Entrypoint: ['/bin/bash', '-c'],
    Cmd: [SCRIPT_PATH]
  })
  await container.start()
  await container.wait()
  await container.remove()
}
```

Figura 5.4: Función encargada de generar los tests

En la figura 5.3 se puede ver la función que se encarga de procesar la petición al entrypoint. Los pasos que se realizan son los siguientes:

1. Limpieza del proyecto: la función `cleanProject` elimina archivos de solicitudes anteriores.
2. Cargar código en un archivo: del cuerpo de la petición se extrae el código y se guarda en un archivo dentro de un proyecto de C++.
3. Generación de tests: en la figura 5.4 se muestra el código de la función

`generateTests`. Los instrucciones que se ejecutan son:

   a. Se crea un contenedor temporal utilizando Dockerode, un módulo de Node.js usado para interactuar con Docker.
   b. Este contenedor se instancia con la imagen de UTBotCpp.
   c. Se le pasa un bind que contiene un proyecto en C++ con el archivo cargado anteriormente.
   d. Se define un entrypoint que lanza el script para generar los tests.
   e. Durante su ejecución se muestra por pantalla la salida del contenedor.
   f. Finalmente, se espera a que acabe la generación y se elimina.

4. Envío de respuesta: por último se lee el contenido del archivo de test y se envía como respuesta en el body.

5.2 Prairielearn

Prairielearn se organiza en cursos, y cada curso está representado por una carpeta con la siguiente estructura:

- `infoCourse.json`: este archivo especifica información básica sobre el curso.
- `questions`: directorio con todas las preguntas del curso.
- `courseInstances`: directorio que contiene diferentes instancias del curso. Una instancia de curso se refiere a una oferta curso en un período determinado. Por ejemplo “Otoño 2019”.

En cuanto a un ejemplo de pregunta de programación, tendría la siguiente estructura:

- `info.json`: archivo que define las propiedades de la pregunta.
- `question.html`: archivo que define como se verá el enunciado..
- `code.cpp`: archivo inicial que el alumno utilizará para resolver la pregunta.
- `tests`: carpeta que contiene las pruebas utilizadas para generar la calificación.

Prairielearn ofrece la opción de activar la calificación automática. En el archivo `info.json`, como se ve en la figura 5.5, se debe especificar la imagen de Docker que se

va a usar y la ruta del archivo que lanzará el proceso de calificación. El archivo `info.json` tendría un contenido como:

```
{
    "uuid": "149f9b40-2b1d-40f2-88a2-932802c5269c",
    "title": "First question",
    ...
    "externalGradingOptions": {
        "enabled": true,
        "image": "my-grader",
        "entrypoint": "grade/init.sh"
    }
}
```

Figura 5.5: Archivo de configuración de una pregunta

Durante el proceso de calificación, se pasará al contenedor un volumen con el siguiente contenido:

- `results`: informe del resultado de la calificación que se le mostrará al alumno.
- `student`: archivos enviados por los alumnos.
- `tests`: pruebas que se ejecutan al código del alumno.

En cuanto al informe de calificación, se debe escribir los resultados en un archivo `results/results.json` para que se puede mostrar al usuario, siguiendo la estructura:

1. `gradable`: booleano que indica si el entregable es calificable.
2. `message`: mensaje sobre los resultados.
3. `score`: un flotante que refleja la puntuación de la tarea, entre 0.0 y 1.0.
4. `tests`: resultados de cada prueba. Cada una tiene:
    a. `name`: nombre de la prueba.
    b. `points`: puntos obtenidos.
    c. `max_points`: máximos puntos posibles.
    d. `output`: salida del test.

5.2.1 Creación de calificador para C++

Para generar una calificación, se evaluará el número de pruebas que pase el código del alumno sobre el total de pruebas generadas. Por lo tanto, será necesario crear una imagen de Docker que sea capaz de coger los tests, ejecutarlos sobre el código del alumno y generar un archivo que muestre al usuario el resultado de las pruebas.

```
FROM ubuntu:22.04
RUN apt-get update && apt-get install -y g++ cmake git python3
RUN git clone https://github.com/google/googletest.git && \
...
```

Figura 5.6: Dockerfile del calificador de C++

Dado que UTBotCpp genera los tests en formato Google Test en la imagen de docker se instaló esta librería, como se ve en la figura 5.6. También, se instaló Python, para crear un scripts que genere el archivo del resultado.

```
#!/bin/bash
output=$(g++ -o ${EXECUTABLE} ${TEST_DIR}/*.cpp -lgtest -lgtest_main -pthread
2>&1)
if [ $? -ne 0 ]; then
    mkdir -p $JSON_FOLDER && touch $JSON_FILE
    python3 /config/errorToJson.py "$output" "$JSON_FILE"
else
    ${EXECUTABLE} --gtest_output="xml:$XML_FILE"
    mkdir -p $JSON_FOLDER && touch $JSON_FILE
    python3 /config/gtestToJson.py "$XML_FILE" "$JSON_FILE"
fi
```

Figura 5.7: Script para generar la calificación

En la figura 5.7, se puede observar el script para el proceso de calificación, se realiza de la siguiente manera:

1. Se compila el código y las pruebas. Si la compilación falla, se captura el mensaje de error.
2. Comprobación del resultado:

a. Si la compilación no es exitosa, se crea un archivo JSON donde se guardaría el mensaje de error utilizando un script en Python.
b. Si la compilación es exitosa, se ejecuta el programa compilado y se almacena el resultado en un archivo XML. Utilizando otro script en Python, se lee el contenido del archivo XML para generar un JSON con la calificación en función a los tests superados.

5.2.2 Fork del código

Para poder realizar cambios y guardarlos en un repositorio remoto como GitHub, se hizo un fork (bifurcación) del código original de Prairielearn. Hacer esto es necesario porque no tenemos permisos para editar el repositorio original. Al hacer un fork, se crea una copia del repositorio en nuestra cuenta, permitiendo realizar modificaciones sin afectar al proyecto original.

5.2.3 Crear pregunta por defecto de programación

Cuando se añade una pregunta a un curso, lo que se hace es copiar una pregunta ya existente de un curso de ejemplo. Anteriormente, al añadir una nueva pregunta, se creaba una que no tenía relación con la programación.

Para añadir una pregunta de programación por defecto, se creó una nueva pregunta siguiendo la estructura definida anteriormente al principio del apartado 5.2, añadiendo nuestro calificador creado en el punto 5.2.1. Posteriormente, se cambió la ruta para que utilizase la nueva pregunta.

5.2.4 Conexión con la API

En el panel de administración del curso existe un botón para añadir una nueva pregunta. Su función original es clonar una pregunta por defecto y redirigir a la sección de configuración de la pregunta.

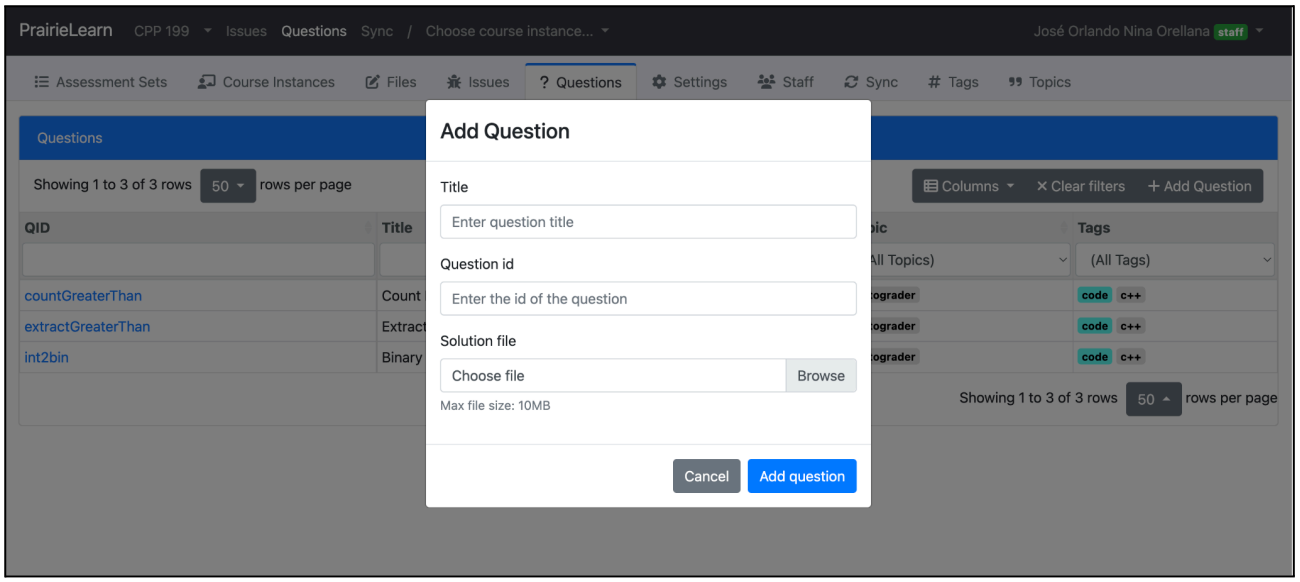

Figura 5.8: Modal para añadir una pregunta

Para conectar con la generación de pruebas, se creó un modal, como se ve en la figura 5.8, que solicita el título de la pregunta, el nombre de la carpeta y el archivo de código de la solución. Al pulsar en añadir, se genera la pregunta copiando la pregunta de programación por defecto. Además, en segundo plano, se realiza una petición a la API generadora de tests con el contenido del archivo de solución. Cuando se termina de crear la pregunta se muestra un toast (un mensaje temporal que se muestra por pantalla) indicando si la pregunta se ha creado correctamente.

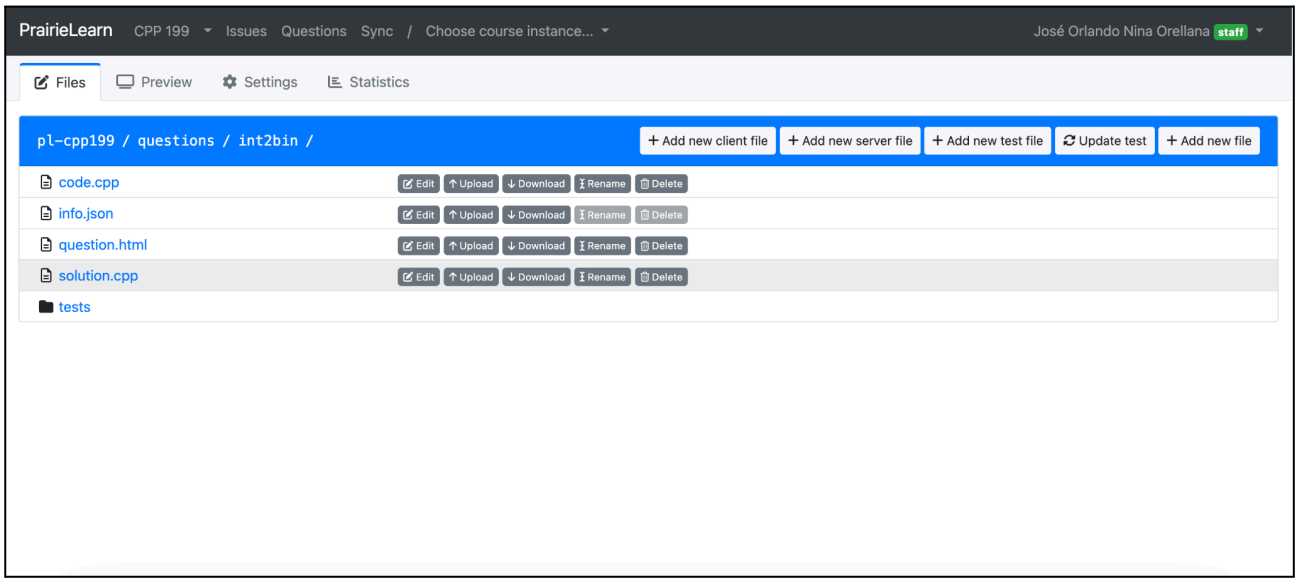

Figura 5.9: Estructura de archivos de una pregunta

Al crear la pregunta, se tiene la siguiente estructura de archivos, como se ve en la figura 5.9. Los archivos de tests tienen la posibilidad de ser editados desde esta interfaz. Además, se añadió un nuevo botón de “Update tests”, de forma que, al editar el archivo de solución, se borran los tests actuales y se generan unos nuevos.

5.3 Modelo de IA para la generación de tests

5.3.1 Creación del modelo

Se propuso crear una alternativa para la creación de tests con Inteligencia Artificial, usando Ollama, una herramienta que permite ejecutar modelos de lenguaje a gran escala (LLMs) de forma local.

```
FROM codellama:7b
PARAMETER temperature 0
SYSTEM """
You are an AI assistant specialized in generating unit tests for C++ code
using the Google Test framework. Your task is to generate test cases given a
function.
"""
```

Figura 5.10: Modelfile para crear un modelo de IA personalizado

Ollama también permite crear modelos personalizados mediante un archivo de configuración denominado Modelfile. Como se muestra en la figura 5.10, se define lo siguiente:

1. Se especifica que se usará como modelo base codellama, un modelo especializado en código.
2. Un parámetro denominado temperatura que define en una escala de 0 a 1 que tan creativo puede llegar a ser el modelo.
3. Por último, un prompt que define cuál es la funcionalidad del modelo.

Utilizando la imagen de Docker `ollama/ollama` se puede crear el modelo. Como

se ve en la figura 5.11, se define un contenedor en el cual se usa la imagen mencionada anteriormente, un volumen para guardar los modelos creados y un puerto al cual se podrán enviar peticiones hacia los modelos.

```
version: '3.8'
services:
 ollama:
    image: ollama/ollama:latest
    ports:
      - 11434:11434
    volumes:
      - ./ollama/models:/ollama/models
```

Figura 5.11: Archivo docker-compose para crear el modelo de IA personalizado

Si levantamos el contenedor y nos metemos adentro se puede crear el modelo utilizando el comando: `ollama create test-generator -f Modelfile`. En este comando le pasamos el nombre que tendrá el modelo y también la ruta del archivo de configuración.

5.3.2 Ejemplo de uso

Para poder utilizar Ollama, se puede hacer a través de un chat por línea de comando o a través de un API que trae por defecto. Utilizaremos la segunda opción, empleando el puerto 11434 expuesto en la figura 5.11.

```
{
    "model": "test-generator",
    "prompt": "int add(int a, int b) {\n  return a + b;\n}"
}
```

Figura 5.12: Petición de ejemplo a la IA generadora de tests

Como se puede observar en la figura 5.12, se envía un json especificando el modelo y el prompt, que en nuestro caso es una función a partir de la cual se generarán los tests. Como resultado, se retornó los tests que se pueden ver en la figura 5.13:

```
TEST(addition, test_1) {
    EXPECT_EQ(add(2, 3), 5);
}
TEST(addition, test_2) {
    EXPECT_EQ(add(-2, 3), 1);
}
TEST(addition, test_3) {
    EXPECT_EQ(add(2, -3), -1);
}
TEST(addition, test_4) {
    EXPECT_EQ(add(-2, -3), -5);
}
```

Figura 5.13: Tests resultantes del ejemplo a la IA generadora de tests

5.3.3 Conexión con Prairielearn

En cuanto a la conexión con Prairielearn, en el modal que se hizo en la figura 5.8, se añadió un checkbox para dar la opción de generar los tests usando el modelo de generación de tests. De esta manera, al marcar la opción de IA enviará una petición como se mostró en el apartado 5.3.2.

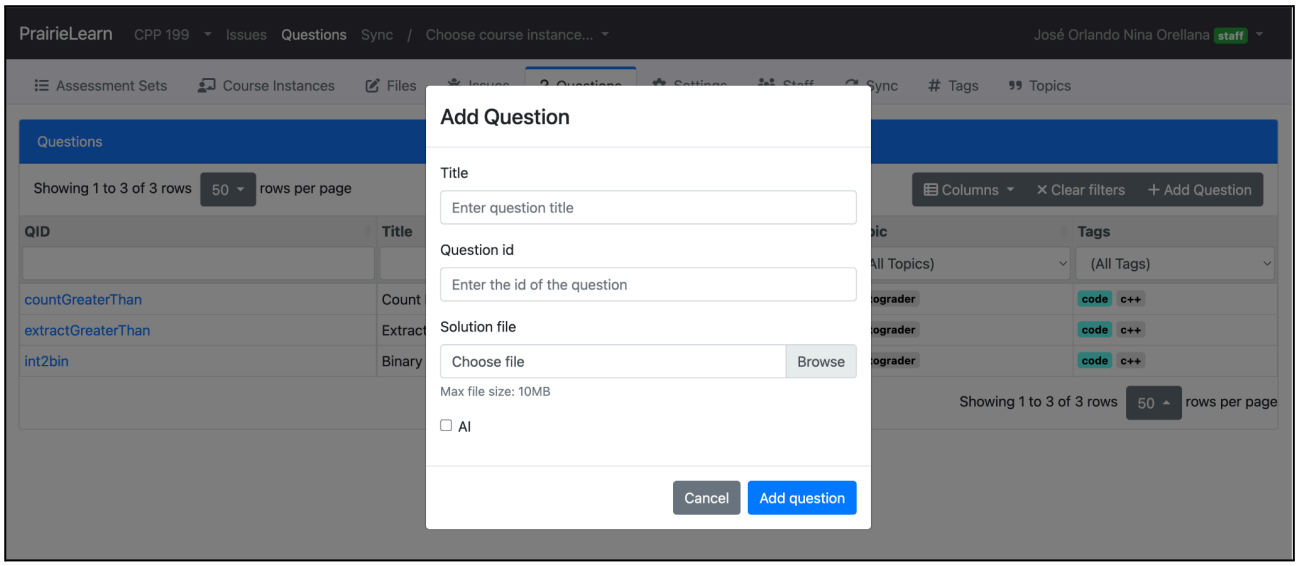

Figura 5.14: Modal para añadir una pregunta con la opción de IA

Capítulo 6

Despliegue

En cuanto al despliegue, nos referimos al proceso de instalación de nuestra aplicación web en un servidor. De esta forma, la aplicación será pública para que tanto el profesorado como el alumnado pueda acceder a ella a través de internet. A continuación, los pasos realizados para hacer el despliegue.

6.1 Autenticación con Google

Como método de autenticación se usó Google OAuth 2 [9], que proporciona a la aplicaciones acceso limitado a recursos protegidos del usuario sin necesidad de compartir sus credenciales.

Para que los usuarios puedan iniciar sesión usando su cuenta de Google, hay que seguir los siguientes pasos:

1. Crear un nuevo proyecto en Google Cloud Console.
2. Obtener credenciales.
    a. Seleccionar Aplicación web como tipo de aplicación.
    b. Añadir la URI de la aplicación y la URI de redireccionamiento después de la autenticación, como se ve en la figura 6.1. Es importante que las URIs no sean direcciones IP y que terminen en un dominio válido para cumplir los requisitos de seguridad de Google OAuth 2.

Figura 6.1: URIs de Google OAuth 2

Al completar estos pasos se generarán un ID de cliente y un secreto que se usará en la configuración al levantar la aplicación.

6.2 Organización de GitHub

Prairielearn te da la opción de poder alojar los cursos de forma remota, para ello se debe crear una organización en Github. Dentro de esta organización se creó un repositorio donde se alojó la plantilla para instanciar nuevos cursos solicitados. Para crear la organización se se siguieron los siguientes pasos:

1. Se accedió a nuestra página principal de GitHub y se pulsó en el botón de “más” al lado de nuestra foto de perfil. Se seleccionó en “nueva organización”.
2. Se escogió un plan según los requisitos de la organización, en este caso el plan gratuito.
3. Por último, se rellenaron campos necesarios especificando el nombre de la organización, un correo de contacto y el tipo de organización.

De esta forma, cada vez que se reciba y acepte una petición de creación de curso, se creará un repositorio en la organización utilizando la plantilla del curso.

6.3 Archivo de configuración y despliegue

La aplicación se puede configurar mediante un archivo config.json. En este archivo, se incluyó los siguientes elementos:

Tabla 6.1: Elementos del archivo de configuración

Elemento	Descripción
googleClientId	ID de cliente de Google OAuth
googleClientSecret	Secreto de cliente de Google OAuth
googleRedirectUrl	URI de redireccionamiento después de la autenticación
githubClientToken	Token de acceso a GitHub
githubCourseOwner	Nombre de la organización de GitHub
githubCourseTemplate	Repositorio de la plantilla de cursos
cookieDomain	Dominio de las cookies

El despliegue se realizó utilizando Docker Compose que permite ejecutar aplicaciones multicontenedor. En la figura 6.2, se puede ver como se definieron tres contenedores y un volumen `postgres` para la base de datos. En este archivo se especificaron las imágenes que van a utilizar, los puertos en los que se van a alojar y los archivos que se pasan mediante volúmenes.

```
version: '3.8'
services:
  pl:
    image: orlqndo/pl:v1.0.0
    ports:
      - 3000:3000
    volumes:
      - postgres:/var/postgres
      - /var/run/docker.sock:/var/run/docker.sock
      - ./config.json:/PrairieLearn/config.json

  tg:
    image: orlqndo/test-generator
    ports:
      - 3030:3030
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - /cpp-project/:/cpp-project/

  ollama:
    image: ollama/ollama:latest
    ports:
      - 11434:11434
    volumes:
      - ./ollama/models:/ollama/models

volumes:
  postgres:
```

Figura 6.2: Archivo docker-compose para desplegar el proyecto

Ejecutando el comando `docker compose up` se levantó el proyecto teniendo como punto de acceso el puerto 3000 de la máquina.

6.4 Reverse proxy

La máquina que se quería establecer como punto de acceso es codetest.iaas.ull.es que tiene IP pública y permite el acceso externo. Sin embargo, nuestra aplicación está desplegada en un máquina virtual interna con IP 10.6.130.178. Por lo tanto, se desea que el tráfico que llega a codetest.iaas.ull.es llegue a la aplicación. Para solucionarlo, en la máquina a donde apunta el dominio se instala Nginx. Se edita el fichero de configuración

indicando que el tráfico se redirige a la IP de la máquina que aloja la aplicación en el puerto 3000, como se ve en la figura 6.3.

```
server {
    location / {
      proxy_pass http://10.6.130.178:3000;
      ...
    }
}
```

Figura 6.3: Archivo de configuración de Nginx

De esta manera, los usuarios pueden acceder a la aplicación usando simplemente la URL codetest.iaas.ull.es sin necesidad de especificar el puerto. A continuación, se muestra la figura 6.4 con la estructura física final:

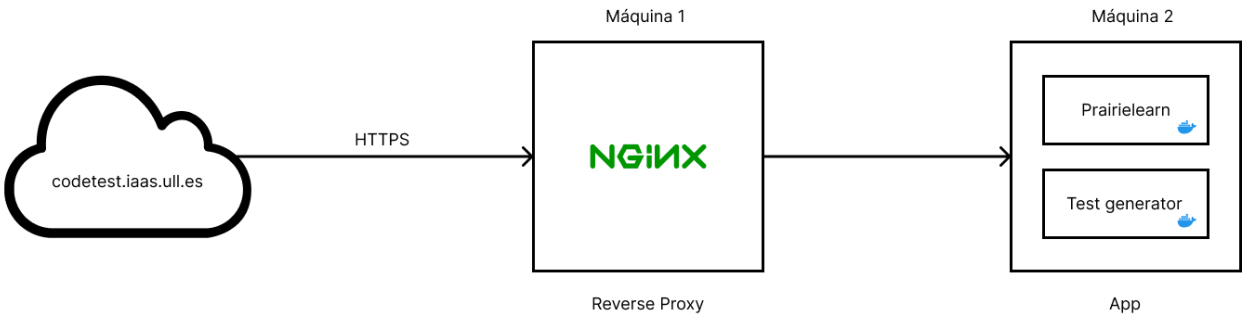

Figura 6.4: Estructura del proyecto

Capítulo 7

Validaciones y resultados

Para poner a prueba la herramientas la herramientas de generación de tests se pasaron a ambos, tanto UTBotCpp como el modelo de IA, el mismo código para ver el desempeño. Se seleccionaron los ejemplos que se pueden observar en las figuras 7.1 y 7.2.

```
std::string int2bin(unsigned int number) {
  std::string res;
  if (!number) return "0";
  while (number) {
    res = ((number & 1) ? "1" : "0") + res;
    number >>= 1;
  }
  return res;
}
```

Figura 7.1: Función que pasa un número a binario

```
int whileLoop(int k) {
  int i = 0;
  int sum = 0;
  while (i < k) {
      sum += i;
      i += 1;
    }
    return sum;
}
```

Figura 7.2: Función que utiliza un bucle

En cuanto a los resultados obtenidos, el modelo de IA pudo generar correctamente los tests para ambos ejemplos, cómo se pueden ver en las figuras 7.3 y 7.4. En cambio, UTBotCpp sólo pudo generar tests para el segundo ejemplo, como se puede observar en la figura 7.5, debido a que no soporta librerías como la de string y solo permite trabajar con tipos de datos primitivos. Además, recalcar que en el segundo caso UTBotCpp empleó menos tiempo, ya que el modelo de IA requiere de más recursos.

```
TEST(CodeTest, TestCase1) {
  unsigned int input = 3;
  std::string expected = "11";
  EXPECT_EQ(int2bin(input), expected);
}
TEST(CodeTest, TestCase2) {
  unsigned int input = 7;
  std::string expected = "111";
  EXPECT_EQ(int2bin(input), expected);
}
TEST(CodeTest, TestCase3) {
  unsigned int input = 15;
  std::string expected = "1111";
  EXPECT_EQ(int2bin(input), expected);
}
```

Figura 7.3: Tests generados por IA a la función que pasa un número a binario

```
TEST(whileLoop, TestCase1) {
  int result = whileLoop(5);
  EXPECT_EQ(result, 10);
}
TEST(whileLoop, TestCase2) {
  int result = whileLoop(10);
  EXPECT_EQ(result, 45);
}
TEST(whileLoop, TestCase3) {
  int result = whileLoop(100);
  EXPECT_EQ(result, 4950);
}
```

Figura 7.4: Tests generados por IA a la función que utiliza un bucle

```
TEST(regression, whileLoop_test1)
{
    int actual = whileLoop(1);
    EXPECT_EQ(0, actual);
}
TEST(regression, whileLoop_test2)
{
    int actual = whileLoop(0);
    EXPECT_EQ(0, actual);
}
```

Figura 7.5: Tests generados por UTBotCpp a la función que utiliza un bucle

Capítulo 8

Conclusiones y líneas futuras

El presente trabajo fin de grado ha permitido la creación de una plataforma online capaz de proporcionar a los estudiantes un medio donde subir su código y recibir retroalimentación, y disminuir el tiempo a los profesores empleado en la evaluación manual. En cuanto al cumplimiento de los objetivos, se concluye lo siguiente:

- Se logró desplegar la herramienta Prairielearn que ofrece una interfaz para que los estudiantes puedan subir su código.
- Se desarrolló una imagen de Docker compatible con Prairielearn, que automatiza el lanzamiento de tests y la generación de una puntuación.
- Se implementó un sistema principal para la generación de tests utilizando UTBotCpp, que tiene limitaciones, y también un modelo experimental de IA, al cual le faltan realizar pruebas más profundas. Por lo tanto, este punto necesita mejoras para poder completar este objetivo.

La realización de este trabajo ha ayudado a entender el funcionamiento interno de herramientas complejas. Entre las dificultades encontradas, destaca adaptarse a tecnologías sin ninguna experiencia previa como Docker y, sobre todo trabajar con código ajeno que implica analizar y entender cómo funciona para luego añadir nuevas funcionalidades.

En cuanto a posibles líneas futuras de desarrollo, se barajan distintas opciones para añadir funcionalidades o mejoras a la herramienta. Sería interesante cambiar la interfaz gráfica, que actualmente parece externa a la universidad. Se deberían cambiar estilos y textos que se ajusten más a la institución. También, eliminar elementos de Prairielearn que no tienen relación con la docencia y se utilizan como publicidad.

También se debería testear la herramienta en un caso real para evaluar su impacto en la docencia. También por cuestiones de rendimiento, probar cómo se desenvuelve, al

ser utilizada por un grupo amplio de usuarios en tiempo real. Esto podría conllevar a un rediseño del sistema para que pueda ser escalable. Además ayudaría a identificar problemas que no se hayan detectado durante el desarrollo.

Por último, se propone integrar algún tipo de evaluador con Inteligencia Artificial que, además de corregir código, pueda proporcionar sugerencias, ejercicios de refuerzo y que sea de alguna forma personalizado a cada alumno, ya que todos no avanzan de la misma manera.

Capítulo 9

Summary and Conclusions

This final degree project has allowed the creation of an online platform capable of providing students with a means of uploading their code and receiving feedback, and reducing the time teachers spend on manual evaluation. Regarding the fulfillment of the objectives, we conclude the following:

- It was possible to deploy the Prairielearn tool that provides an interface for students to upload their code.
- A Docker image compatible with Prairielearn was developed, which automates the launching of tests and the generation of a score.
- A main system for test generation using UTBotCpp was implemented, which has limitations, and also an experimental AI model, which lacks further testing. Therefore, this point needs improvement in order to complete this objective.

The completion of this work has helped to understand the inner workings of complex tools. Among the difficulties encountered, it stands out adapting to technologies without any previous experience such as Docker and, above all, working with foreign code that involves analyzing and understanding how it works and then adding new functionalities.

As for possible future lines of development, different options are being considered to add functionalities or improvements to the tool. It would be interesting to change the graphical interface, which currently seems external to the university. Styles and texts should be changed to be more in line with the institution. Also, remove elements of Prairielearn that are not related to teaching and are used as advertising.

The tool should also be tested in a real case to evaluate its impact on teaching. Also for performance issues, test how it performs when used by a large group of users in

real time. This could lead to a redesign of the system to make it scalable. It would also help to identify problems that were not detected during development.

Finally, it is proposed to integrate some kind of evaluator with Artificial Intelligence that, in addition to correcting code, can provide suggestions, reinforcement exercises and that is somehow personalized to each student, since not everyone advances in the same way.

Capítulo 10

Presupuesto

A continuación, se puede ver una tabla con las distintas fases del proyecto, con las horas dedicadas a cada una de ellas y su coste por hora, resultando en un total de 8.300 euros.

Tabla 9.1: Presupuesto del proyecto

Tipo	Horas	Coste	Total
Investigación	40 horas	20 €/hora	800,00 €
Pruebas de herramientas	60 horas	25 €/hora	1.500,00 €
Desarrollo	160 horas	30 €/hora	4.800,00 €
Despliegue	40 horas	30 €/hora	1.200,00 €
Total			8.300,00 €

Bibliografía

[1] Cheang, B., Kurnia, A., Lim, A., & Oon, W. (2003). “On automated grading of programming assignments in an academic institution”. [En línea]. Disponible en: https://doi.org/10.1016/s0360-1315(03)00030-7

[2] GitHub. “UnitTestBot/UTBotCpp: Tool that generates unit test by C/C++ source code, trying to reach all branches and maximize code coverage”. [En línea]. Disponible en: https://github.com/UnitTestBot/UTBotCpp/wiki.

[3] Unikraft. “Testing concepts - Unikraft”. [En línea]. Disponible en: https://unikraft.org/guides/testing.

[4] PrairieLearn. “About | PrairieLearn”. [En línea]. Disponible en: https://www.prairielearn.com/about.

[5] Docker. “Docker: Accelerated Container Application Development”. [En línea]. Disponible en: https://www.docker.com/.

[6] Docker Hub. “Docker Hub Container Image Library | App Containerization”. [En línea]. Disponible en: https://hub.docker.com/.

[7] IBM. “¿Qué es PostgreSQL?”. [En línea]. Disponible en: https://www.ibm.com/es-es/topics/postgresql.

[8] Eman Kashif. “What are include and extend relationships in a use case diagram? ”. [En línea]. Disponible en: https://www.educative.io/answers/what-are-include-and-extend-relationships-in-a-use-case-diagram.

[9] Google for Developers. “Cómo usar OAuth 2.0 para acceder a las API de Google”. [En línea]. Disponible en: https://developers.google.com/identity/protocols/oauth2?hl=es-419.