UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

HERRAMIENTA PARA MEJORAR RETENCIÓN DE ALUMNAS EN TALLER DE PROGRAMACIÓN DE LA ORGANIZACIÓN NIÑAS PRO

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERA CIVIL EN COMPUTACIÓN

FLORENCIA MIRANDA ICAZA

PROFESORA GUÍA:
JOCELYN SIMMONDS WAGEMANN

MIEMBROS DE LA COMISIÓN:
MARÍA CECILIA BASTARRICA PIÑEYRO
ANDRÉS FARÍAS RIQUELME
WILLY MAIKOWSKI CORREA

SANTIAGO DE CHILE
2020

Resumen

Niñas Pro es una ONG que realiza anualmente un curso introductorio de programación en C++ en el que participan mujeres entre 15 y 18 años. Dado el aumento en la cantidad de estudiantes en el curso, el seguimiento y retroalimentación oportuna de parte de quienes las acompañan se hace más difícil. Para solucionar esto, la organización ha introducido el uso herramientas de revisión de código automáticas, pero no están diseñadas para uso pedagógico. Dado esto, la retroalimentación que reciben las estudiantes no es suficiente y las tutoras no pueden realizar el seguimiento adecuado.

En este contexto, la presente memoria introduce una plataforma tecnológica que entrega retroalimentación más completa que la actual a los problemas resueltos por estudiantes y que contiene funcionalidades para la gestión de cursos, aspectos que no son considerados en conjunto en los sistemas de revisión de código, o sistemas de gestión de cursos.

Como base del trabajo se buscó extender un sistema de corrección automática de código existente. Fueron seleccionados para evaluación "Ocimatic", diseñado para evaluar códigos de los competidores de la Olimpiada Chilena de Informática e "EasyProgramChecking" utilizado en el curso Algoritmos y Estructuras de Datos de la FCFM, para que las y los estudiantes tengan una retroalimentación de sus tareas. EasyProgramChecking resultó ser la más adecuada, debido a que parte de la funcionalidad podrá ser reutilizada y que la información se presenta de forma más clara que Ocimatic.

El sistema diseñado permite dar retroalimentación a estudiantes utilizando casos de prueba. Así, las tutoras pueden analizar aquellos que fallan, identificar errores comunes e ingresar sugerencias que pueden servir a otras estudiantes con problemas similares. Con esto, las estudiantes pueden tener una retroalimentación que les sugiere cómo seguir. Además, se diseñó un módulo que permite hacer seguimiento de problemas resueltos y otro para la gestión de asistencia al curso. Así las tutoras pueden hacer seguimiento más fiable del avance de estudiantes y tomar medidas a tiempo, para evitar la deserción de estas. Este diseño fue implementado exitosamente a través de una aplicación web.

El sistema fue validado de dos formas. Primero, con tutoras, a través de dos pruebas de usabilidad, donde se concluye que los módulos de seguimiento y asistencia entregan información útil y de fácil acceso; y que el sistema de retroalimentación, es fácil de comprender, y que será de utilidad para el curso. La segunda validación fue por medio de un caso de estudio exploratorio con estudiantes del curso. En él se buscó entender si la información entregada por el sistema al dar retroalimentación es útil para resolver errores que tienen sus soluciones. Se observó que las estudiantes le dieron gran uso a los casos de prueba y las sugerencias, para entender dónde fallaron, sin embargo, en varios casos la retroalimentación no fue suficiente para resolver el problema planteado sin ayuda.

En resumen, por medio de esta memoria se extendió un sistema de corrección de código, para lograr entregar mejor retroalimentación a las estudiantes y permitir a las tutoras hacer un seguimiento fidedigno de ellas. Como trabajo futuro se sugiere profundizar en el estudio de la interacción de las estudiantes con diferentes tipos de retroalimentación, para mejorar la forma en que se enseña programación; y evaluar el efecto de la plataforma en la deserción.

A mis abuelas y abuelos por heredarme el amor por las matemáticas, la ingeniería y la música.

Agradecimientos

A Mali, Marcelo, Victoria, Diego, Hada, Trini y Pablo por todo el apoyo y cariño que me han dado siempre y sobretodo en estos 7 años de universidad.

A mis amigas Caro, Kathy y Cata por todas las risas, los carretes, los llantos y las horas de estudio; sin ustedes la universidad no sería un recuerdo tan hermoso.

A mis amigos del DCC, Vale, Jorge, Alvaro, Lucas, Dani y Cobo, porque no habría superado estos 5 años de computación sin ustedes y porque sé que seguiremos apoyándonos siempre.

Agradezco a mis amigos de plan común por recibirme en su grupo al final del primer año y por siempre estar dispuestos a compartir sus conocimientos conmigo y el resto del grupo, serán excelentes injenieros.

Quiero mencionar también a la Macumba Batucada, a Oikos y al CADCC 2017 porque ser parte de estos equipos me dio la energía necesaria para sobrevivir estos 7 años de carrera, y porque gracias a ustedes en la universidad aprendí muchas más cosas que matemáticas y computación.

A mis amigas del colegio y mi comunidad, porque por más de 7 años han sido un espacio seguro lleno de risas y amor, donde puedo ser yo misma. Me pone muy feliz ser amiga de personas tan talentosas y agradezco que siempre estén ahí, sin importar cuanto tiempo pase.

A Niñas Pro, por dejarme ser parte de esta organización llena de mujeres secas, y por confiar en mí y en este proyecto, que espero que tenga un impacto positivo en nuestras estudiantes y tutoras. Agradezco a mis compañeras de trabajo en la práctica, Kari, Feña y Anto, por creer en mi proyecto y por darlo todo para que su trabajo en la práctica fuera el mejor.

Finalmente quiero agradecer al Departamento de Computación, a los y las docentes, funcionarios y estudiantes con los que compartí estos últimos 5 años, por hacer que el departamento siempre sea un lugar cómodo. En especial agradezco a Jocelyn Simmonds y Willy Maikowski, mis guías, por toda la buena disposición y apoyo que me dieron durante este proyecto.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Contexto

En los últimos años ha existido un gran aumento en la importancia de las tecnologías y computación en Chile, manteniéndolo como el mejor país latinoamericano en este ámbito según el índice de disposición a la conectividad 2016 (Networked Readiness Index) [4]. A pesar de estar avanzados en temas de tecnologías y computación, se predice que al año 2020 habrá un 38 % de déficit de profesionales especializados en Tecnologías de Información (TI) [7].

Es por este déficit que el año 2013 nace la Olimpiada Chilena de Informática (OCI), la cual es una competencia de programación organizada por la corporación C100 y está orientada a estudiantes de educación secundaria. El objetivo de estas olimpiadas es poder "Difundir la Ciencia de la Computación y la Informática entre las y los estudiantes de secundaria a nivel nacional" y "Descubrir tempranamente, alentar y reconocer, a los jóvenes talentos en la Ciencia de la Computación" [3]. Así al hacerse más conocida el área, se espera que más jóvenes se motiven a estudiar carreras de ciencias, tecnología, matemáticas o ingeniería (STEM por sus siglas en inglés).

Con la OCI también comenzaron variados talleres de programación, orientados a entrenar a escolares para las olimpiadas. Sin embargo, en estos talleres y en la OCI se ha observado una participación muy baja de mujeres. Según datos de la OCI proporcionados por los organizadores, en la primera versión de la competencia no hubo ninguna mujer en la fase nacional y en el año 2015 de las 18 mujeres que participaron en la fase regional en Santiago, sólo 6 pasaron a la nacional, representando un 18 % de las personas que compitieron en esta etapa. La baja participación de mujeres en la OCI también se presenta en diferentes áreas de las STEM. Esto se ve claramente en el proceso de admisión universitaria del año 2017, en el cual un 9,1 % de las mujeres eligieron carreras STEM como informática, electrónica, electricidad, ingenierías, entre otras [9]. El Global Gender Gap Report 2020 [5] señala que "La paridad de género influye fundamentalmente en si las economías y las sociedades prosperan o no. El desarrollo y despliegue de la mitad del talento disponible en el mundo influye enormemente en el crecimiento, la competitividad y la preparación para el futuro de las economías y empresas de todo el mundo". Dado esto, se espera que en todas las áreas del conocimiento la

participación de hombres y mujeres sea la misma, lo que no ocurre en todas las STEM.

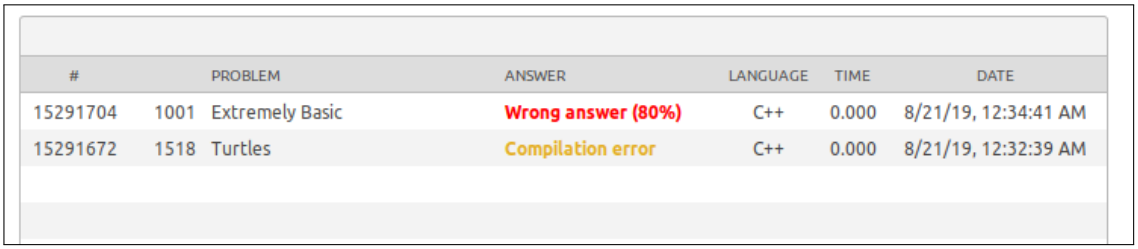
Considerando que se busca que la participación de hombres y mujeres estén en la misma proporción, en la OCI y en los talleres de programación relacionados a esta existe una falta de participación de mujeres. Es por esto que el año 2016, nace la organización Niñas Pro, de la cual quien escribe este documento ha sido participante activa entre los años 2017 y 2020. La misión de Niñas Pro es "empoderar niñas y adolescentes a través de la enseñanza de programación e inspirar vocaciones científicas y tecnológicas". Para cumplir con esta misión, la principal actividad de la organización es realizar un curso anual de programación, enfocado en preparar a las estudiantes para participar de la OCI. El Curso Anual se creó en la Facultad de Ciencias Físicas y Matemáticas (FCFM) de la Universidad de Chile y gracias al crecimiento de la red de voluntariado durante el 2020 el curso también se está realizando de forma coordinada en una segunda sede en Santiago y en las ciudades de Coquimbo y Concepción.

El curso consiste en 18 clases de 3 horas los días sábado en la mañana para niñas entre 15 y 18 años y son impartidas por tutoras que de forma voluntaria participan durante todo el curso. En cada clase hay una cátedra inicial de una hora realizada por una tutora, donde se enseñan conceptos básicos de programación. Luego, de esta cátedra, las estudiantes pasan a un laboratorio de computadores donde durante dos horas, aplican la materia aprendida resolviendo, de forma individual o en parejas, diferentes problemas de programación que se les entregan. En dicho momento las tutoras están en el laboratorio resolviendo dudas de las estudiantes y apoyando en el aprendizaje. Cabe destacar que cada 5 estudiantes se asigna una tutora durante el laboratorio para que el trabajo sea más personalizado.

Durante los 5 años que ha existido la organización se han realizado 6 cursos de programación, con los que se ha llegado a más de 200 jóvenes. En la primera y segunda versión del curso hubo 30 estudiantes respectivamente, en las siguientes 3 versiones se inscribieron entre 60 y 80 participantes en cada una y actualmente participan aproximadamente 90 estudiantes entre las 4 sedes donde se imparte el curso. El aumento sostenido en la cantidad de participantes en los cursos, además de ser muy positivo para los objetivos de la organización, refleja un aumento en el interés por aprender a programar de parte de las jóvenes y el aumento en la visibilidad de esta. Con este aumento también vino un incremento considerable en la cantidad de participantes mujeres en la fase regional de la OCI, en la región Metropolitana, pasando de 18 participantes el 2015 a 46 el año 2018, de las cuales 18 eran estudiantes de Niñas Pro.

Por otro lado, a medida que la cantidad de estudiantes en el curso crece, el tiempo personal que las tutoras invierten en corregir soluciones o tareas, y analizar el avance de cada estudiante, aumenta. Este aumento no tiene una medida cuantitativa, pero si se cuenta con la opinión de las tutoras que se vieron afectadas. Para reducir esta carga se comenzó a utilizar un juez en línea para que las estudiantes pudieran tener feedback de sus soluciones durante la clase. Para esto la organización empezó a utilizar la plataforma *URI Online Judge* (`https://www.urionlinejudge.com.br`). En esta, cada estudiante accede con su cuenta de usuaria y puede subir su solución a los problemas para que sean corregidas automáticamente pasando por un conjunto de casos de prueba. Esta corrección consiste en ejecutar el código con diferentes valores de entrada (también llamados input) para luego comparar el valor de salida (también llamado output) que generó el código con el output esperado para ese caso,

si son iguales se pasa una prueba. Al terminar la evaluación del juez, las estudiantes sabrán si su solución pasó todas las pruebas o si falló. En este último caso solo se sabe el porcentaje de pruebas que fallaron, sin más información (ver Figura 1.1). Participando como profesora del taller, quien escribe este documento ha identificado que al recibir tan poco feedback muchas estudiantes se rinden y cambian a otro problema, dejando muchos problemas incompletos sin aprender de sus errores. Con esto en mente se hizo una revisión de los problemas que las estudiantes del Curso anual 2019 dejaron incompletos, y de un total de 38 problemas incompletos, 26 de estos tuvieron solo un envío. Es decir, que las estudiantes obtuvieron el mensaje "Wrong answer" y no siguieron haciendo mas intentos. Esta práctica, junto con el poco seguimiento que se hace de las estudiantes, lleva a que finalmente muchas abandonen el curso.



#		PROBLEM	ANSWER	LANGUAGE	TIME	DATE
15291704	1001	Extremely Basic	Wrong answer (80%)	C++	0.000	8/21/19, 12:34:41 AM
15291672	1518	Turtles	Compilation error	C++	0.000	8/21/19, 12:32:39 AM

Figura 1.1: Feedback del *URI Online Judge* a una solución.

El problema del seguimiento y feedback a estudiantes no se presenta solo en los cursos de Niñas Pro, sino que otros autores han detectado problemas similares en cursos introductorios de programación. Para esta memoria se estudiaron 5 autores que buscan generar feedback personalizado automáticamente para el estudiantado en cursos de programación. Algunas técnicas que se utilizan son la revisión con casos de prueba [1], análisis automático del código comparándolo con códigos "buenos" [6] [14] o creando modelos que identifican errores comunes [12] y sugieren cambios. En estos estudios se ha observado que los errores que se cometen en problemas introductorios de programación suelen repetirse entre el estudiantado por lo que con la detección de estos se puede mejorar el feedback que se le entrega al estudiantado [12]. Por otro lado, se ha observado que la existencia de herramientas de feedback automático ayuda a disminuir la deserción en cursos de programación [1].

1.2. Problema a resolver

Como se mencionó anteriormente, un gran problema en el curso es que el *URI Online Judge* no es una herramienta pensada para ser usada de forma pedagógica y esto se ve reflejado por un lado en la forma en que las estudiantes interactúan con ella y por otro lado, en la información que entrega a las tutoras sobre los problemas que han resuelto las estudiantes. En la Figura 1.2 está la vista que tienen las tutoras de los problemas resueltos por las estudiantes para una clase. En esta vista se observa de forma general el avance de las estudiantes, pero no se sabe específicamente donde se están equivocando.

Por otro lado, para llevar la asistencia de las participantes del curso, se tiene una planilla en Google Drive como la de la Figura 1.3 donde se tiene el nombre de cada estudiante y la

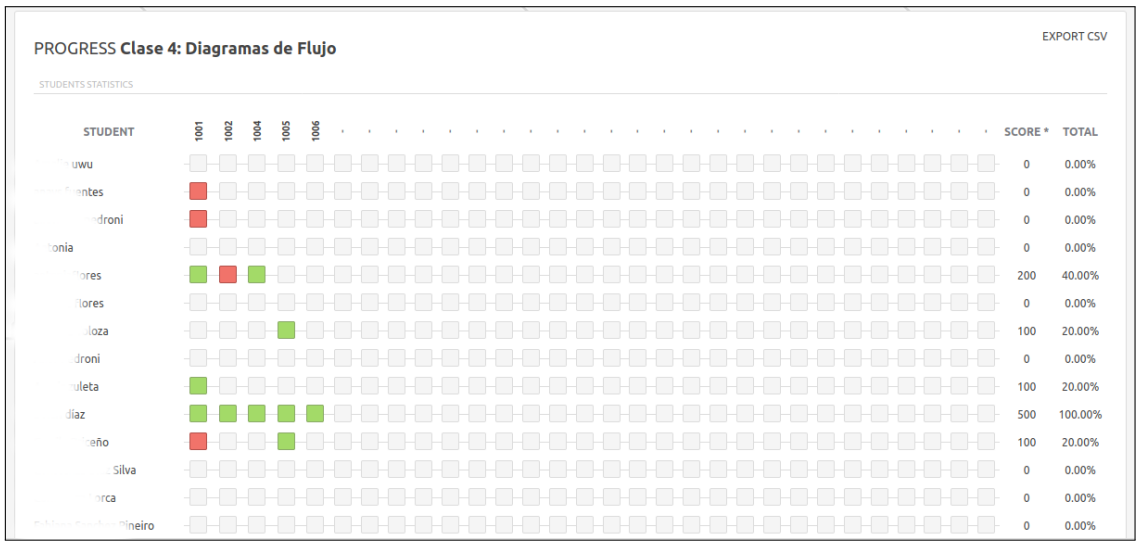


Figura 1.2: Vista en el *URI Online Judge* de una clase y los problemas que ha resuelto cada estudiante.

fecha de las clases de curso. Con esta planilla, cada clase las tutoras anotan quienes asistieron con un 1, pero muchas veces estas olvidan rellenar la información o no hay una forma rápida de hacerlo. Como consecuencia no se tiene información completa y fidedigna sobre lo que ocurre en las clases.

Finalmente esto provoca que las tutoras no sepan quienes son realmente las estudiantes activas en el curso y quienes necesitan más reforzamiento ya que hacer estos análisis requiere más tiempo del que se dispone. Por lo tanto, si una estudiante está desmotivada y pensando en desertar, es probable que las tutoras no sean capaces de advertir esta situación y no puedan tomar acciones a tiempo para evitarlo. Al automatizar el proceso de feedback y seguimiento, se abre la posibilidad, por un lado, de llevar la asistencia y control del nivel de motivación de las estudiantes y, por otro lado, que las tutoras enseñen y las alumnas aprendan de mejor manera el pensamiento computacional.

1.3. Objetivos de la memoria

Al inicio de esta memoria se buscaron sistemas que pudieran resolver el problema que se planteó anteriormente para evitar crear algo que ya existiera. En esta búsqueda se encontraron dos sistemas que permiten pasar uno o más códigos de programación por una serie de casos de prueba y entregan al usuario el resultado obtenido con cada uno de los casos. El primer sistema se llama Ocimatic y fue diseñado para evaluar los códigos de los competidores de la OCI. El segundo sistema se llama EasyProgramChecking y se utiliza en el curso Algoritmos y Estructuras de Datos de la FCFM para que los estudiantes del curso puedan tener feedback de sus tareas antes de enviarlas.

Dado que estas dos herramientas ya implementaron la corrección de código automática, se decidió estudiarlas, compararlas y tomar una para extenderla en esta memoria y crear un

Alumna	06 abr.	13 abr.	20 abr.	27 abr.	04 may.	11 may.	18 may.	25 may.	01 jun.	08 jun.	15 jun.	22 jun.
Alumna 1	1	0		0	0	0	0	0			0	
Alumna 2	1	1		0	1	0	0	0				
Alumna 3	1	1		1	1	1	1	0	1			
Alumna 4	0	1		1	0	0	0	0	1			
Alumna 5	1	1		0	1	1	1	0				
Alumna 6	1	1		0	0	1	1	1				
Alumna 7	1	1		0	1	0	0	0				
Alumna 8	1	1		0	1	1	0	0				
Alumna 9	1	1		1	1	1	1	0	1			
Alumna 10	1	1		1	1	0	1	1	1		1	
Alumna 11	1	1		1	1	0	1	0	1			
Alumna 12	0	1		0	0	1	1	1	1		1	
Alumna 13	1	1		1	1	1	1	1	1		1	
Alumna 14	1	0		0	0	0	0	1			1	
Alumna 15	1	1		1	1	1	1	1	1			
Alumna 16	1	0		0	1	1	1	0	1			
Alumna 17	0	1		1	1	0	0	0				
Alumna 18	1	1		0	0	0	0	0				
Alumna 19	1	1		1	1	0	0	0	1			
Alumna 20	1	1		1	0	1	1	0			1	
Alumna 21	1	1		1	1	0	1	1				
Alumna 22	1	1		0	0	1	0	0				
Alumna 23	1	1		0	1	1	1	1			1	
Alumna 24	1	1		0	1	1	1	1				
Alumna 25	1	1		1	1	1	1	1	1		1	
Alumna 26	1	0		1	1	1	1	1				
Alumna 27	1	1		0	1	1	0	0	1			
Alumna 28	1	1		1	1	0	1	0				
Alumna 29	1	1		1	0	0	0	1				
Alumna 30	1	1		0	0	1	0	1				
Alumna 31	1	1		0	0	0	0	0				
Alumna 32	1	1		1	1	0	0	0	1			
Alumna 33	1	1		1	1	1	0	1				

Figura 1.3: Planilla de asistencia 2019.

sistema que ayude a resolver los problemas en el Curso Anual de Niñas Pro.

1.3.1. Objetivo general

El objetivo general de la memoria es extender una aplicación web de corrección automática de código para mejorar el feedback a las estudiantes del Curso Anual de Niñas Pro y permitir el seguimiento más fidedigno del avance de las participantes.

1.3.2. Objetivos específicos

Para lograr este objetivo general, se requiere cumplir los siguientes objetivos específicos:

1. Evaluar las herramientas Ocimatic e EasyProgramChecking para entender cual se ajusta más a las necesidades de Niñas Pro.
2. Diseñar una funcionalidad que permita saber quiénes son las estudiantes activas dentro del curso, y conocer su avance respecto a los problemas propuestos, para saber a quienes se debe apoyar.
3. Diseñar una funcionalidad que permita entregar un feedback oportuno acogiéndose a la metodología de enseñanza de Niñas Pro.
4. Implementar las funcionalidades necesarias para cubrir las soluciones diseñadas.
5. Evaluar la utilidad de información que entrega cada funcionalidad al actor correspondiente a través de un piloto con estudiantes y tutoras del taller de Niñas Pro.

1.4. Solución propuesta

En base al contexto actual y los problemas que se presentan en el Curso Anual de Niñas Pro, es que se decide crear una aplicación web que extiende un sistema de evaluación de código ya existente, para que genere feedback más completo que el que entrega el *URI Online Judge* utilizado actualmente. Para esto se evaluarán dos aplicaciones ya existentes: Ocimatic que se utiliza para evaluar problemas en la OCI, e EasyProgramChecking que es una aplicación que se utiliza en algunas secciones del curso Algoritmos y Estructuras de Datos dictado en la FCFM, para evaluación de tareas.

El sistema tendrá un módulo de feedback que permitirá subir soluciones en C++ para ser evaluadas automáticamente con casos de prueba creados por las tutoras. A diferencia del *URI Online Judge*, este sistema mostrará a las estudiantes en qué casos fallaron y una descripción de cada caso. Para complementar los casos de prueba, el sistema recolecta los casos en que la estudiante no obtuvo el resultado esperado. Con esta información la tutora podrá ver los valores de salida erróneos que generaron las soluciones de las estudiantes y escribir sugerencias para corregir el error que generó esta salida. Gracias a esto, cuando otra estudiante genere una salida errónea que ya tiene sugerencia, además de tener los casos de prueba, tendrá esta sugerencia para identificar su error. Con esto se espera que las estudiantes tengan una mejor guía de cómo arreglar su solución y además entiendan cómo crear casos de prueba para evaluar su código antes de mandarlo a un juez en línea.

Por otro lado, se creará un módulo de asistencia para que las tutoras puedan visualizar la asistencia general del curso y tengan información fidedigna de esta. El sistema tendrá también, un módulo de seguimiento que complementará información del módulo de feedback con el de asistencia para que las tutoras tengan acceso a un dashboard donde podrán ver de manera generalizada el avance de todas las estudiantes respecto a las clases. Esto permitirá detectar a tiempo a las estudiantes que estén teniendo dificultades, y así las tutoras podrán tomar medidas para ayudarlas y evitar su deserción.

Para acceder a la aplicación habrá que tener una cuenta, por lo que cada participante tendrá información respecto a su avance en el curso y, junto con esto podrán acceder a los códigos de los problemas ya resueltos y ver los que aún no resuelven.

Toda la información de las clases, los problemas a resolver, los casos de prueba, la materia asociada a la clase, entre otros, serán gestionados por las tutoras del curso, gracias a un módulo de cursos.

Finalmente en la tabla 1.1 se encuentran listados los problemas mencionados anteriormente y una breve explicación de cómo cada módulo aporta a resolver el problema. En esta tabla no se incluye el módulo de cursos ya que este es transversal a todos los módulos y problemas. Gracias al módulo de cursos se tendrá información personalizada de cada estudiante permitiendo la existencia del módulo de seguimiento, asistencia y feedback.

Problema	Módulo	Porqué el módulo ataca el problema
No se conoce el avance de cada estudiante.	Seguimiento	Con el módulo de seguimiento las tutoras verán los problemas resueltos por cada estudiante y los que están incompletos, basado en la información que recopila el módulo de feedback. Con esto, podrán identificar a quienes requieren apoyo, y ayudarlas.
No se sabe de forma fidedigna qué materia y problemas están siendo más difíciles para las estudiantes.	Feedback, Seguimiento	Gracias al módulo de feedback las tutoras tendrán información sobre los errores más frecuentes cometidos por estudiantes. Por otro lado, con el módulo de seguimiento las tutoras sabrán qué problemas han tenido menos respuestas. Con esta información podrán identificar de forma temprana los conceptos que hay que reforzar.
La plataforma utilizada actualmente entrega feedback incompleto.	Feedback	El módulo de feedback entregará información más completa de la que reciben actualmente mostrando los casos en que su solución falló y sugerencias en caso de tener un error común.
No se sabe que estudiantes siguen en el curso y quienes no.	Asistencia	Gracias a este módulo se tendrá información fidedigna de la asistencia de las estudiantes.
Se olvida pasar la asistencia.	Asistencia	El sistema pasará a ser parte de las clases, por lo tanto se espera que al estar interactuando con el sistema durante toda la clase, las tutoras recuerden pasar la asistencia.

Tabla 1.1: Tabla resumen de los problemas a resolver y cómo los módulos propuestos aportan a esto.

1.5. Metodología

Para desarrollar la memoria se utilizará un desarrollo iterativo e incremental. Inicialmente se crearán mockups de las interfaces a desarrollar, según las historias de usuario planteadas, luego, estos serán revisados por los profesores a cargo de esta memoria y serán mejorados. Luego, se implementarán los módulos propuestos, revisando semanalmente el avance del proyecto con los profesores a cargo para mejorar y extender la funcionalidad.

Para validar el trabajo realizado, se hará un caso de estudio exploratorio con estudiantes del curso para ver cómo estas interactúan con la plataforma y ver si la información entregada es útil. Por otro lado, se harán dos estudios de usabilidad con tutoras del Curso Anual para medir la usabilidad de la plataforma.

1.6. Estructura del informe

El resto del documento consta de cinco capítulos. En el Capítulo 2 se explica un problema de programación competitiva para dar contexto de los problemas a los que se enfrentan las estudiantes, luego se realiza un análisis de bibliografía y finalmente se evalúan las herramientas Ocimatic e EasyProgramChecking para ver cuál extender en esta memoria.

En el Capítulo 3 se definen las historias de usuario para la herramienta a desarrollar, se hace un análisis del sistema a extender y se realiza el diseño del nuevo sistema, y luego,

en el Capítulo 4 se explica la implementación del sistema diseñado y los desafíos que se encontraron.

En el Capítulo 5 se muestran los resultados de la validación realizada y finalmente, en el Capítulo 6 se muestran las conclusiones del trabajo realizado y el trabajo futuro.

Capítulo 2

Marco teórico

Debido al gran aumento en el interés por aprender programación en todo el mundo es que se hace necesario generar herramientas de feedback automatizado y personalizado. Dado esto, diversos autores han estudiado el problema y han generado herramientas que lo abordan desde diferentes aristas.

En la Sección 2.1 de este capítulo se explica un problema de programación competitiva para dejar claro el contexto en que surgen estos sistemas, y las dificultades que presentan las estudiantes con estos. En la Sección 2.2 se analizan algunas de las investigaciones mencionadas antes y sus resultados, para tenerlos en cuenta a la hora de desarrollar esta memoria, y en la Sección 2.3 se evalúan dos herramientas existentes de corrección de código para ser utilizadas en este trabajo.

2.1. Ejemplo de problema

En esta sección se explicará un problema llamado "Calculadora" que se obtuvo de la fase regional de la OCI 2017. Este consiste en que dos amigos (Mauricio y Camilo) discuten si un número en la pantalla de una calculadora representa el mismo valor al dar vuelta la calculadora en 180 grados o no. Mauricio dice que sí y Camilo dice que no. En la Figura 2.1 se ven ejemplos de números que se ven iguales al girar la calculadora en 180 grados (o que son invertibles) y en la Figura 2.2 se encuentran números que no representan el mismo valor al dar vuelta la calculadora en 180 grados.

El valor de entrada del problema es un número entero N, que representa la cantidad de dígitos que se ingresaron en la calculadora y luego vienen N números enteros que representan los dígitos que fueron ingresados. Por ejemplo, si se quiere probar con el número 906, la entrada será *3 9 0 6*. El número 3 representa que el valor a ingresar tendrá 3 dígitos, y luego el 9, 0 y 6 serán los tres dígitos del número 906.

La salida de este problema será una línea que dirá "Mauricio" o "Camilo" según quién tenga la razón. En el ejemplo anterior para el número 906 el resultado será Mauricio, ya que

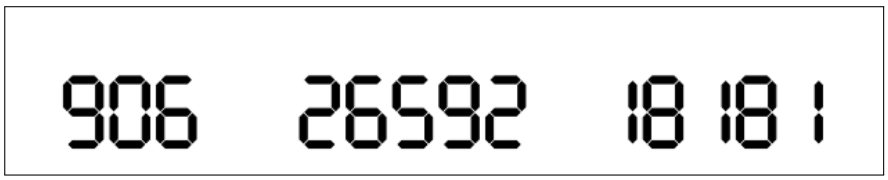


Figura 2.1: Números que tienen el mismo valor al rotar la calculadora.

Figura 2.2: Números que no tienen el mismo valor al rotar la calculadora.

el número tiene el mismo valor al girar la calculadora.

En general un caso de prueba tendrá un *input* que será la entrada con que se probará este caso, un *output* que será la salida esperada para ese caso. En este ejemplo se incluye una categoría que permitirá agrupar los casos de prueba.

El caso más básico es que el número tenga un solo dígito, ya que solo hay que ver si es invertible o no, los dos primeros ejemplos del Código 2.1 representan este caso. Un caso un poco más complejo es cuando son dos dígitos ya que hay que revisar que un dígito sea el inverso del otro, en el Código 2.1 el segundo y tercer caso tienen dos dígitos. Otro caso es cuando se tiene un número con 3 dígitos ya que el del centro debe ser invertible y los números de la izquierda deben ser el inverso de los de la derecha, en el Código 2.1 los tres últimos casos tienen 3 dígitos. Finalmente, para casos más complejos se necesita hacer ciclos para ir revisando si los dígitos de la primera mitad del número son los opuestos de la segunda mitad del número.

Al resolver un problema como este, un error común que se comete es confundir los límites que se le asignan a los ciclos, haciendo que el programa haga más iteraciones de las que debería o haciendo que termine antes de recorrer todos los dígitos. Por otro lado, en el enunciado no se explicita qué números son invertibles, ni qué números son opuestos, por lo tanto una estudiante podría olvidar considerar algún dígito como invertible y entregar una solución equivocada.

Finalmente, es importante entender la estructura de los ejercicios de programación competitiva y qué dificultades tienen las estudiantes al resolver un problema de este estilo para saber cómo abarcar el sistema a desarrollar.

```
{
    "Input":"1 1",
    "Output": "Mauricio",
    "Categoria": "Un digito"
},
{
    "Input":"1 2",
```

```
    "Output": "Mauricio",
    "Categoria": "Un digito"
},
{
    "Input":"2 2 5",
    "Output": "Camilo",
    "Categoria": "Dos digitos"
},
{
    "Input":"2 5 2",
    "Output": "Camilo",
    "Categoria": "Dos digitos"
},
{
    "Input":"3 6 4 9",
    "Output": "Camilo",
    "Categoria": "Tres digitos"
},
{
    "Input":"3 4 7 6",
    "Output": "Camilo",
    "Categoria": "Tres digitos"
},
{
    "Input":"3 2 4 5",
    "Output": "Camilo",
    "Categoria": "Tres digitos"
},
```

Código 2.1: Ejemplos de casos de prueba para problema Calculadora.

2.2. Análisis de bibliografía

Un sistema de Reparación Automática de Código (APR por su sigla en inglés) es una tecnología que se utiliza hoy en día para corregir bugs automáticamente en un software. Yi et al. [15] estudió cómo utilizar un APR en sistemas de tutoría inteligente, utilizando códigos obtenidos de un curso introductorio de programación del *Indian Institute of Technology Kanpur*, concluyendo que estos sistemas no pueden reparar completamente las soluciones enviadas por el estudiantado ya que estas suelen necesitar muchos cambios, y los APR solo hacen pequeños cambios. Para abordar esto, proponen dar al estudiantado sugerencias para hacer arreglos parciales del código y así generar que avancen hacia la respuesta correcta por pasos. Es en base a este estudio que para esta memoria se desarrollará una herramienta que entregue sugerencias a las estudiantes para guiarlas a reparar errores asociados a un caso de prueba específico. Así, no se les dará la respuesta al problema sino que se les guiará a mejorar su solución. Se cree que con esta forma de dar feedback, se promueve mejor que las estudiantes generen pensamiento computacional y no solo tengan muchos problemas resueltos.

Otro aporte de este estudio, es que se observó que los profesores auxiliares del curso de introducción a la programación estudiado hacían uso más eficiente de las sugerencias que el estudiantado, ya que estos últimos tienen problemas para entender y saber cómo utilizar las sugerencias. En la presente memoria se buscará dar feedback en lenguaje natural, de tal forma que las estudiantes puedan comprenderlo, pero si alguna estudiante no lo logra, se espera que las tutoras comprendan el feedback fácilmente y podrán ayudarlas eficientemente.

Para esta memoria se revisaron 5 diferentes autores que buscan dar feedback automático en cursos de programación, pero la mayoría tenían una alta complejidad debido a que buscan indicar exactamente qué línea de código cambiar para llegar a una buena solución [6] [14]. Dado que en Niñas Pro se entrega educación personalizada se cree que por el momento no es necesario tener una herramienta con feedback tan específico, además, como se dijo anteriormente, no se cree que al entregar la respuesta completa al error de una estudiante, ella esté generando pensamiento computacional.

En general, para problemas introductorios de programación se ha observado que los errores que presentan las soluciones de estudiantes suelen repetirse y esto fue demostrado por Singh et al en [12]. Singh et al. crea un modelo para que los instructores del curso definan potenciales correcciones que las soluciones de los estudiantes podrían necesitar y en base a estas el sistema sugerirá cómo reparar el código de la solución. En aquel trabajo se utilizaron más de mil soluciones creadas por estudiantes del curso de Introducción a la Programación en MIT, para medir la eficacia del sistema, obteniendo que un 64 % de las soluciones pudieron ser arregladas por el sistema. Para la presente memoria se ven dos limitaciones a este modelo; la primera es que como se dijo antes, el feedback que entrega este sistema consiste en decir exactamente qué línea de código arreglar y esto no es lo que se busca en el curso de Niñas Pro. Por otro lado, se necesita tener ejemplos de códigos entregados por estudiantes para analizarlos e identificar errores comunes para luego poblar el modelo de errores. Considerando esto es que el sistema a desarrollar en la presente memoria guardará los *outputs* erróneos recibidos para cada caso de prueba y su frecuencia. Así, cuando haya un *output* erróneo con frecuencia alta se sabrá que es un error común y se podrá agregar manualmente una sugerencia para apoyar a las estudiantes en futuros intentos.

Varios autores [12] [15] [6] mencionan que para soluciones muy incorrectas, es difícil que un sistema de feedback automático pueda generar sugerencias que permitan llegar la solución correcta. Esto no sería un problema en Niñas Pro debido a que se cuenta con una tutora por cada 5 estudiantes, por lo que en casos extremos la tutora podrá guiar a la estudiante a desarrollar una mejor solución si la ayuda entregada por sistema no es suficiente.

Un buen referente para esta memoria será el trabajo realizado por Benotti et al. [1] en el cual se generó una herramienta web llamada Mumuki, que entrega feedback automático para problemas de programación y es utilizada en universidades y colegios de Argentina. En la Figura 2.3 se encuentra una captura de pantalla de la retroalimentación que entrega el sistema Mumuki. Esta herramienta hace una evaluación con casos de prueba y entrega como feedback los casos que pasaron y los que no. Además, para asegurar que el estudiantado utilice los patrones o conceptos que el docente espera, el sistema hace un análisis sintáctico del código. En dicha investigación probaron el sistema en dos cursos de programación con Haskell de universidades diferentes, y encontraron que la cantidad de estudiantes que desertaron en

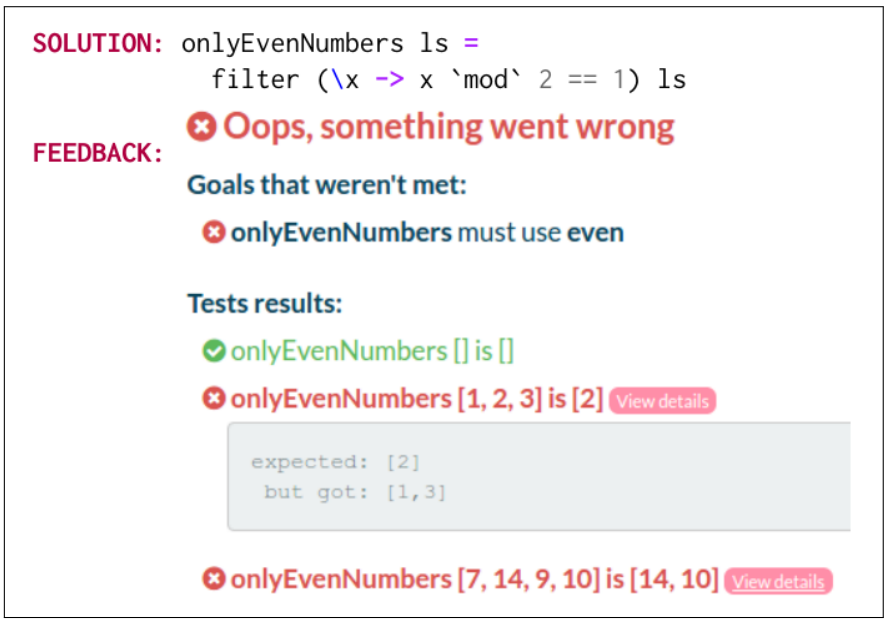


Figura 2.3: Ejemplo de feedback entregado por herramienta Mumuki. Fuente: [1]

los cursos disminuyó considerablemente en relación a la tasa de deserción de los dos años anteriores. El estudio indica que la única diferencia entre las diferentes versiones de los cursos fue que en la última versión se utilizó Mumuki y en las versiones anteriores el estudiantado probaba sus soluciones directamente en la consola. Esta es una motivación para esta memoria y en el futuro se espera poder hacer un estudio como el de Benotti et al. con las diferentes sedes del curso de Niñas Pro y evaluar el nivel de deserción.

Keuning et al.[8] realizaron una revisión sistemática de literatura sobre feedback automatizado para ejercicios de programación. Con esta revisión los autores identificaron más de cien herramientas diferentes para esta tarea. Algunas conclusiones a las que se llegó es que la retroalimentación se centra principalmente en identificar errores y no tanto en arreglarlos, por otro lado los profesores no pueden adaptar fácilmente las herramientas a las necesidades de sus cursos. Otras conclusiones son que la diversidad de tipos de retroalimentación ha aumentado en las últimas décadas y cada vez se están desarrollando nuevas técnicas que favorecen a los y las estudiantes. Dado que la literatura no incluye mucha información acerca de cómo es la interacción de profesores con las herramientas, se hace difícil elegir qué plataforma se ajustará más al Curso Anual de Niñas Pro y a los temas que se pretende abordar en esta memoria. Dado esto, se decide tomar herramientas de revisión automática de código existentes y de código abierto, y adaptarlas a las necesidades de Niñas Pro, teniendo en consideración los elementos de la literatura mencionados en esta sección.

2.3. Evaluación de herramientas existentes

Tal como se mencionó en la Sección 1.3 se estudiaron dos herramientas, Ocimatic e Easy-ProgramChecking, que implementan la corrección de código automática y se elegirá una para ser extendida en esta memoria. Ambas herramientas pueden ser extendidas para este proyecto

ya que tiene a disposición su código fuente.

A continuación se presenta una breve descripción de Ocimatic e EasyProgramChecking para entender su funcionamiento. Luego, se explican los criterios que se utilizaron para la evaluación y los resultados de esta.

2.3.1. Herramientas a evaluar

2.3.1.1. Ocimatic

Ocimatic es una herramienta diseñada para la OCI que permite pasar uno o más códigos escritos en Java o C++, por una serie de casos de prueba y entrega el resultado obtenido con cada uno de los casos.

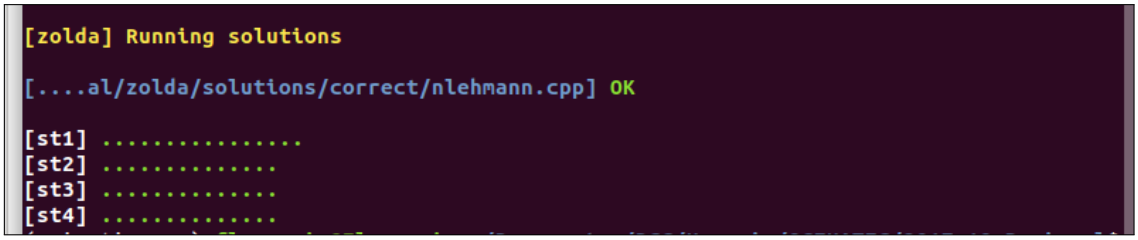


Figura 2.4: Evaluación de un código con Ocimatic en la terminal.

La creación de casos de prueba se hace con códigos que permiten crear *inputs* con números aleatorios y luego en base a una solución correcta del problema, se crean los *outputs esperados*. Este proceso se hace desde la terminal del computador. Por otro lado, la revisión de soluciones se pueden hacer en la terminal del computador o en una aplicación web, la cual está desarrollada en Python con el framework Flask.

El resultado de revisar una solución usando la terminal se ve en la Figura 2.4, aquí el problema se llama " zolda", los casos se dividieron en 4 categorías (st1,st2,st3,st4) y todos los casos fueron aprobados ya que los puntos son verdes.

En la Figura 2.5 se encuentra la vista de la aplicación web para revisar una solución. A la izquierda está la opción de elegir un problema, el lenguaje de programación de la solución y se puede insertar el código de la solución. A la derecha está el resultado de la ejecución del código con los casos de prueba.

Ocimatic se debe ejecutar dentro de un directorio que contenga los códigos para la creación de casos de prueba de los problemas y las soluciones a revisar. Como este sistema está diseñado para la OCI, por cada fase de la competencia habrá un directorio diferente que contiene todos los problemas de esa fase.

Finalmente con Ocimatic se puede obtener información sobre la cantidad de casos que pasaron, cuántos fallaron y el tiempo que demoró la ejecución.

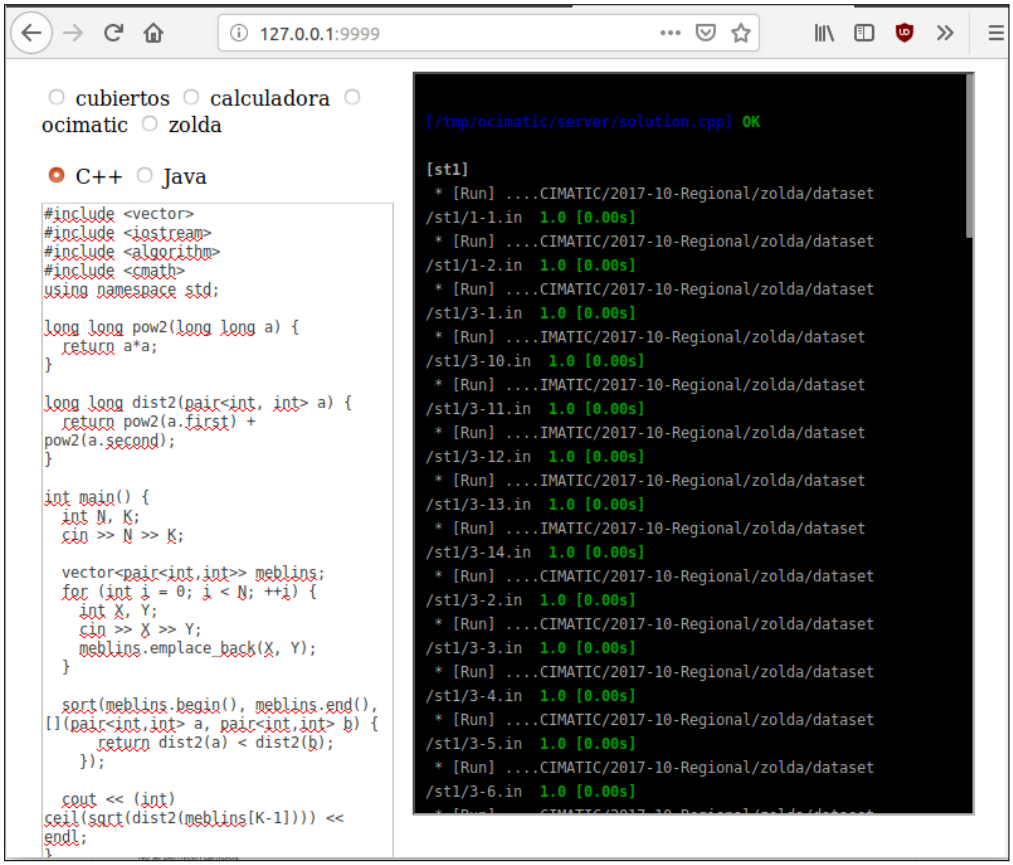


Figura 2.5: Evaluación de un código con Ocimatic en la versión de navegador.

2.3.1.2. EasyProgramChecking

La segunda herramienta en evaluación se llama EasyProgramChecking. Esta es una aplicación web para corrección de tareas en Python utilizada actualmente para el curso de Algoritmos y Estructuras de Datos en la FCFM. La herramienta permite a los ayudantes del curso probar las tareas del estudiantado, con ciertos casos de prueba y determinar la cantidad de casos que fallan y los que no, para establecer una nota. Por otro lado, el estudiantado puede acceder a la herramienta para probar la solución que ha creado para su tarea. Esta herramienta fue creada por un equipo del curso Ingeniería de Software 2, de la misma facultad, para el profesor Jérémy Barbay.

EasyProgramChecking está desarrollada en Python con el framework Django. La aplicación permite el manejo de usuarios, creación de cursos y la asignación de alumnos y profesores a cada uno de estos. Cada curso tiene "assignments" (o tareas). Para cada tarea se puede ver su enunciado y subir una solución al sistema para que sea probada con los casos de prueba que el profesor le asignó (ver Figura 2.6). Las soluciones a las tareas pueden ser en Python 2 o Python 3.

Luego de enviar una solución el usuario obtiene los resultados en una vista como la de la Figura 2.7. En esta vista el usuario obtendrá información sobre los test (o casos de prueba) que falló y los que pasó, información sobre cada caso y finalmente al hacer click sobre la

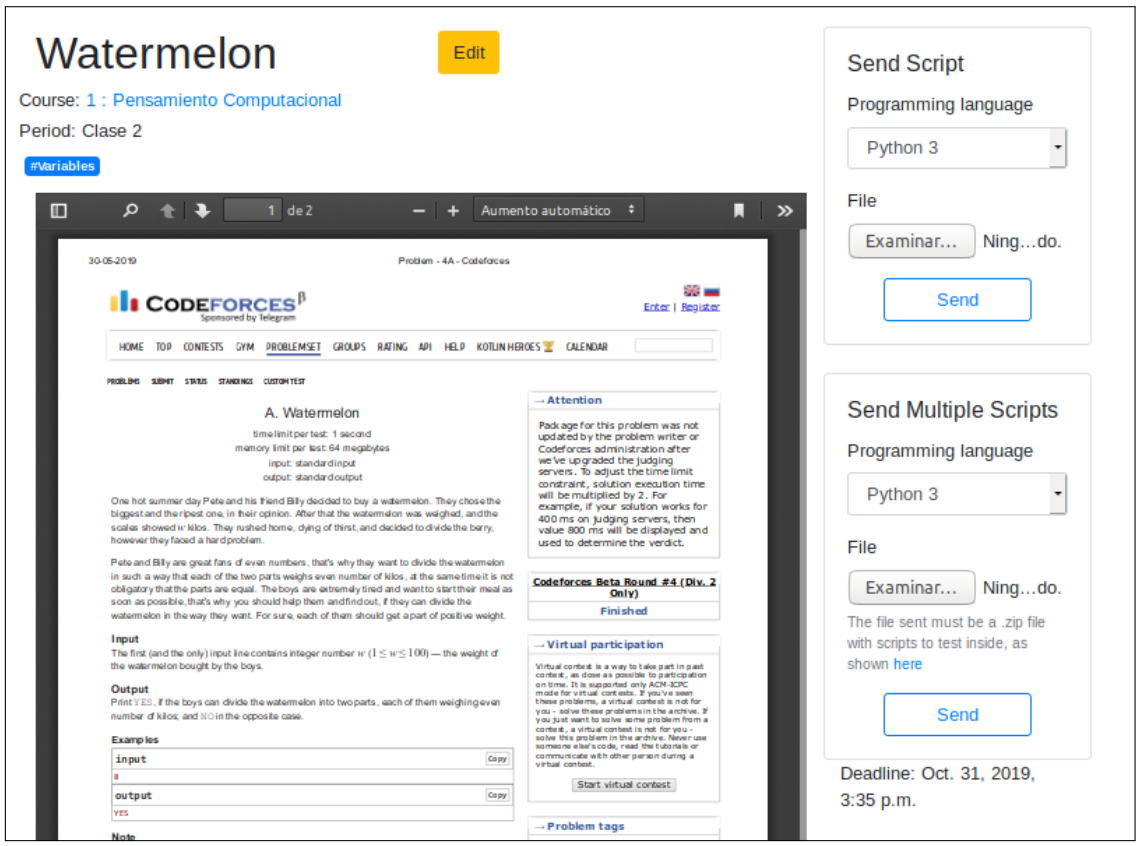


Figura 2.6: Vista de una tarea en EasyProgramChecking.

última columna se tiene acceso total o parcial al *input* que se utilizó para el caso, el *output esperado* y el *output* que entregó el programa. Como se ve en la Figura 2.6 existe la opción de subir múltiples códigos y obtener feedback de todas las soluciones.

2.3.2. Criterios de evaluación

Para la evaluación de las herramientas se seleccionaron tres criterios que se describen a continuación, las definiciones de los criterios fueron tomadas del libro "Ingeniería del software, Un enfoque práctico" de Roger S.Pressman [10] :

Mantenibilidad y extensibilidad: Pressman indica que algunas características de un software mantenible son la modularidad, el uso de patrones de diseño que permitan facilidad de comprensión y que haya sido construido con estándares de codificación, generando un código comprensible. Dada esta definición se evaluarán estas características para cada plataforma. Además, se evaluará la documentación disponible.

Reusabilidad: Pressman define reusabilidad como: "Grado en el que un programa (o partes de uno) pueden volverse a utilizar en otras aplicaciones". Para esta memoria es necesario evaluar los elementos ya existentes en cada herramienta que puedan ser reutilizados para lograr los objetivos de la memoria.

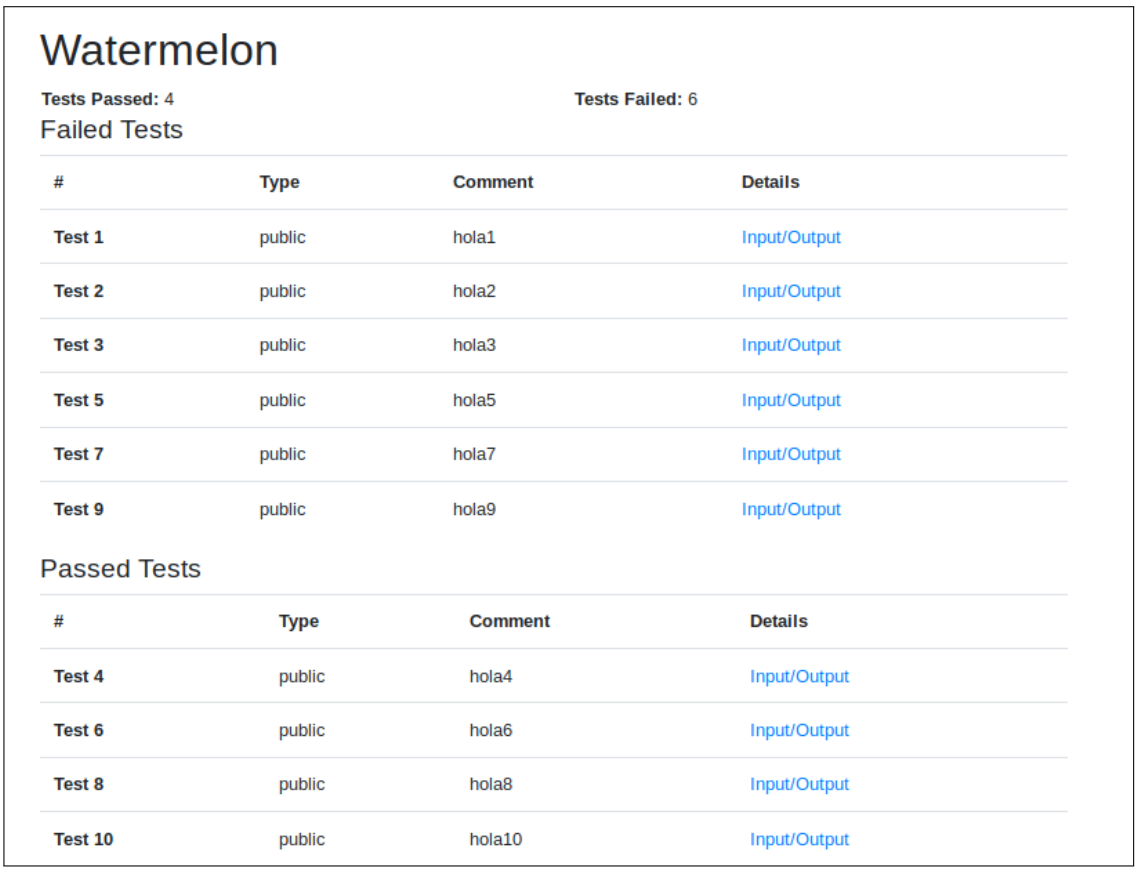


Figura 2.7: EasyPrograChecking : vista del feedback a una solución.

Usabilidad: Se define usabilidad como el "Esfuerzo que se requiere para aprender, operar, preparar las entradas e interpretar las salidas de un programa". Es importante medir la usabilidad de las herramientas ya que serán la base del proyecto de la memoria. Se necesita que la herramienta a extender sea usable, tanto para estudiantes como para tutoras.

2.3.3. Resultados de la evaluación

Para la evaluación de las herramientas se pondrá un puntaje de 1 a 5 a cada criterio mencionado en el punto anterior, siendo 1 el puntaje más bajo y 5 el más alto. Luego, se suman los puntajes y la herramienta con el puntaje más alto, será considerada la más adecuada para ser utilizada en esta memoria.

Existe un factor en la evaluación que es la cercanía con los desarrolladores originales de las herramientas. Al momento de comenzar a entender las herramientas la memorista estuvo en comunicación con los desarrolladores, y dado que los creadores de EasyProgramChecking son estudiantes del mismo departamento, la comunicación fue más fluida que con los desarrolladores de Ocimatic, los cuales se encuentran en Estados Unidos y no son conocidos de la memorista.

En la Tabla 2.1 se encuentra un resumen de los puntajes asignados, según los criterios establecidos. De la evaluación se concluye que la herramienta más apropiada para ser extendida en esta memoria es EasyProgramChecking ya que obtuvo 5 puntos más que Ocimatic.

	Ocimatic	EasyProgramChecking
Mantenibilidad y extensibilidad	1	3
Reusabilidad	3	4
Usabilidad	2	4
Total	6	11

Tabla 2.1: Resultados de la evaluación del sistema a extender.

A continuación se describe la evaluación que se realizó de cada herramienta.

2.3.3.1. Evaluación final Ocimatic

Mantenibilidad y extensibilidad:

En primer lugar, la única documentación que existe sobre Ocimatic es un archivo README con algunas instrucciones de uso. Al empezar a utilizar Ocimatic, la memorista tuvo que consultar a los creadores de la herramienta cómo usarla, ya que estas instrucciones no fueron suficientes para poder crear casos de prueba y revisar una solución en la terminal. En segundo lugar, el código que se utiliza para ejecutar la herramienta en la terminal contiene 268 líneas de código de las cuales sólo 6 son comentarios, lo cual hace difícil comprender cómo opera y entender toda la funcionalidad que esta tiene. Estas dos situaciones hacen que sea difícil familiarizarse con el código del sistema, volviéndolo poco mantenible y extensible.

Por otro lado, Ocimatic tiene una arquitectura cliente-servidor, donde el cliente será la aplicación web o las instrucciones de la terminal para revisar una solución, y el servidor será el sistema que revisará el código. A pesar de que el sistema tiene estas dos partes, esto no se ve reflejado en la forma en que están distribuidos los archivos. Como se ve en la Figura 2.8 el código del proyecto consta de un solo directorio con 14 archivos de extensión ".py" que contienen más de 1600 líneas de código, donde se encuentran los archivos para ejecutar la aplicación web, el archivo main.py que permite ejecutar los comandos en la terminal, junto con los archivos para la revisión de código. De todos estos archivos, 11 son parte del sistema que corrige código y este consta de 1.136 líneas de código. Esto afecta la extensibilidad y mantenibilidad del sistema ya que hay mucho código que mantener actualizado y no existe una distribución clara de los módulos del sistema.

Al analizar el código, se concluye que la aplicación web está desarrollada con el framework Flask, el cual, según su documentación[1], permite extender fácilmente las aplicaciones integrando extensiones. Esto es positivo ya que se podría agregar a la aplicación web algunas funcionalidades necesarias para cumplir el objetivo de la memoria.

[1] `https://palletsprojects.com/p/flask/`

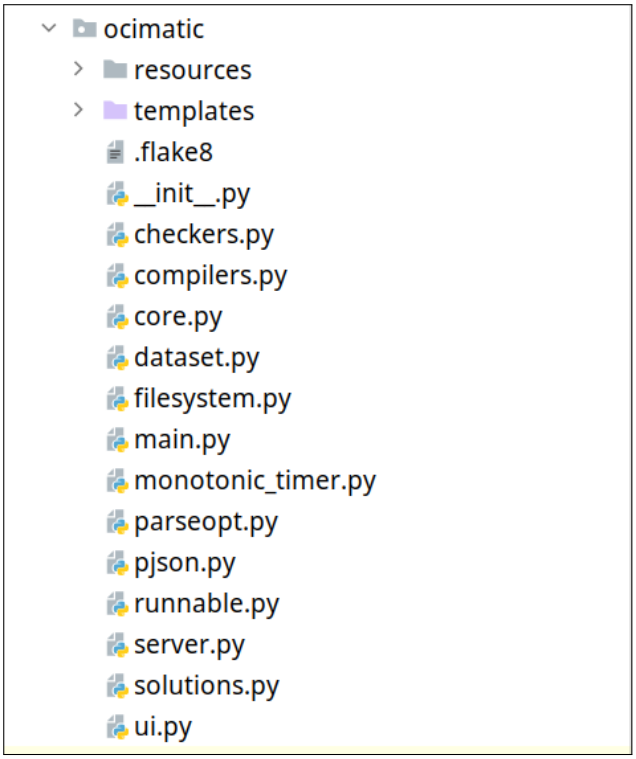


Figura 2.8: Estructura del proyecto Ocimatic.

En conclusión dado que 1) el sistema no es modular, 2) el código no cuenta con suficientes comentarios para ayudar a la comprensión, 3) el uso del patrón de diseño cliente-servidor no es claro en el código y 4) la documentación no es suficiente para utilizar el sistema fácilmente y 5) el framework Flask permite extensibilidad; se le asignó 1 punto en mantenibilidad y extensibilidad a Ocimatic.

Reusabilidad:

Como Ocimatic está diseñado en el contexto de la OCI se pueden corregir problemas escritos en Java o C++. En el taller de Niñas Pro se utiliza C++ por lo que la base de la funcionalidad de corrección ya está lista.

En el Curso Anual de Niñas Pro los problemas que las estudiantes realizan están separados según el día de clases en que deben desarrollarse. En el caso de Ocimatic se podría separar las clases como si fueran fases de la competencia e incluir los problemas correspondientes, sin embargo, el sistema solo puede ser ejecutado en una fase a la vez. Esto es una desventaja ya que se podría pensar en ejecutar Ocimatic para cada clase y solo mostrar los problemas de esta, pero si una estudiante quisiera revisar los problemas que trabajó en otra clase no los encontraría.

Respecto a la aplicación web, ésta permite corregir código desde un archivo o también escribir código en el editor que trae la aplicación, por lo que cumple con la funcionalidad mínima para el sistema que se implementará.

En conclusión, la evaluación que reciben los elementos ya existentes de Ocimatic es 3 ya

que soporta los archivos escritos en C++ y tiene una versión web básica, pero solo es posible mostrar los problemas de una fase a la vez, lo cual es una gran desventaja.

Usabilidad:

La versión de la terminal de Ocimatic podría ser difícil de utilizar por las estudiantes, ya que manejar la terminal exige una mínima cantidad de experiencia que, en un principio, las estudiantes de Niñas Pro no tienen. Para lograr solucionar esto se debería enseñar a las estudiantes a utilizar el programa, lo cual generaría una pérdida de tiempo para los talleres. Sin embargo, existe la versión de navegador que contiene una interfaz sencilla para subir una solución.

A pesar de esto, en ambas versiones (Figura 2.4 y Figura 2.5), la forma en que se muestra el feedback no es intuitiva, ya que no aparecen el *input* y *output* que se utilizaron en cada caso, sino que se ve un una secuencia de caracteres asignada al caso de prueba.

Por el lado de las tutoras, el sistema necesita varios pasos para crear casos de prueba para un problema y estos pasos solo se pueden realizar en la terminal. Dado que la mayoría de las tutoras son programadoras, no debería haber problemas con utilizar la terminal para esto. Sin embargo, se espera que las tutoras utilicen la aplicación web para revisar los resultados de las estudiantes y su avance, por lo tanto sería recomendable tener todas las funcionalidades del sistema en una sola plataforma y no en dos plataformas como hace actualmente Ocimatic.

Finalmente la evaluación de la usabilidad de Ocimatic es 2 ya que a pesar de que la versión web si podría ser utilizada por estudiantes principiantes, el feedback que entrega la herramienta es difícil de comprender, tanto en la versión web como en la versión de terminal. Por otro lado, que las tutoras tengan que utilizar la versión web y la versión de terminal del sistema no es recomendable en términos de usabilidad.

2.3.3.2. Evaluación final EasyProgramChecking

A continuación se presenta la evaluación de cada criterio para EasyProgramChecking.

Mantenibilidad y extensibilidad:

Este sistema tiene una arquitectura de cliente-servidor, en que el cliente será una aplicación web en Django y el servidor será el sistema que ejecuta las soluciones con los casos de prueba. En la Figura 2.9 se encuentra la estructura de los archivos de EasyProgramChecking, aquí el directorio scriptserver tiene todo el código relacionado con el servidor y los otros directorios forman parte de la aplicación web del cliente. Esta estructura es modular, ya que no depende de qué sistema está en el cliente, solo basta que se envíe la información correcta al servidor para ejecutar el código con sus pruebas.

Respecto al servidor, éste está formado por dos módulos. Un módulo para la comunicación con el cliente (directorio communication en Figura 2.9) y un módulo para ejecutar la solución con los casos de prueba (directorio testing en la Figura 2.9). El código de estos dos módulos está ordenado en directorios diferentes lo que hace que las funcionalidades estén bien divi-

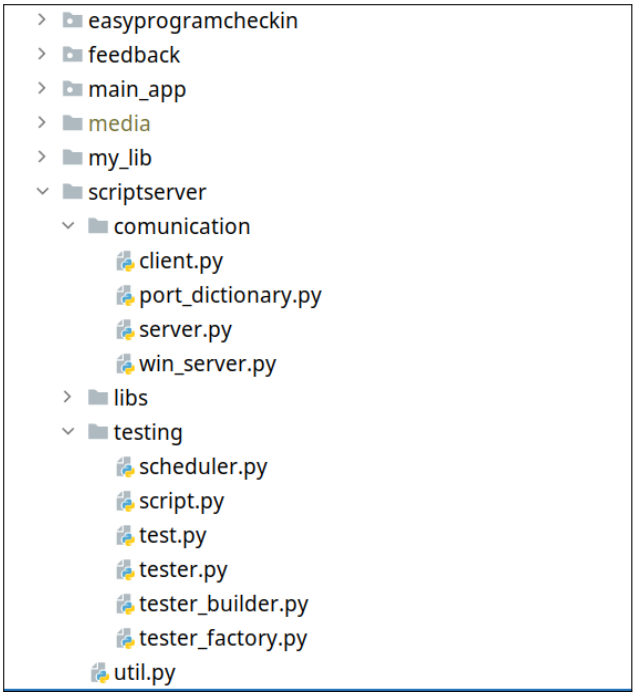


Figura 2.9: Estructura del proyecto EasyProgramChecking.

didas. En total estos dos módulos están formados por 410 líneas de código, sin embargo el código no trae mucha documentación, por lo que es difícil familiarizarse con éste para poder extenderlo.

La aplicación web de este sistema está formada por tres módulos: uno de cursos, uno de tareas y uno de feedback. Esto le da extensibilidad al sistema, ya que para modificar la funcionalidad se trabajará solo dentro de un módulo y para agregar nueva funcionalidad se pueden añadir módulos sin afectar los existentes.

En conclusión, el servidor de esta herramienta está estructurado de forma modular y no tiene muchas líneas de código, pero no cuenta con suficiente documentación, por lo que extenderlo no será sencillo. También, cuenta con una aplicación web en el cliente con módulos claros y desarrollada con un framework de desarrollo afín a la memorista, por lo que esta parte es más extensible. Dado lo anterior se le asignó una nota 3 a la mantenibilidad y extensibilidad de EasyProgramChecking.

Reusabilidad:

Respecto a la corrección de código, EasyProgramChecking sólo puede ejecutar código en Python, por lo tanto se necesitaría agregar los comandos para ejecutar código en C++.

Respecto a la aplicación web, esta permite tener sistema de usuarios y roles para estos, además de la creación de cursos y tareas. Para esta memoria es importante que el sistema permita el seguimiento de las estudiantes, por lo tanto esta aplicación es una buena base para lo que se quiere lograr.

Dado lo anterior, se asignan 4 puntos a EasyProgramChecking por Reusabilidad.

Usabilidad:

EasyProgramChecking es una aplicación web que está en inglés, lo cual es una desventaja ya que las estudiantes de Niñas Pro no necesariamente saben inglés, y esto podría dificultar su aprendizaje en los talleres, haciéndolo más complejo para las que tengan barreras de idioma.

Respecto al feedback, este se presenta de una forma limpia y clara (ver Figura 2.7), entregando detalles de cada caso de prueba. Esto es una gran ventaja ya que parte del problema que se requiere solucionar es que las herramientas que se utilizan actualmente no entregan información sobre los casos de prueba a los que se somete una solución.

En EasyProgramChecking los casos de prueba se definen en un archivo de extensión ".json", ".csv" o ".yml" y luego se suben al sistema a través de la aplicación web. Este proceso requiere trabajo de las tutoras para crear los casos de prueba, pero una vez creados, es simple e intuitivo subirlos al sistema.

Según lo mencionado anteriormente el sistema de EasyProgramChecking logra corregir código y entregar un feedback de forma clara y simple, sin embargo el sistema está en inglés por lo que se le asignan 4 puntos a la Usabilidad.

2.3.3.3. Comentarios finales de la evaluación

Dada la evaluación que se realizó de cada herramienta, se extenderá EasyProgramChecking para cumplir con el objetivo de esta memoria dado que obtuvo 5 puntos más que la evaluación de Ocimatic. Esta herramienta cuenta con una aplicación web modular que permitirá implementar nuevas funcionalidades. Por otro lado, a pesar de que Ocimatic ya permite revisar código en C++, el módulo de corrección de código de EasyProgramChecking tiene una estructura con módulos claros que permitirá extender la herramienta para más lenguajes.

Finalmente, EasyProgramChecking fue diseñada para ser utilizada en el contexto de un curso, en cambio Ocimatic fue diseñada en un contexto competitivo y esto se vio reflejado en la reusabilidad de cada herramienta y la usabilidad que se encontró en la evaluación.

2.4. Resumen

En este capítulo se presentó un ejemplo de ejercicio de programación competitiva, de manera de dar mejor contexto al sistema que se desarrolló en la presente memoria. Luego, se hizo un análisis de bibliografía de aquellas publicaciones que han tratado sobre el problema de dar feedback automatizado a ejercicios de programación y las diferentes soluciones que se proponen para lograrlo. De estos estudios se tomaron algunos conceptos para ser incluidos en el sistema que se desarrolló en esta memoria.

Finalmente se hizo una evaluación de dos herramientas de revisión de código, Ocimatic y EasyProgramChecking, para decidir cual era más afín a la solución que se quería imple-

mentar en la presente memoria, concluyendo que EasyProgramChecking es más mantenible y extensible que Ocimatic, que los elementos que ya están implementados se acercan mejor a lo que se quiere desarrollar y que EasyProgramChecking tiene mejor usabilidad para las distintas usuarias que utilizarán el sistema.

Capítulo 3

Análisis y diseño

En esta sección se presenta el diseño del sistema propuesto en esta memoria y se analizará el sistema legado. En la Sección 3.1 se explicarán los requisitos del sistema a través de historias de usuario. Luego, en la Sección 3.2 se analizará la arquitectura lógica, la base de datos y la aplicación web del sistema legado. Para finalmente mostrar el diseño del sistema para la memoria en la Sección 3.3.

3.1. Historias de usuario

A continuación se describen las historias de usuario que explican el sistema a realizar en esta memoria.

En primer lugar se define la existencia de 3 tipos de usuario: estudiante, voluntaria y profesora. La diferencia entre una voluntaria y una profesora es que la profesora tendrá el poder de editar la información de los cursos a los que pertenece. Para el resto de las funcionalidades tendrán los mismos permisos, por lo que en este informe se hablará de docentes para referirse a ambas.

Cabe destacar que en secciones anteriores se hizo referencia a las docentes del curso como "tutoras" y no como voluntarias o profesoras, debido a que al momento de escribir este informe la estructura organizacional del curso de Niñas Pro se modificó. Por lo tanto, desde este momento una tutora será el equivalente a una profesora.

3.1.1. Estudiantes

A continuación, se listan las historias de usuario para una estudiante:

- **HE1:** Como estudiante, quiero acceder a un curso, para ver los problemas a resolver en una clase.

- **HE2:** Como estudiante, quiero ver el enunciado a un problema, para resolverlo.
- **HE3:** Como estudiante, quiero subir una solución a un problema, para recibir feedback sobre mi trabajo.
- **HE4:** Como estudiante, quiero ver qué casos de prueba no aprobé, para saber en qué me equivoqué.
- **HE5:** Como estudiante, quiero tener una sugerencia en un caso, para orientarme en cómo arreglar un error.
- **HE6:** Como estudiante, quiero saber qué problemas ya completé, para no resolverlos de nuevo.
- **HE7:** Como estudiante, quiero saber a qué clases no he asistido, para ponerme al día con los problemas que no he resuelto.

3.1.2. Profesoras

Las historias de usuario del rol de profesora corresponden a la creación y edición de cursos, clases y problemas. Como estas historias son comunes en una aplicación web, no se describirán con mayor detalle.

3.1.3. Docentes

A continuación se listan las historias de usuario para las voluntarias y profesoras (docentes):

- **HD1:** Como docente, quiero ver qué problemas han resuelto las estudiantes, para saber a quienes reforzar.
- **HD2:** Como docente, quiero ver los errores que se repiten para una caso de prueba, para entregar sugerencias de cómo arreglarlos.
- **HD3:** Como docente, quiero ver las sugerencias que ya se han agregado, para revisar los errores más comunes y poder guiar a las estudiantes.
- **HD4:** Como docente, quiero ver la asistencia general de un curso, para saber si hay estudiantes que han dejado de ir.
- **HD5:** Como docente, quiero pasar la asistencia de una clase, para tener información fidedigna de las estudiantes activas del curso.

3.2. Análisis del sistema a extender

Considerando los objetivos de esta memoria y las historias de usuario, se analizó la herramienta EasyProgramCheckin para buscar qué partes mantener y qué partes debían ser modificadas para llegar al producto final. Para facilitar este análisis y entender la lógica de

la herramienta, se utilizó el informe que realizó el equipo desarrollador de la aplicación para el curso de Ingeniería de software 2 de la FCFM.

Para analizar el sistema legado se instaló la herramienta y se crearon clases y tareas relacionadas con Niñas Pro para simular el uso de la aplicación por usuarias de la organización. Por otro lado, para analizar el código que se encarga de corregir las tareas y entregar feedback, se revisó todo el flujo de información que ocurre al subir una solución a corregir y todas las funciones que intervienen este flujo.

En esta sección se analizará la arquitectura lógica del sistema, el modelo de datos y la aplicación web que forma parte del sistema.

3.2.1. Arquitectura lógica

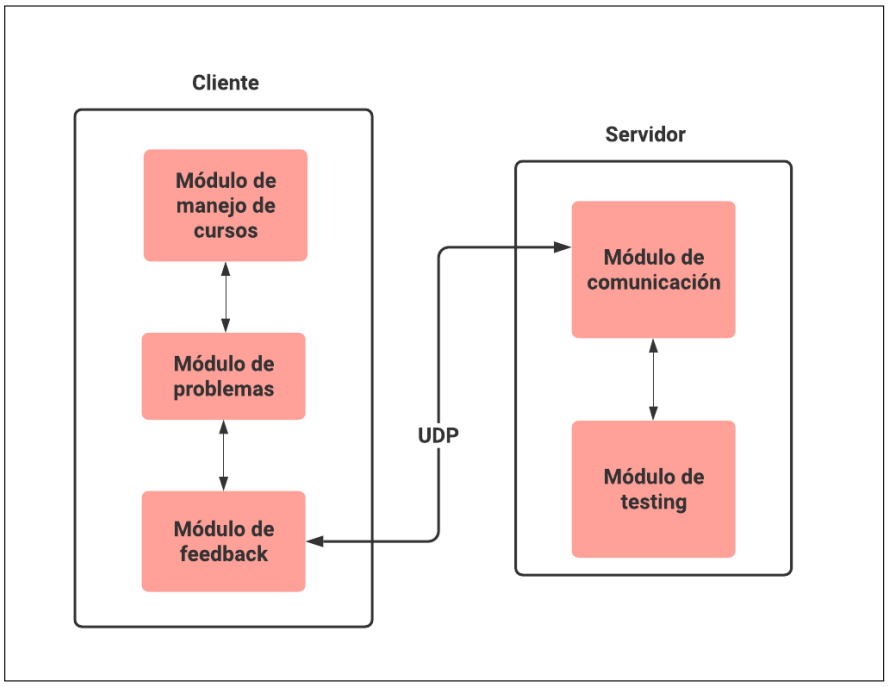


Figura 3.1: Diagrama de arquitectura lógica EasyProgramCheckin.

El sistema legado tiene una arquitectura cliente-servidor y se presenta en la Figura 3.1. El cliente será una aplicación web desarrollada en Django que está formada por los tres módulos que se ven en la figura. El servidor será el sistema encargado de recibir la solución a una tarea (módulo de comunicación), ejecutarla con sus casos de prueba (módulo de testing) y luego entregar feedback de la ejecución al cliente (módulo de comunicación). Como se ve en la imagen, el Módulo de Feedback del cliente se conecta por UDP a través de `sockets` de Python con el módulo de comunicación del servidor.

A continuación se explicará en detalle la arquitectura lógica del servidor y del cliente del sistema legado.

3.2.1.1. Servidor

El servidor recibirá el path de la solución a revisar, el path del archivo de casos de prueba del problema y un *lang* que será el lenguaje de programación que se utilizará. En este caso el *lang* puede ser "python2" o "python3".

Un archivo de casos de prueba podrá tener formato `.csv`, `.json` o `.yml`. Cada caso tendrá un *input*, *output*, *comment* y *test type*. Este último representa si el caso de prueba será visible para el estudiante (*public*) o no (*private*). En caso de ser *semiprivate*, el estudiante solo tendrá acceso al *input* del caso.

El servidor funcionará con un patrón "master/slave" que consiste en tener un thread principal llamado *Scheduler* que dará instrucciones a otros threads "esclavos" llamados *SchedulerSlave*. Cada *SchedulerSlave* ejecutará una de las soluciones a revisar y generará un resultado que será enviado por el thread principal al cliente.

El resultado que entrega el servidor será una lista, donde cada elemento será el feedback específico para cada caso de prueba. Cada elemento de esta lista tendrá la siguiente información:

- **passed**: si el caso de prueba pasó o no.
- **test_input:** input para el caso de prueba.
- **expected_output:** *output esperado* para el caso de prueba.
- **comment :** comentario asociado al caso de prueba.
- **type:** tipo asociado al caso de prueba que puede ser *public*, *private* o *semiprivate*.
- **err:** 0 si no hubo error, 1 si hubo un error de ejecución y 2 si hubo timeout al probar el caso.
- **actual_output:** *output* obtenido al ejecutar el código con el *input* asociado.

En la Figura 3.2 se encuentra un extracto del diagrama de secuencia del servidor. En este se explica el funcionamiento del *SchedulerSlave* desde su creación hasta que entrega el feedback de la solución al Servidor. Este es un ejemplo específico en que el *lang* es "python2".

En este diagrama hay dos actores importantes: *Python2Script* y *Tester*. *Python2Script* es una clase que hereda de la clase *Script* como se ve en el diagrama de clase de la Figura 3.3 además, por cada *lang* que tenga el sistema existirá una clase que hereda de *Script*, en este caso existe "Python2" y "Python3". Por otro lado, existe en el sistema un diccionario donde la llave será la *lang* y el valor será un método que permite crear una instancia de la clase con el path de la solución a revisar.

Cada una de las clases hija sobreescribe el método *get_process* de donde se obtiene el comando para ejecutar la solución en el lenguaje específico. En cuanto al método run de Script, recibirá un input y el tiempo que se esperará antes de dar timeout. Este método ejecutará la solución con el comando obtenido con *get_process* y el input recibido. Como se ve en el diagrama de la Figura 3.2. Los métodos mencionados anteriormente utilizan la

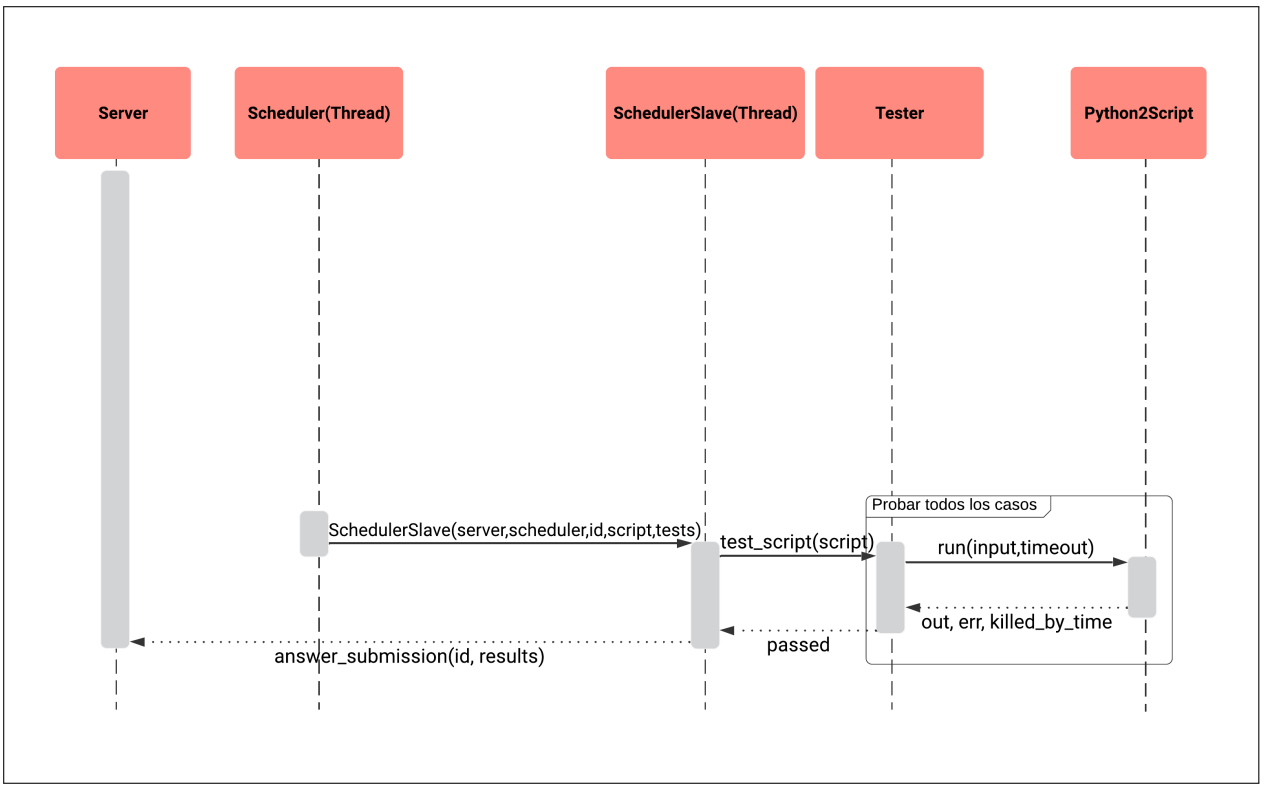


Figura 3.2: Diagrama de secuencia servidor.

librería subprocess [1] de Python para crear un proceso que ejecute los comandos.

Por otro lado, la clase *Tester* tendrá como atributo una lista con todos los casos de prueba del problema. Además, tendrá un método llamado *test_ script* que llamará al método *run* de *Script* con el *input* de cada caso de prueba. El método *run* ejecutará la solución según el *lang* elegido y entregará un feedback que será guardado en una lista por el *Tester*.

Finalmente el *Tester* entregará la lista con los feedbacks al *SchedulerSlave* y este enviará la lista al *Server* para ser procesada y enviada al cliente.

La ejecución de código se realiza de forma síncrona, por lo que cuando un usuario sube una solución a una tarea para corregir, deberá esperar a que se termine la ejecución antes de acceder a otra página.

3.2.1.2. Cliente

La aplicación web del cliente estará formada por dos módulos. Un módulo de manejo de cursos y problemas y un Módulo de Feedback.

Con el módulo de manejo de cursos se guardarán los cursos, problemas y sus respectivos archivos de casos de prueba.

[1] `https://docs.python.org/3/library/subprocess.html`

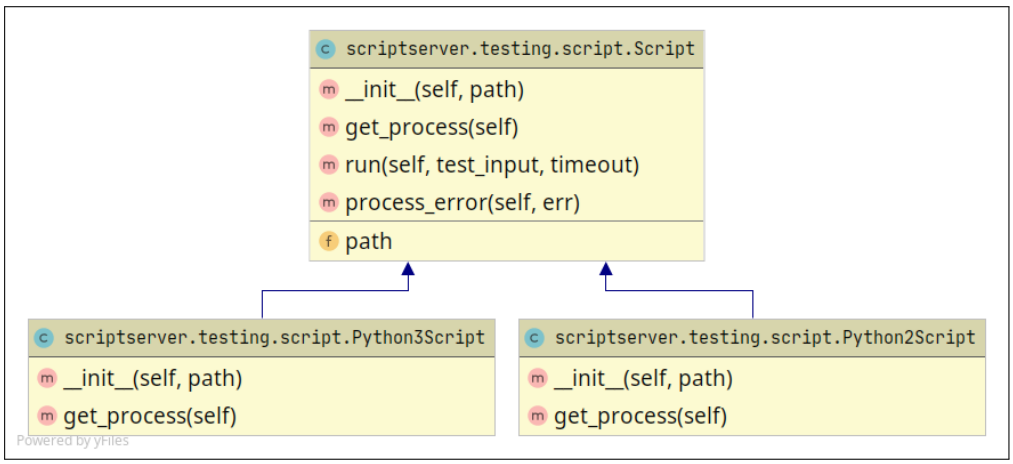


Figura 3.3: Diagrama de clase *Script*.

En el Módulo de Feedback se guardarán los códigos enviados por los estudiantes, se enviarán las soluciones al servidor para ser revisadas y se recibirá el feedback para luego mostrarlo a los usuarios.

3.2.1.3. Análisis de limitaciones de la arquitectura lógica

En el servidor se deberán hacer modificaciones ya que solo se pueden ejecutar códigos en Python y para el contexto de Niñas Pro se necesita ejecutar código en C++. Además, se modificará la estructura del archivo de casos de prueba, lo que tendrá un gran impacto en el servidor. Por el lado del cliente, se mantendrán los dos módulos existentes, pero se deberán modificar para cumplir con las necesidades de las historias de usuario.

3.2.2. Modelo de datos

En la Figura 3.4 se encuentra el modelo de datos relacionado con el manejo de cursos, tareas y usuarios. En la Figura 3.5 se encuentra el modelo de datos asociado al feedback, a continuación se listan y explican las principales entidades de este modelo de datos:

- **account:** Corresponde a la cuenta de un usuario y por lo mismo tiene una relación 1:1 con un *User* de Django. Esta entidad tiene un atributo llamado *can_create* que representa si una cuenta puede o no crear un curso.
- **course:** Representa un curso, esta entidad tiene el nombre, el código y quién lo creó.
- **account_teaching_courses:** Esta entidad relaciona un curso con una usuaria que tendrá el rol de profesora. Una usuaria puede ser profesora de varios cursos y un curso puede tener varias profesoras. La entidad account_assistant_courses es equivalente, sin embargo la usuaria tendrá el rol de estudiante.
- **assignment:** Esta entidad representa las tareas del curso y un curso puede tener varias tareas. Algunos atributos importantes son *title* que sería el título, *statement* que es el path al enunciado, *tests* que es el path al archivo de test y *period* que representa el

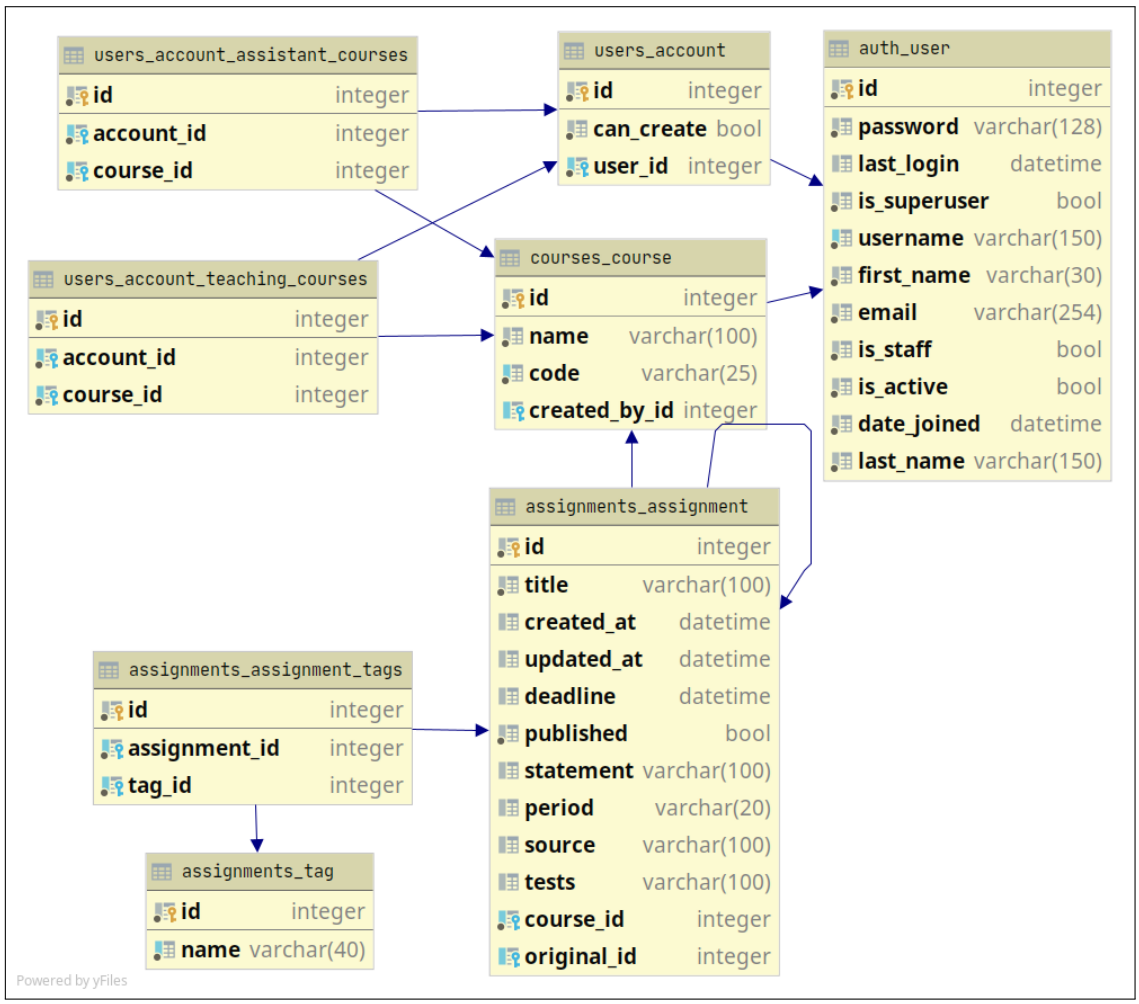


Figura 3.4: Modelo de datos legado, entidades relacionadas con el manejo de cursos y problemas.

semestre donde se creó la tarea; es importante destacar que period es un `varchar` y no tiene relación con otra entidad del modelo.

- **feedback:** Esta entidad guarda la información de cuando un usuario sube una solución para ser corregida y estará relacionado con un *assignment* y un *account*. Dado que, en el sistema se permite subir archivos masivamente para obtener varios feedback a la vez, se generará un feedback por cada archivo del grupo y cada feedback estará relacionado con la entidad *feedbackgroup*.

- **publictestfeedback:** Por cada caso de tipo *public* que se prueba con una solución se guardará su feedback en la entidad *publictestfeedback*. Esta guardará el *input* y *output* del caso, si el caso pasó o no, el *output* que generó la solución y si hubo un error; además de tener una llave foránea a la entidad *feedback*. Las entidades *privatetestfeedback* o *semiprivatetestfeedback* funcionan de la misma manera que esta, pero representan casos de tipo *private* o *semiprivate*.

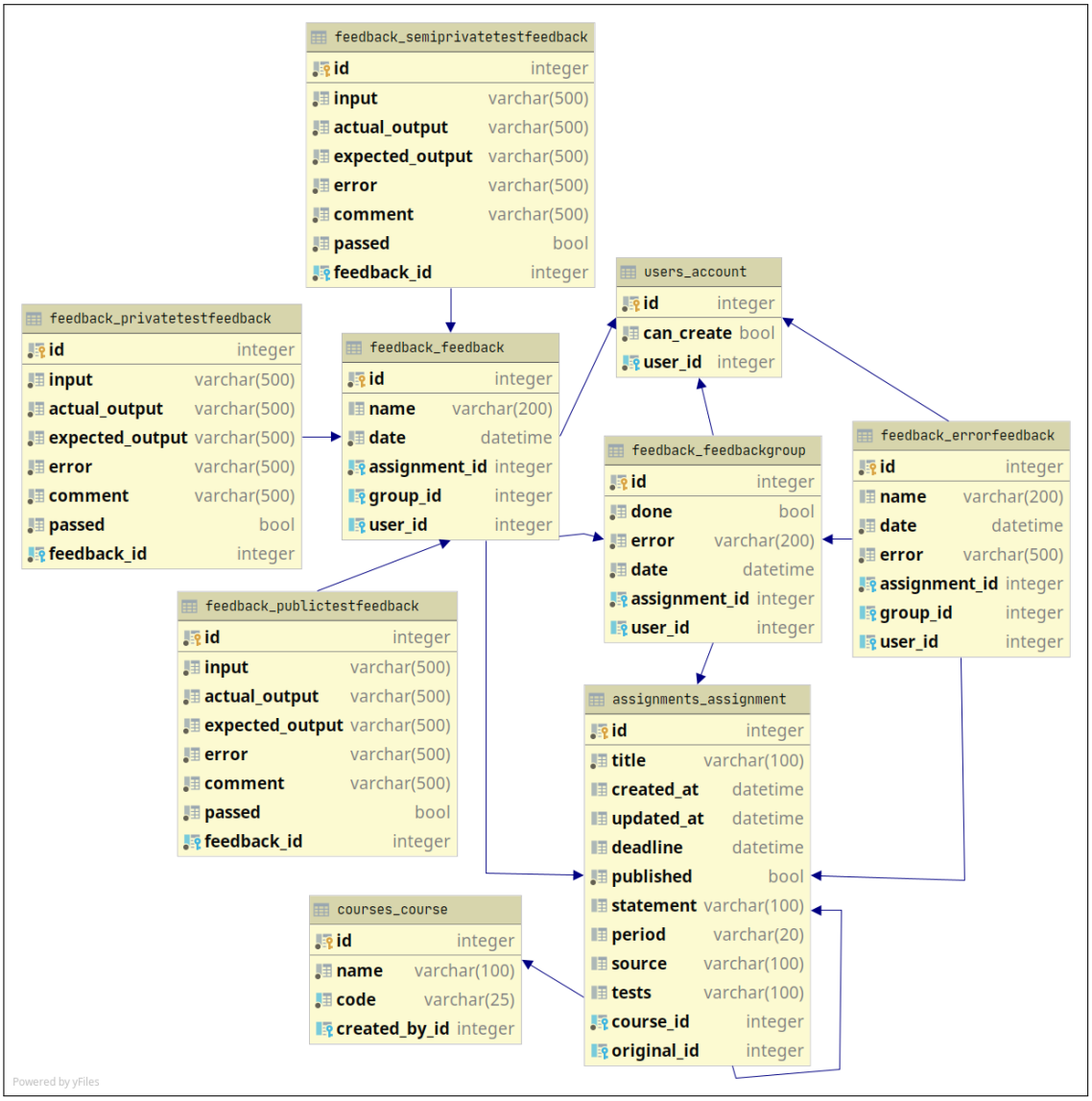


Figura 3.5: Modelo de datos legado, entidades relacionadas con el feedback.

3.2.2.1. Análisis de limitaciones

Dado que para esta memoria se necesitarán más roles que solo profesora y estudiante, se deberá modificar el modelo de user y account con el fin de distinguir los tipos de usuario. Por otro lado, en este modelo la única forma de diferenciar entre problemas de una clase u otra sería con el atributo *period*, lo cual es un problema ya que este atributo solo es un `varchar` y no es posible crear dos problemas con un mismo *period* si no es copiando el nombre de este. Debido a esto, habrá que agregar una entidad que represente las clases que se realicen en un curso.

Respecto al modelo de datos de feedback, el primer cambio que se necesitará hacer es quitar los tipos de casos, debido a que se querrán mostrar todos los casos de prueba a una estudiante cuando reciba feedback. Esto implica eliminar las tablas *publictestfeedback*, *privatetestfeedback* y *semiprivatetestfeedback*, y solo dejar una tabla de *testfeedback* con la

misma información. Por otro lado, según las historias de usuario de la Sección 3.1 no sería necesario subir soluciones masivamente, por lo que la entidad *feedbackgroup* no sería necesaria.

3.2.3. Aplicación web

Para cumplir los objetivos de esta memoria se deberán introducir grandes cambios al modelo de datos de la aplicación legada, lo que generará repercusiones en toda la aplicación web. Además, como las aplicaciones están en inglés y esto es una barrera para las alumnas que no manejan el idioma, se decidió crear una nueva. En esta se mantendrán los módulos que forman la aplicación legada, pero se aprovechará la oportunidad de partir desde cero con un proyecto de Django para crear una aplicación orientada a la estructura del curso anual de Niñas Pro.

Será importante tener como referencia la aplicación legada a la hora de crear problemas y casos de prueba, ya que al conectar la aplicación con el servidor se deberá disponer de toda la información que este necesita para procesar una solución.

3.3. Diseño del sistema

Para el nuevo diseño del sistema se mantendrá una arquitectura cliente -servidor. A continuación se mostrarán los cambios que se harán en el servidor para agregar C++ como lenguaje disponible y el nuevo diseño de la aplicación web del cliente.

3.3.1. Servidor

Dada la descripción que se hizo del funcionamiento del servidor en la Sección 3.2.1.1 se creará una nueva clase que hereda de la clase *Script* para poder compilar y ejecutar códigos en C++. Esta clase se llamará *CppScript* y deberá sobreescribir el método *get_process* para tener los comandos específicos para compilar y ejecutar un código en C++.

Suponiendo que se tiene un código escrito en C++ llamado *solucion.cpp*, los comandos para compilar y ejecutar este archivo serían los siguientes:

```
$ g++ -std=c++11 solucion.cpp -o solucion.o
$ ./solucion.o
```

Esto mismo deberá hacer la nueva clase *CppScript*.

Por otro lado, se modificará el formato de los casos de prueba para quitarles la clasificación por tipos.

3.3.2. Cliente

Como se explicó en la Sección 3.2 se diseñará una nueva aplicación web para la parte del cliente que contará con cuatro módulos: Módulo de Cursos, Módulo de Feedback, Módulo de Seguimiento y Módulo de Asistencia.

El diseño del sistema se hizo pensando en que las estudiantes solo participarán de un curso a la vez y que las tutoras participan en varios cursos.

Para el diseño de interfaces se crearon mockups en papel que luego fueron digitalizados. Por otro lado, el diseño del modelo de datos se realizó de forma iterativa, a medida que se desarrollaron los módulos.

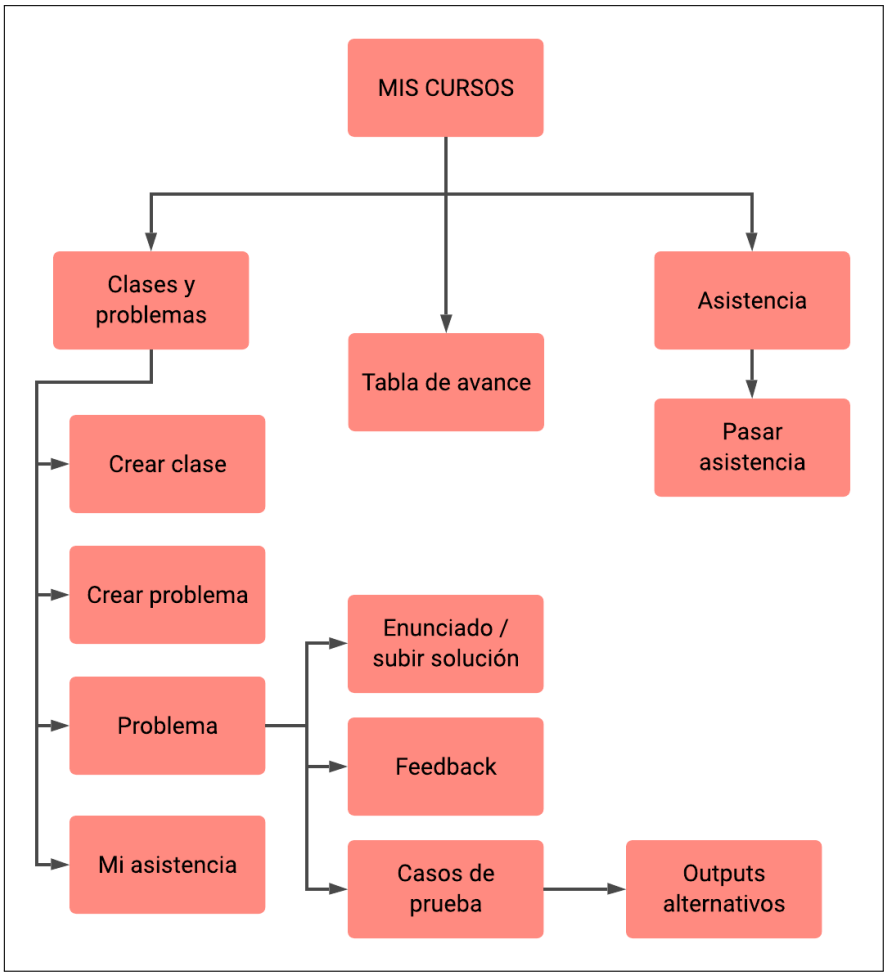


Figura 3.6: Diagrama de navegación del sistema.

En la Figura 3.6 se encuentra el diagrama de navegación del sistema a desarrollar en la memoria. Las vistas que se encuentran bajo "clases y problemas" serán parte del módulo de manejo de cursos. Las vistas que se desprenden del "problema" serán parte del Módulo de Feedback. La vista "tabla de avance" será parte del Módulo de Seguimiento y finalmente, las vistas "asistencia" y "pasar asistencia" serán parte del Módulo de Asistencia.

A continuación se procederá a explicar en detalle el diseño de cada módulo. Para cada

uno se presentarán los mockups de las interfaces y el modelo de datos.

3.3.2.1. Módulo de Cursos

El Módulo de Cursos será el encargado de la creación y visualización de cursos, clases y problemas. Además, se incluye en este módulo el manejo de usuarios. Es muy importante la existencia de este módulo, ya que el curso anual de Niñas Pro, cuenta con 4 sedes independientes, entonces hay que distinguir entre estudiantes y tutoras de diferentes cursos y los problemas y clases se realizan en cada uno.

Modelo de datos

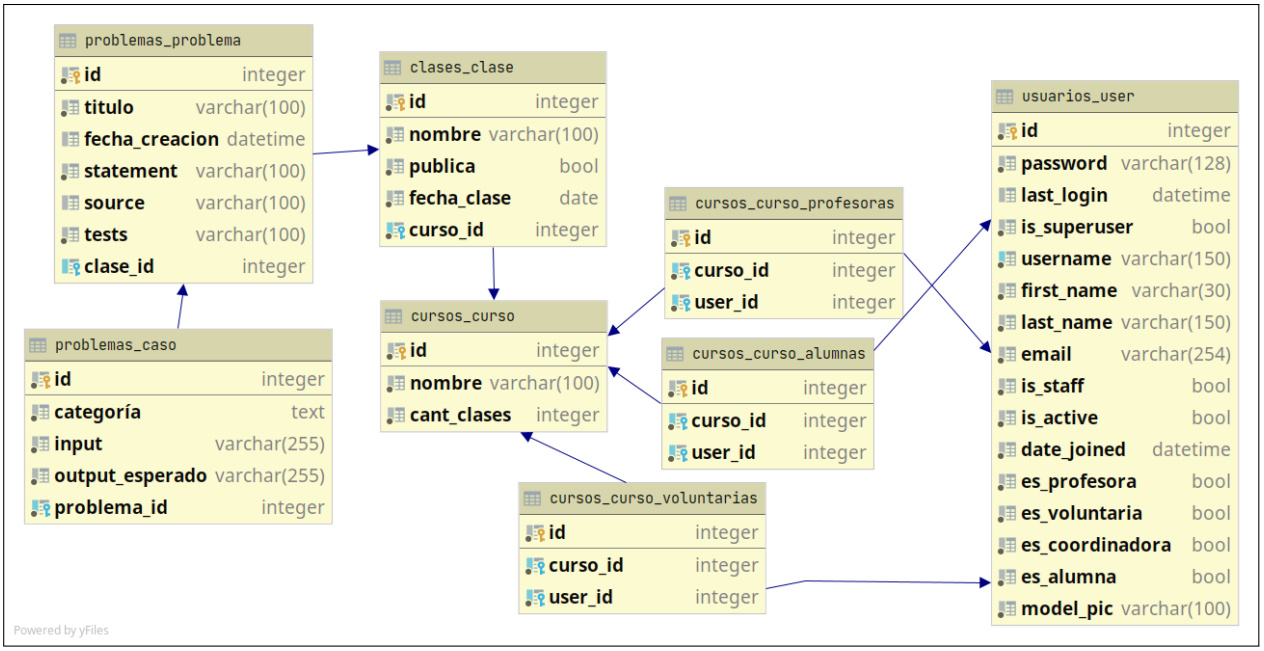


Figura 3.7: Modelo de datos para Módulo de Cursos.

En este módulo se realizaron varios cambios al modelo de datos,reestructurando la forma en que funcionaban los usuarios y se agregó una tabla de clases entre los cursos y los problemas. El resultado se encuentra en la Figura 3.7. Además, se tradujeron todos los nombres de las tablas a español. A continuación se detallan las entidades más importantes del nuevo modelo de datos:

- **user:** Se extendió el modelo *User* de Django para guardar nueva información sobre los usuarios. El gran cambio realizado en este modelo es que se guardará el tipo de usuario como un booleano. Por ejemplo, para registrar una profesora, el campo *es_profesora* será `True` y los campos *es_voluntaria* y *es_alumna* y *es_coordinadora* serán `False`. El campo *es_coordinadora* hace referencia a una usuaria de tipo coordinadora, que finalmente no se incluyó en esta memoria.
- **curso:** En esta entidad se añadió un campo llamado *cant_clases* para tener la cantidad aproximada de Clases que tendrá el Curso.

- **curso_profesora:** Al igual que en el modelo legado, esta tabla representa la relación entre una profesora y un Curso. Una profesora puede enseñar en varios Cursos y un Curso puede tener varias profesoras. Las tablas *curso_alumnas* y *curso_voluntarias* son equivalentes a esta.
- **clases:** Esta es una entidad que se añadió para distinguir entre las diferentes Clases de un Curso. La Clase tendrá un nombre, por ejemplo "Clase de variables", tendrá un booleano que indica si la clase estará pública o no para las estudiantes, la fecha de la Clase y el Curso al que está asociada.
- **problemas:** Esta entidad es el símil de la entidad *assignment* en el modelo legado. En este caso el *Problema* tendrá como llave foránea el *id* de la *Clase* a la que pertenece. En un inicio del diseño se quiso que los *Problemas* pudieran ser utilizados en varias *Clases*. Así, al crear un nuevo *Curso* se podrían reutilizar *Problemas* existentes. Más tarde se consideró que para este prototipo esta funcionalidad no sería necesaria y que basta con que los *Problemas* estén asociados a una sola *Clase*. Para trabajo futuro se debe evaluar si integrar esta funcionalidad.
- **caso:** Esta es una nueva entidad introducida al sistema. Al crear un *Problema* se procesa el archivo de casos de prueba para agregarlos a la base de datos. Un *Caso* tendrá una *categoría* asignada por la tutora, el *input* del caso de prueba, el *output* que se espera para ese *input* y el *id* del *Problema* al que pertenece. Gracias a la introducción de esta entidad se evita tener información repetida en el modelo de datos y se puede relacionar fácilmente el feedback con los casos.

Interfaces

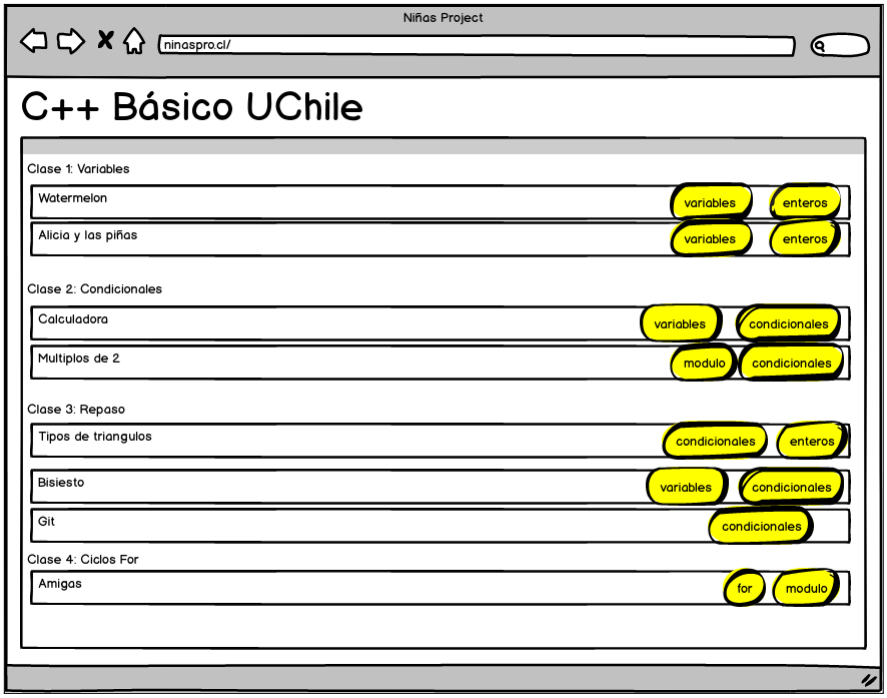


Figura 3.8: Mockup vista de inicio de un curso.

Dado que las estudiantes tendrán acceso a un único curso, cuando estas inicien sesión irán directamente a la vista inicial del curso. En el caso de las voluntarias y profesoras se iniciará

en una vista llamada *Mis cursos*, donde verán los cursos a los que pertenecen y podrán acceder al inicio de un curso, ver el seguimiento o ver la asistencia de este. En la Figura 3.8 se muestra el mockup de la vista de inicio de un curso, donde las estudiantes y docentes podrán ver todas las clases y los problemas de estas. Al diseñar el sistema se pensó en poner etiquetas sobre el tema que abarca cada problema, pero finalmente no se implementó. Es por esto que en el mockup de la vista de inicio de un curso se ven unas etiquetas amarillas.

En esta vista las profesoras tiene la opción de editar y agregar una clase o problemas, pero no se presentan estas funcionalidades en este informe dado que no son relevantes para el prototipo.

3.3.2.2. Módulo de Feedback

Para esta memoria se extenderá el Módulo de Feedback del sistema legado para complementar el feedback con casos de prueba. Dado que en la Sección 2.2 se vio que los errores que se cometen en problemas introductorios de programación suelen repetirse, es que en el nuevo sistema se recolectarán los *outputs* que genera una solución y que no concuerdan con el *output esperado*, para que con esta información las tutoras puedan identificar errores comunes que cometan las estudiantes y sugerirles cómo arreglar su código. Los *outputs recibidos* que no coinciden con el *output esperado* se llamarán *Output alternativo*. Cada uno de estos se guardará en conjunto con la frecuencia de aparición y el caso al que está asociado en la base de datos.

A continuación se explica cómo funcionará esto en el sistema:

Cuando una estudiante suba una solución y se reciba el feedback del servidor, el sistema buscará los casos de prueba que la solución no haya pasado. Para cada uno de estos casos se buscará si existe un *Output Alternativo* asociado al *output recibido*. En caso de que exista, se aumentará en uno la frecuencia de aparición. En caso contrario se creará un nuevo *Output Alternativo* con frecuencia 1.

Cuando una docente entre a ver los casos de prueba de un problema podrá ver los *Outputs Alternativos* que existen. La docente solo podrá ver los *Outputs Alternativos* que tengan una frecuencia mayor a 1, ya que así se sabrá que más de una estudiante obtuvo el mismo resultado. Para cada *Output Alternativo* la docente podrá agregar una sugerencia así, cuando otra estudiante genere este mismo, podrá ver un mensaje sugiriendo donde podría estar su error.

Modelo de datos

En la Figura 3.9 se encuentra el nuevo modelo de datos para el Módulo de Feedback, y a continuación se listan los mayores cambios hechos en el modelo.

- **feedback:** En esta entidad se añadió un campo donde se guardará el path a la solución enviada, así se podrán revisar los códigos de cada estudiante en el futuro.
- **testfeedback:** Al igual que en el sistema legado esta entidad representa el resultado de un caso de prueba específico para una solución. Como se mencionó en la Sección 3.2.2,

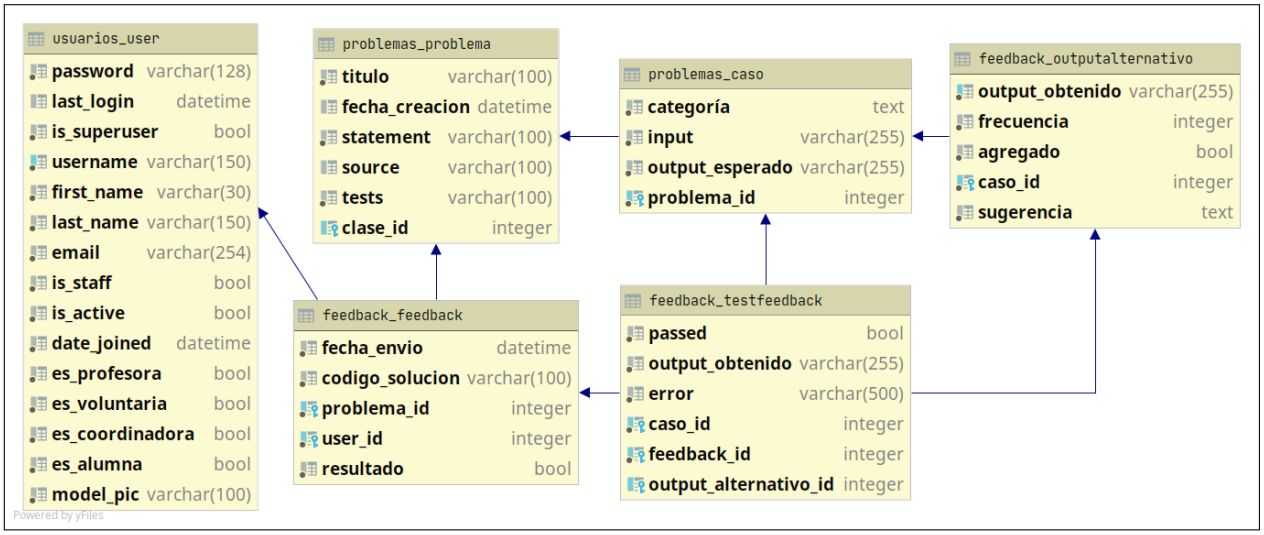


Figura 3.9: Modelo de datos para el Módulo de Feedback.

se quitaron los tipos de caso de prueba por lo que solo se mantiene una tabla de *testfeedback*. En el Módulo de Cursos se agregó una entidad caso para guardar cada caso de prueba, por lo tanto un *testfeedback* estará asociado a una instancia de caso.

- **outputalternativo:** Esta entidad guardará los *outputs* generados por las soluciones de las estudiantes que no coinciden con el *output esperado* de un caso. Los campos obligatorios de esta entidad serán *output obtenido*, *frecuencia* con que aparece el *Output Alternativo*, el *caso* al que está asociado y un booleano que representa si ya se agregó una sugerencia llamado *agregado*, que inicialmente será `False`. Cuando una tutora agregue una sugerencia de porqué se podría haber generado este *output* y cómo solucionarlo, esta se guardará en un campo *sugerencia* y el campo *agregado* pasará a ser `True`.

Interfaces

Las principales interfaces de este módulo serán la vista del feedback de una solución, la vista de los casos de prueba de un problema y la vista de los respectivos *outputs alternativos*. A todas estas vistas se tendrá acceso al seleccionar un problema en la vista de inicio de un curso.

Cuando una estudiante reciba feedback de una solución llegará a la vista de la Figura 3.10. Aquí se verán todos los casos de prueba del problema con el *input* del caso, el *output esperado* y el *output obtenido*. Cuando el *output esperado* y el *output obtenido* sean iguales, el *output obtenido* se pintará de verde ya que ese caso aprobó. En caso contrario se pintará el *output obtenido* de rojo. Por último, cuando la solución no aprueba un caso y existe un *output alternativo* con sugerencia para ese *output obtenido*, se mostrará la sugerencia en pantalla.

La vista de casos de prueba mostrará una tabla que tendrá todos los casos de un problema con su *input*, *output esperado* y una notificación si existen *outputs alternativos* que aún no tienen sugerencia. A esta vista tendrán acceso las voluntarias y profesoras.

Al hacer click sobre una notificación por *output alternativo* se accederá a los *Outputs Alternativos* para este caso como se ve en la Figura 3.11.

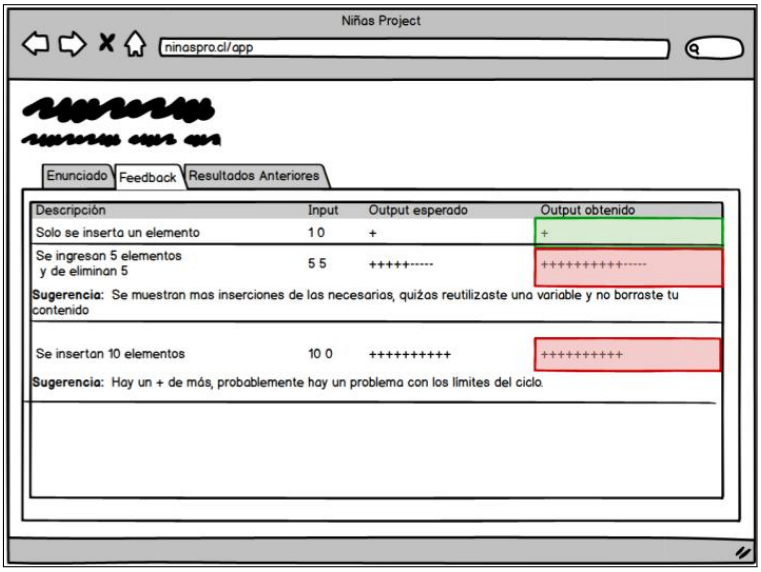


Figura 3.10: Mockup de la vista al recibir feedback.

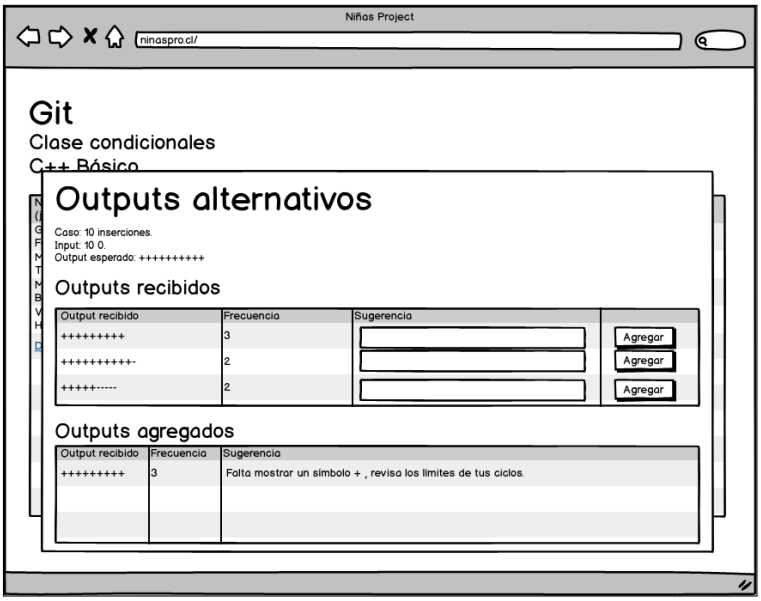


Figura 3.11: Mockup vista de *Outputs Alternativos* para un caso.

En la vista de los *outputs alternativos* se verá información sobre el caso de prueba y dos tablas, donde la primera tendrá los *outputs recibidos*, es decir, todos los *outputs* que han generado las soluciones de las estudiantes que no son el *output esperado* para ese caso y que aún no tienen sugerencia; donde las tutoras podrán agregar una sugerencia para las próximas estudiantes que obtengan el mismo caso. En cuanto a la segunda tabla, esta tendrá la información de los *outputs* que ya tienen sugerencias.

3.3.2.3. Módulo de Seguimiento

Este módulo será el encargado de mostrar los avances de las estudiantes respecto a los problemas que han resuelto y no posee un modelo de datos, ya que solo toma los datos de otros módulos y los visualiza.

Para el trabajo de esta memoria solo se realizó una interfaz en este módulo, pero en futuras versiones se espera generar más visualizaciones que permitan hacer mejor seguimiento del curso y sus estudiantes incorporando variables como la asistencia o el tiempo de interacción que tienen las estudiantes con la plataforma.

Interfaces

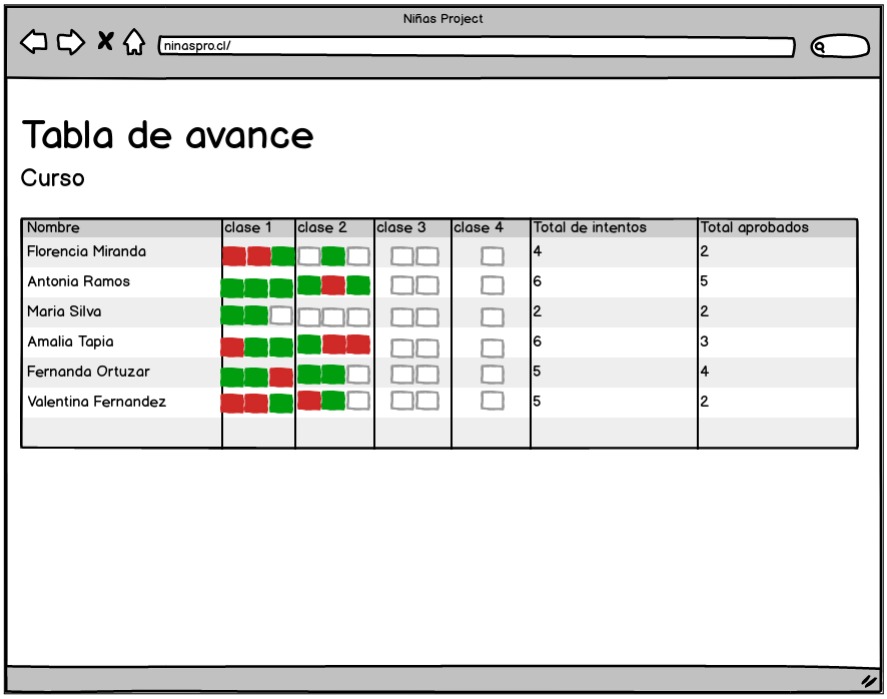


Figura 3.12: Mockup vista de seguimiento.

El mockup de la interfaz para este módulo se encuentra en la Figura 3.12. En este mockup se ven los nombres de todas las estudiantes de un curso y todas sus clases. Los cuadrados rojos, verdes y blancos representan los problemas de cada clase. Si el cuadrado está rojo es porque la estudiante intentó resolver el problema pero no pasó todos los casos; si el cuadrado está verde es porque intentó resolver el problema y pasó todos los casos. Finalmente si un cuadrado no tiene color es porque la estudiante aún no ha subido una solución para ese problema.

Al final de cada fila se puede ver el total de intentos que ha tenido la estudiante a lo largo del curso y el total de problemas aprobados.

Con esta interfaz se espera que las tutoras puedan identificar a las estudiantes que necesitan ayuda, cuando por ejemplo, tengan muchos problemas en rojo o no hayan subido nada, y también identificar a estudiantes que van más rápido que el resto para desafiarlas más.

Por otro lado, con esta visualización se podría identificar qué problemas tuvieron mejores o peores resultados, para considerarlo en cursos futuros, o para revisarlos en alguna clase.

3.3.2.4. Módulo de Asistencia

Este módulo será el encargado del manejo de la asistencia de las participantes. Permitirá que las tutoras pasen asistencia el día de la clase y que puedan ver la asistencia general de sus cursos.

El diseño e implementación de este módulo se realizó en conjunto con tres estudiantes de Ingeniería en Computación que realizaron su práctica con Niñas Pro. Las practicantes se encargaron de realizar el diseño de interfaces, la implementación del sistema y luego la validación de usabilidad de las interfaces, con la supervisión de la memorista en todo momento. Al finalizar el trabajo de la práctica, se implementaron las funcionalidades que quedaron pendientes.

Modelo de datos

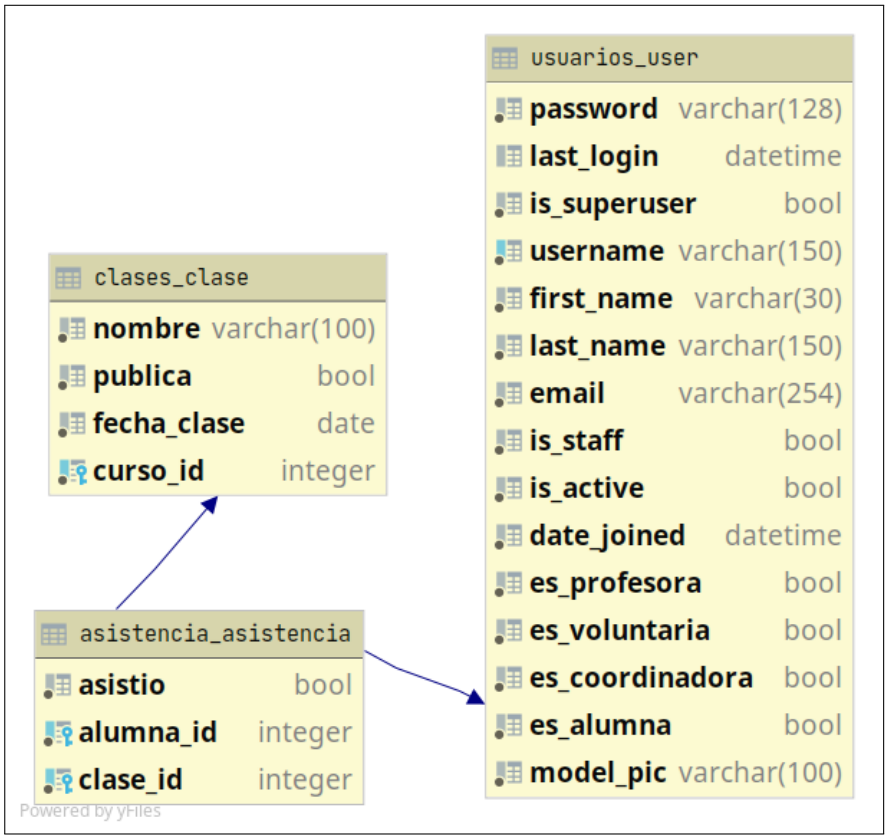


Figura 3.13: Modelo de datos Módulo de Asistencia.

En la Figura 3.13 se encuentra el modelo de datos para el Módulo de Asistencia, el cual solo incorpora una nueva tabla llamada *asistencia*. Una entrada de esta tabla representará la asistencia de una estudiante para una clase.

Interfaces

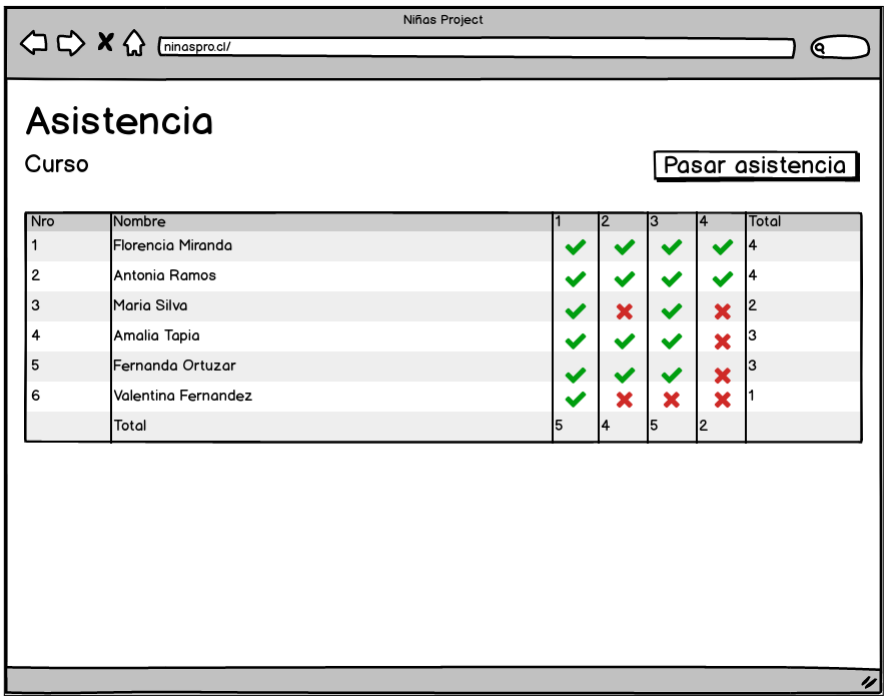


Figura 3.14: Mockup de vista de asistencia general.

Este módulo cuenta con una vista que muestra la asistencia general de un curso, una vista para pasar la asistencia y una vista para que cuando una estudiante vea el inicio de un curso pueda ver su asistencia.

En la Figura 3.14 se encuentra el mockup para la vista de asistencia general. Esta vista tendrá una tabla donde se verán todas las estudiantes y su asistencia a cada clase del curso. Con un tick verde se mostrará a las estudiantes que asistieron y con una cruz roja se marcará a las que no. Tanto para cada clase como para cada estudiante se mostrará al final de la tabla el total de asistencias.

Arriba a la derecha se encuentra el botón para pasar asistencia. Este botón llevará a otra vista que tendrá la lista de las estudiantes donde se podrá marcar quienes asisten. No se mostrará el mockup de esta vista ya que no aporta información relevante.

3.4. Resumen

Al inicio de este capítulo se describieron las historias de usuario para el sistema a implementar en esta memoria. Gracias a esto se hizo un análisis del sistema legado para comprender en profundidad su funcionamiento y para ver qué partes del sistema se mantendrán, cuáles se sacarán y cuáles se extenderán para cumplir con las historias de usuario. De este análisis se concluyó que se mantendrá la arquitectura cliente-servidor del sistema legado pero se creará

una nueva aplicación web en el cliente y se actualizará el modelo de datos, además de agregar C++ como lenguaje disponible en el servidor.

Luego, se presentó cómo se extenderá el sistema de revisión de código para soportar C++ y se presentó el diseño de la nueva aplicación web, con sus módulos, modelo de datos y mockups de las principales interfaces.

En los capítulos siguientes se discuten las decisiones tomadas en la implementación del sistema, los resultados obtenidos y la validación del sistema con usuarias reales.

Capítulo 4

Implementación

Considerando el diseño propuesto en el Capítulo 3, en este capítulo se describirá la implementación del sistema diseñado. Este capítulo se separará en cliente y servidor. Para el servidor se describe cómo se añadió el Script para que el sistema revise codigos en C++, y para el cliente describirán los aspectos generales de la aplicación web, la implementación de los 4 módulos y los problemas que fueron surgiendo en el proceso.

4.1. Servidor

La primera modificación que se hizo en el servidor fue crear una clase llamada *CppScript* donde estará el comando para compilar y ejecutar un código en C++. En la Figura 4.1 se encuentra el nuevo diagrama de clases. Para ejecutar un código en C++ se requiere compilar

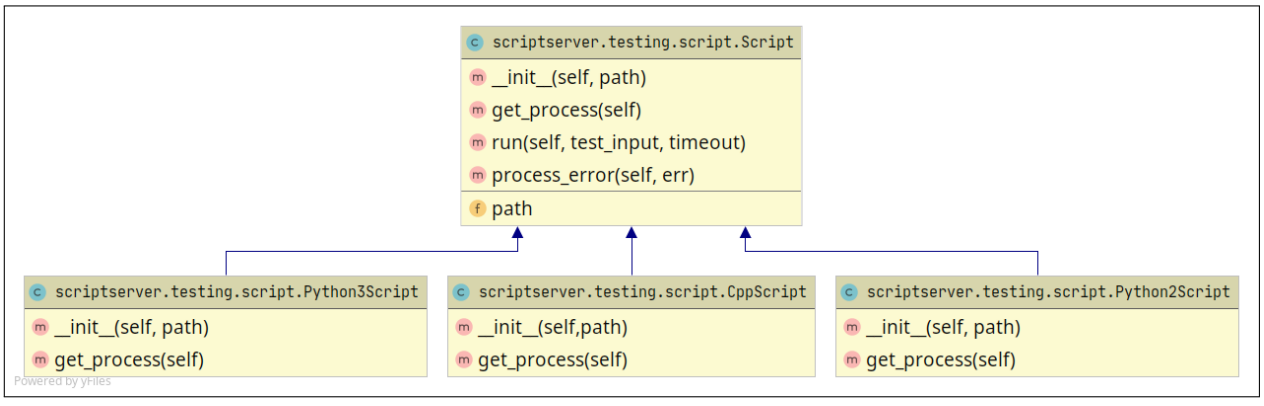


Figura 4.1: Diagrama de clases *Script*.

el archivo antes de ejecutarlo, por lo tanto, se utilizó el método `run` [1] de la librería `subprocess` de Python para que el método *get_process* espere a que la compilación esté completa antes de retornar el comando para ejecutar.

[1] `https://docs.python.org/3/library/subprocess.html#subprocess.run`

Luego de crear la clase *CppScript* y que esta permitiera ejecutar códigos en C++, se trabajó con los errores. Puede ser que una solución enviada por una estudiante no compile, puede ser que un caso de prueba genere un timeout o que aparezca un error de ejecución. En la Sección 4.2 se explica cómo las estudiantes recibirán estos errores y como se les entregará feedback en estos casos. A continuación se explica cómo se hizo para abordar estos tres tipos de error en el servidor:

Error de compilación: Cuando un código no compila no se crea el archivo `.o` por lo tanto cuando el *Script* intenta ejecutar el código no podrá porque no encontrará el archivo ejecutable. Esto provoca una excepción de tipo `OSError` que significa que no se encuentra un archivo. Esta excepción será atrapada en el *SchedulerSlave* y hará que se termine la revisión de ese código y se envíe un mensaje de error al cliente.

Error de timeout: Cuando el *Tester* invoca el método `run` de la clase *Script* le entrega el *input* del caso que se está probando y un valor *timeout*. Este valor indica el tiempo máximo que puede durar la ejecución del código y si se supera este tiempo, se entregará un error de timeout. El trabajo con timeout ya venía implementado en el sistema legado, por lo que no se realizó ningún cambio.

Error de ejecución: Al igual que el error de timeout, el error de ejecución dependerá del caso que se está probando. Para atrapar este error se toma el valor que retorna el proceso que ejecuta el código, y si este valor es negativo, entonces hubo un error. En esta memoria no se tendrá información sobre el error específico que se produjo, solo se sabrá que hubo un error con ese caso.

Otro cambio que tuvo efectos sobre el servidor fue la modificación del formato de los casos de prueba. En el sistema legado los casos de prueba incluían una descripción y un tipo, que podía ser *public*, *private* o *semiprivate*. En el nuevo sistema ya no habrá descripción ni tipo del caso, sino que una categoría. Las *categorías* serán definidas por las tutoras al crear los casos de prueba. Se cambió descripción por categoría ya que al implementar la vista de feedback de un problema se vio la necesidad de agrupar los casos para mejorar la usabilidad del sistema.

Este cambio afecta al servidor ya que en este se procesa el archivo de casos de prueba, para crear el objeto *Tester*. Por otro lado, al generar un feedback se incluyen todos los elementos de un caso de prueba entonces hubo que modificar esto.

4.2. Cliente

A continuación se hablará sobre la implementación de la aplicación web en el cliente. En la Subsección 4.2.1 se mencionan aspectos generales de la aplicación web y en las siguientes Subsecciones se explicará cómo se implementó cada módulo diseñado.

4.2.1. Aspectos generales

Al igual que en la aplicación legada, la aplicación web del cliente fue desarrollada con el framework Django. Para la base de datos se utilizó SQLite, que es la base de datos por defecto en Django. Para darle estilo a las interfaces de la aplicación web se utilizó el framework Materialize [2] y se complementó con hojas de estilo CSS. Para hacer hojas de estilo mantenibles y extensibles, se utilizó una extensión de CSS llamada SASS [3]. SASS permite hacer hojas de estilo utilizando variables, herencia, entre otras funcionalidades. El mayor uso que se le dio a esto fue para la definición de los colores del sistema. Gracias a esto todo el sistema utiliza una base de 7 colores que son definidos al inicio de la hoja de estilo. Así, si se decide hacer un cambio en el estilo general del sistema, bastará con modificar el valor de las variables de color, para que los cambios se vean en todo el sistema.

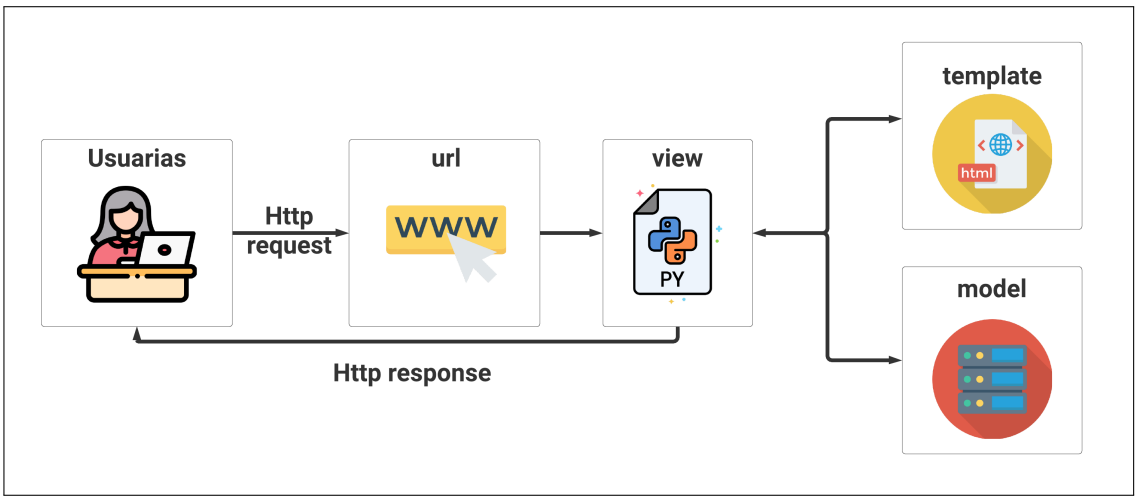


Figura 4.2: Diagrama del funcionamiento de Django. Iconos obtenidos desde `www.flaticon.com`

Django es un framework que sigue el patrón Modelo Vista Controlador (MVC). Este es un patrón de aplicaciones web que separa la base de datos (modelo) de las interfaces (vista) y de la lógica de la aplicación (controlador). A continuación se explica el diagrama de la Figura 4.2 donde se muestra cómo interactúan las diferentes partes de una aplicación en Django:

- Cuando una usuaria entra a la aplicación desde un navegador, la aplicación buscará en una lista de *urls*, que se encuentran en el archivo `urls.py`, si la *url* ingresada tiene un patrón válido para la aplicación.
- Si la *url* es válida entonces estará asociada a una *view*. Una *view* será una clase de Python o un método, que estará encargada de la lógica de la aplicación.
- La *view* podrá acceder a los elementos de la base de datos (llamados *model*) para ver, crear, editar o eliminar información.
- La *view* podrá cargar una interfaz (llamadas *template*) junto con elementos de la base de datos para mostrar información a la usuaria e interactuar con ella. Un *template* será un archivo `html` donde se define una interfaz.

[2] `https://materializecss.com/`
[3] `https://sass-lang.com/`

- Una *view* retornará una response Http a la usuaria, que en general contendrá un *template* y un diccionario con información que se mostrará.

Es importante tener esta estructura en consideración para entender el proceso que se siguió para implementar cada módulo.

4.2.2. Módulo de cursos

En esta subsección se explica cómo se implementó la interfaz de inicio de un curso y la interfaz de inicio de un problema, y todas las funcionalidades que estas tienen; además se mostrarán algunos problemas que surgieron al implementar este módulo.

La implementación de la vista de inicio de un curso se realizó de forma incremental hasta llegar a la vista de la Figura 4.3. Inicialmente se creó una interfaz que mostraba únicamente el nombre del curso, luego, se mostraron sus clases junto con con la funcionalidad para su creación y edición; y finalmente se agregaron los problemas junto con con la opción para agregar uno nuevo.

Figura 4.3: Vista inicio curso.

A continuación se listan con mayor detalle los pasos seguidos para implementar el primer incremento de la interfaz de inicio de un curso:

- **Crear los *models* *User* y *Curso*** para tener estudiantes, profesoras y voluntarias asociadas al curso.
- **Crear el patrón de *url* para acceder a un curso**. Esta *url* tendrá la siguiente forma: <int:curso_id>/curso/. En el caso que el *id* de un curso sea "1", la *url* para acceder a ese curso será *1/curso/*.

- **Crear la *view* asociada a la *url***. Esta *view* llamada "CursoView" revisa el tipo de usuaria que está conectada para saber qué información se mostrará, y comprueba que esta sea parte del curso donde quiere acceder. Además, la *view* toma la información del curso desde la base de datos para que sea accesible en el *template*.
- **Crear el *template* para la vista de inicio de un curso**. Dado que solo existen los *models User* y *Curso*, se crea un *template* que mostrará solamente el nombre del curso.

Con esta interfaz lista se procedió a crear el *model* para las clases y mostrarlas en la interfaz de inicio de un curso. A continuación se listan los pasos seguidos para implementar esto:

- **Crear el *model Clase*** para tener clases asociadas a un *Curso*.
- **Obtener las clases de un *Curso* en la *view* "CursoView"**. En este incremento, se tomará de la base de datos todas las *Clases* asociadas al *Curso* para que sean accesibles en el *template*.
- **Agregar clases al *template* de la vista de inicio de un curso**. Ya que se tienen todas las clases del curso a mostrar, se utilizó un elemento de Materialize llamado "Collection", que permite mostrar una lista de elementos.

Para el desarrollo de las otras interfaces de esta memoria se siguió una metodología parecida a la que recién se presentó, por lo tanto, no se explicará en detalle la implementación de otras vistas, sino que se mostrarán los resultados.

En esta vista se podrá agregar una clase con el botón de arriba a la derecha, editar una clase con el lápiz de la clase respectiva y agregar un problema con el botón con un símbolo +. No se mostrará más información sobre estas funcionalidades ya que no son únicas de este sistema.

La interfaz de la Figura 4.3 muestra el inicio de un curso para una profesora. En el caso de una estudiante o voluntaria no se podrá agregar clases, editarlas, ni agregar problemas.

Otro elemento importante de esta interfaz es que en cada problema las usuarias verán si lo completaron o lo tienen incompleto. Esto quiere decir que si una usuaria sube una solución al problema, y aprueba todos los casos tendrá un mensaje "completo", y si no, tendrá un mensaje "incompleto". En caso de no haberlo intentado, no aparecerá nada.

Desde la vista de inicio de curso se puede acceder a los *Problemas* de cada *Clase* haciendo `click` sobre el título del *Problema*. Con esto se llega a la vista del enunciado del *Problema*. El mockup de esta vista tenía la opción de subir la solución en una ventana aparte a la del enunciado, pero se decidió dejar ambas funcionalidades juntas para disminuir la cantidad de `clicks` a realizar para subir una solución.

El código del *model Problema* es casi el mismo que el del *model Assignment* de la aplicación legada. En este, cuando se crea un nuevo *Problema* se verifica que el archivo de casos de prueba sea válido. Es decir, que todos los casos incluyan todos los elementos esperados, y que la estructura del archivo sea válida. Si el archivo falla, entonces este se elimina y el problema queda sin archivo de casos de prueba. Esto fue un problema cuando se añadió la creación de los casos de prueba en el *model*, porque el sistema no notifica cuando un archivo de casos de

prueba falla, entonces no se sabía porque no se creaban los casos asociados al problema. Esto no se arregló para esta memoria ya que se trabajó siempre con el mismo grupo de problemas, y sus casos de prueba que ya estaban validados. Para trabajo futuro habrá que validar los casos de prueba antes de crear un problema, y notificar a la profesora en caso de fallo para que arregle sus casos antes de subir el problema.

4.2.3. Módulo de feedback

Al recibir una solución a un problema, el sistema la enviará al servidor para generar un feedback. La revisión de una solución puede ser exitosa o no, y según eso será el procesamiento que se hará del resultado en el cliente. Esto se distingue ya que la respuesta del servidor trae un mensaje "success" o "error". En caso de tener un error, en vez de recibir una lista de

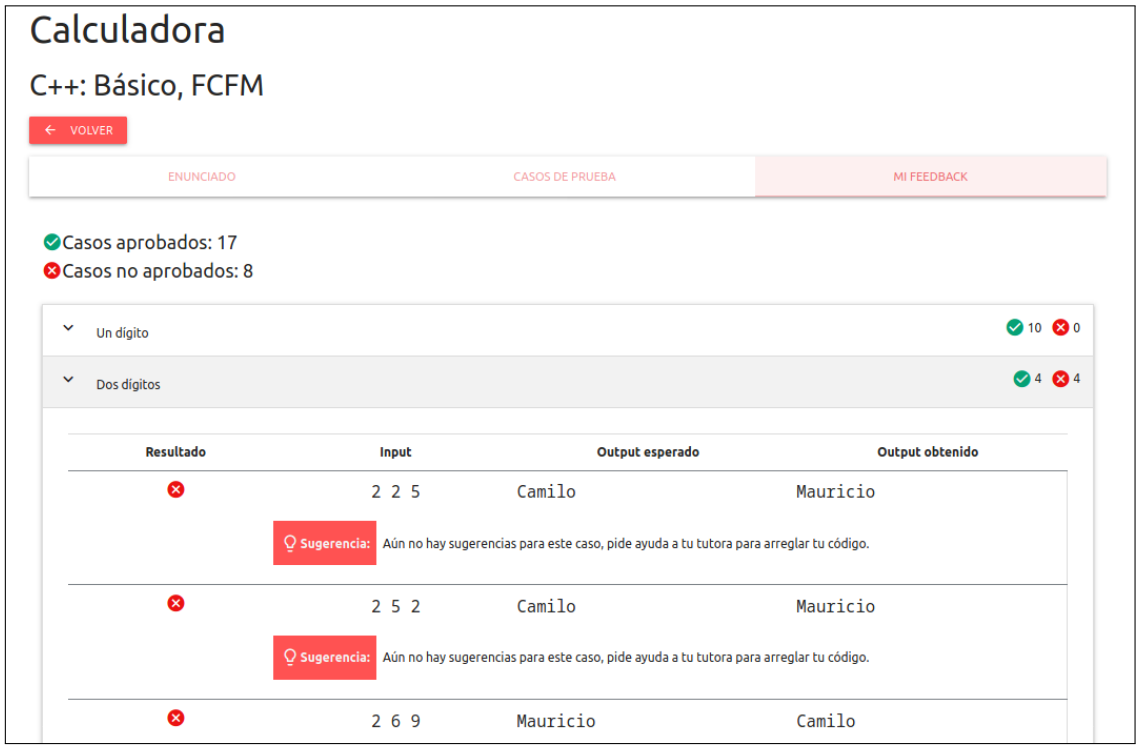


Figura 4.4: Vista del feedback a una solución.

feedback se recibirá un mensaje para mostrar a la usuaria. Para un error de compilación el mensaje es el siguiente: "Ocurrió un error al compilar tu archivo, comprueba con tu tutora que tu código compile y vuelve a intentarlo". El mensaje invita a la estudiante a solicitar ayuda ya que por ahora el sistema no puede dar feedback a un código que no compila.

Si la revisión es exitosa el sistema recibe una lista con el feedback para cada caso de prueba. Con esto, se crea en la base de datos un *Feedback*, con la *usuaria*, el *Problema* y el path al código de la solución; y luego por cada caso de prueba, se crea un *TestFeedback* asociado a este nuevo *Feedback*. Mientras se generan los *TestFeedback* se buscan los casos en que el *output* no es el esperado, para añadirlo a la tabla de *OutputAlternativo*. Con todos

estos pasos ya se tiene la información necesaria para visualizar el feedback y mostrar a las docentes los casos que tienen *Outputs Alternativos*.

En la Figura 4.4 se muestra la vista del feedback a una solución. En esta vista se ve la cantidad de casos aprobados y no aprobados para tener una noción general del resultado. Como se mencionó anteriormente se agrupa el feedback por *categoría*, ya que ver una lista con todos los casos y su resultado juntaba demasiada información. En la interfaz las *categorías* se pueden comprimir, por lo que cuando en una categoría todos los casos aprueban, esta estará comprimida y no se verán sus casos.

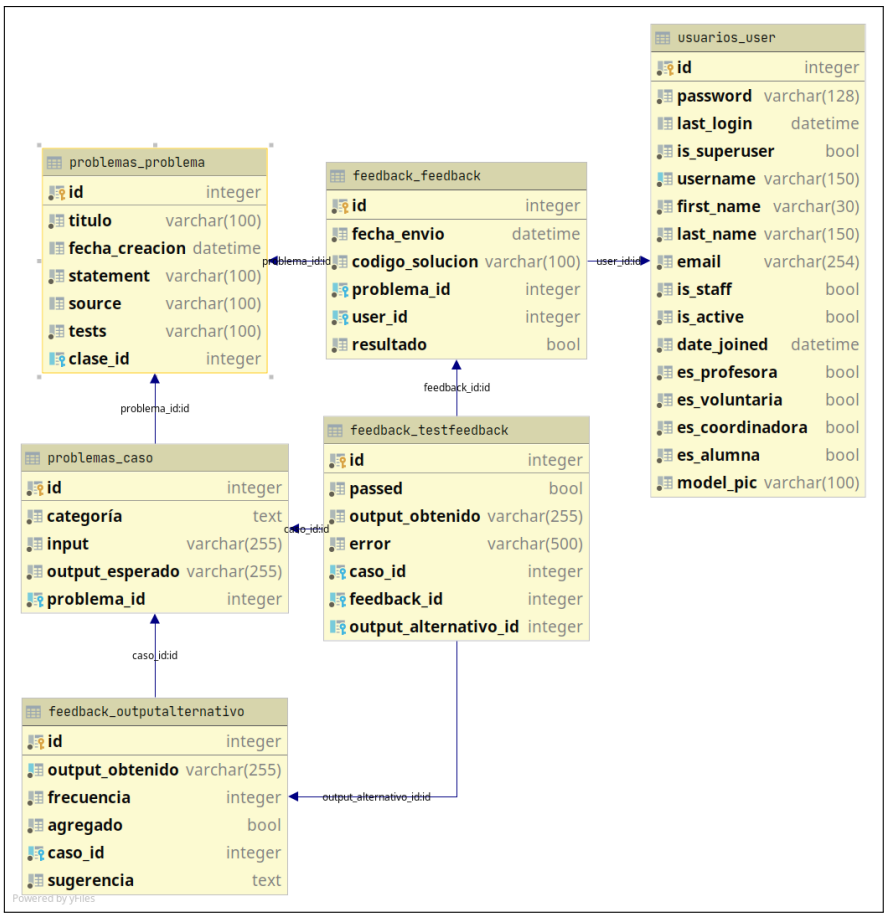


Figura 4.5: Modelo de datos final para módulo de feedback.

Un elemento importante en la vista de la Figura 4.4 es la visualización de la *sugerencia* cuando un caso no aprueba. La palabra "Sugerencia" es casi el único elemento de la interfaz que tiene color, ya que es importante que toda la atención esté puesta en esta información, al igual que en el resultado. Mostrar la información de la *sugerencia* no fue trivial ya que en el modelo de datos no existía una relación directa entre un *TestFeedback* (que tiene el resultado para un caso) y un *OutputAlternativo* asociado al *output recibido* (que puede tener una *sugerencia*). Para solucionar esto se agregó una relación entre la tabla *TestFeedback* y *OutputAlternativo* dejando el modelo de datos como en la Figura 4.5. Gracias a esto, cada vez que se crea un *TestFeedback* para un caso que no aprueba, se le asociará un *OutputAlternativo* y cuando se muestre la sugerencia, esta se obtendrá directamente gracias a la llave foránea.

Como se mencionó en la Sección 4.1 un caso de prueba puede dar un error de ejecución o de timeout. Esta información será entregada a la usuaria en la vista de feedback en vez de una sugerencia. En las Figuras 4.6 y 4.7 se ve cómo se entregarán estos errores. La importancia de mostrar esto es que una tutora tendrá una idea del error que tiene la estudiante y podrá ayudarle de forma más eficiente.

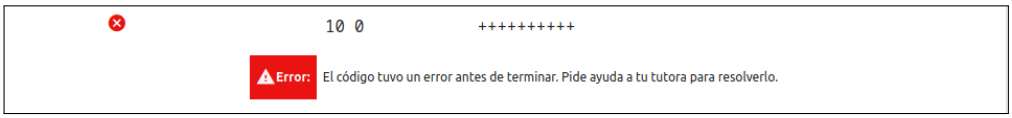


Figura 4.6: Vista de un error de ejecución.

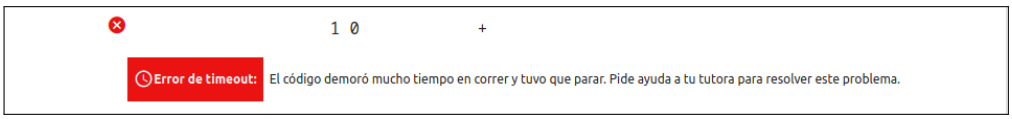


Figura 4.7: Vista de un error de timeout.

Otra vista importante de este módulo es la de casos de prueba y outputs alternativos que se encuentra en la Figura 4.8. En esta se ven los casos separados por *categoría* y por cada caso se sabrá cuántos *Outputs Alternativos* sin *sugerencia* hay. Al hacer `click` sobre el botón "Ver" de un caso se accede a la vista de *Outputs Alternativos* de la Figura 4.9. La carga del modal de *Outputs Alternativos* trajo dificultades ya que se quería cargar la información de este sólo cuando se abriera el modal y no durante la carga de la página. Esta decisión se debió a que si pasa tiempo entre que la tutora abre la página y abre el modal, las estudiante podrían haber subido soluciones y podrían no aparecer todos los *Outputs Alternativos*. Finalmente se utilizó Javascript para cargar el modal sin actualizar la página y obtener la información de los *Outputs Alternativos* más recientes.

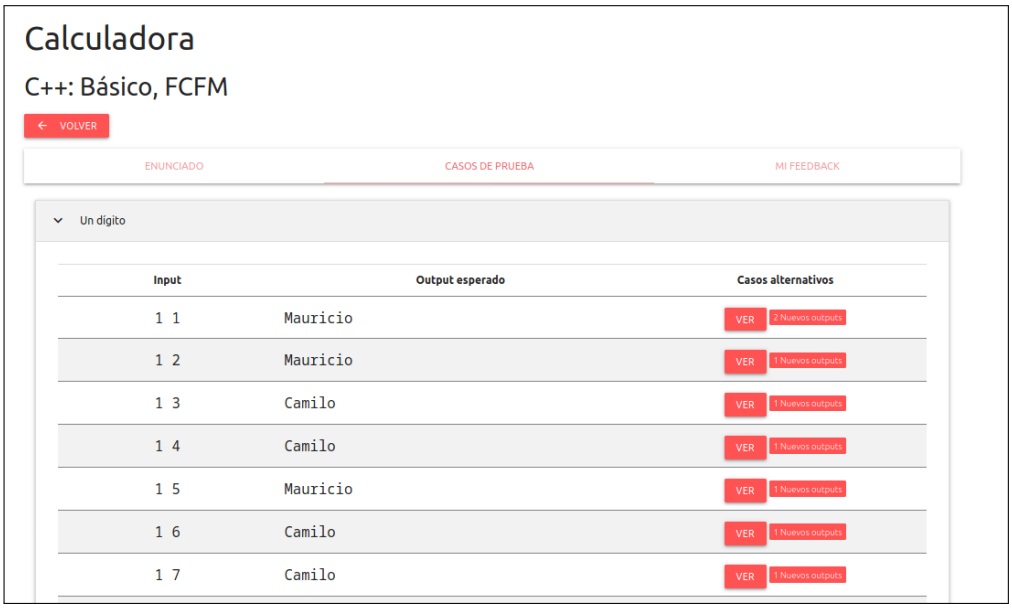


Figura 4.8: Vista de los casos de prueba de un problema.

Figura 4.9: Vista del modal de outputs alternativos.

Bajo el título del modal se añadió una explicación sobre la información que se encuentra en este modal ya que las docentes no conocerán el concepto de *Output Alternativo* al comenzar a utilizar el sistema, entonces necesitarán una guía.

4.2.4. Módulo de seguimiento

Para este módulo se implementó una tabla de seguimiento y la interfaz final se encuentra en la Figura 4.10. Esta interfaz sufrió cambios desde los mockups ya que se consideró importante mostrar el nombre de cada *Problema* en la tabla y no solo representarlos con un símbolo. Para dar orden a la información se agregó sobre los *Problemas* el nombre de cada *Clase*.

Si se quisieran mostrar totales en esta tabla, habría que mostrar total de *Problemas* resueltos, intentados y no intentados. Debido al tiempo, no se alcanzó a diseñar cómo visualizar totales, por lo que queda para el trabajo futuro.

Otra funcionalidad que se podría agregar, es poder ver estadísticas para cada clase. Gracias a esto se tendría información más específica y se podrían incluir más variables. El diseño e implementación de esta funcionalidad queda para el trabajo futuro.

4.2.5. Módulo de asistencia

En la Figura 4.11 se encuentra la vista final de la asistencia general. En esta, cuando una estudiante no asiste se verá una cruz y si asiste se verá un tick. Se espera que con esta interfaz sea fácil detectar patrones de asistencia o inasistencia para apoyar a quienes hayan faltado mucho, entregándoles material y ayuda; y para premiar a quienes tienen mejor asistencia

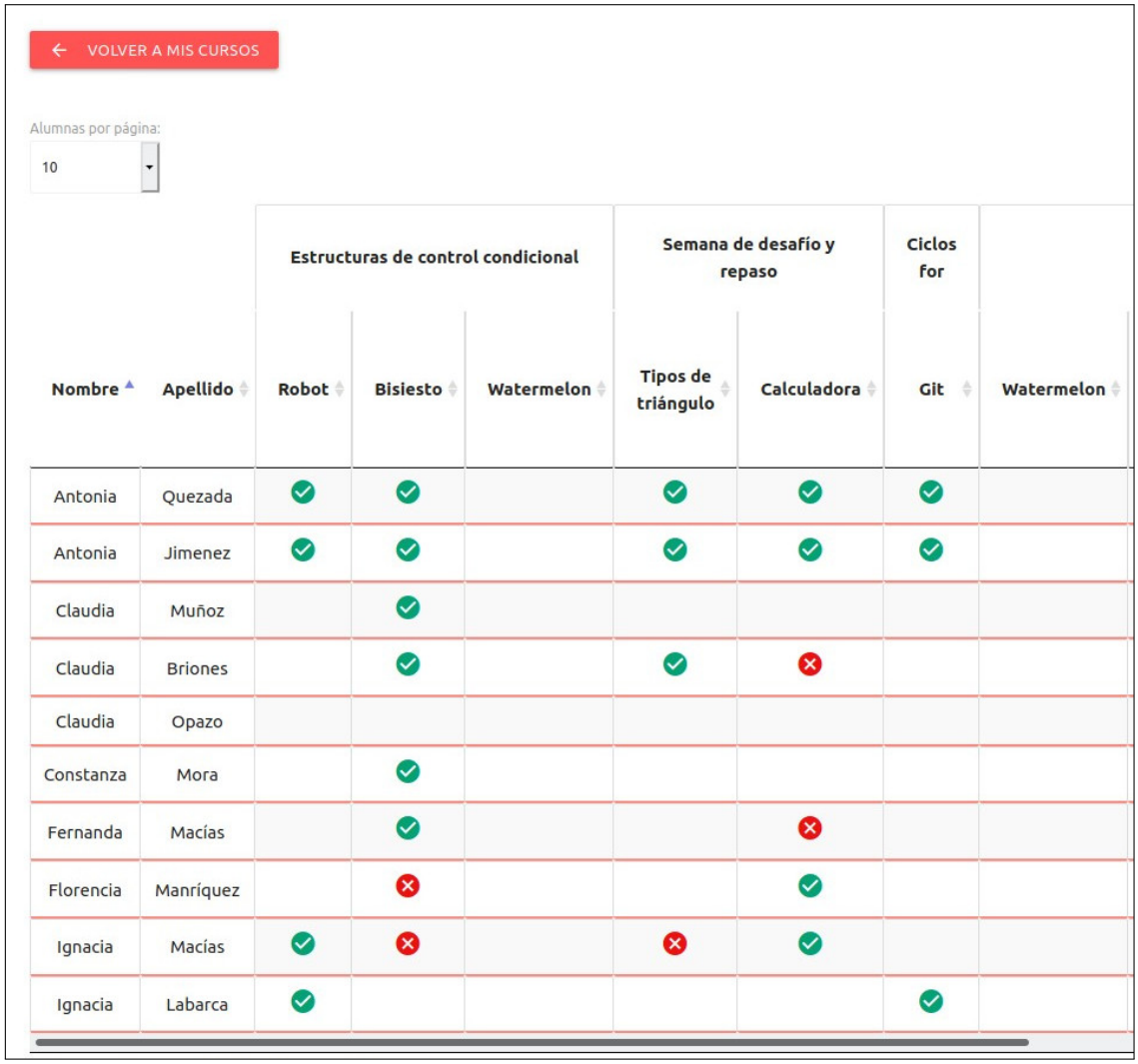


Figura 4.10: Vista de la tabla se seguimiento.

para motivarlas a seguir así.

En una primera versión del módulo, solo se mostraba el total de estudiantes que asistieron a una clase pero luego de hacer una validación de usabilidad, se concluyó que había que mostrar también el total de inasistencias. Luego de esta validación también se agregó la opción de ordenar cada columna en orden ascendente o descendente para facilitar el análisis de la tabla.

Una decisión importante que se debió tomar fue cuándo activar la opción de pasar asistencia, y por cuánto tiempo. Dado que se necesita tener información de todas las clases y que esta sea fidedigna, se determinó que la asistencia tiene que agregarse si o si durante el horario de las clases. Si no fuera así, una tutora podría anotar asistencia en un papel o en otro sistema, y querer traspasarlo a este sistema después, pero esto aumenta las posibilidades de tener información incompleta, si se llegan a olvidar. Para complementar esto, en un trabajo futuro se añadirá una alerta para que las tutoras recuerden pasar la asistencia.

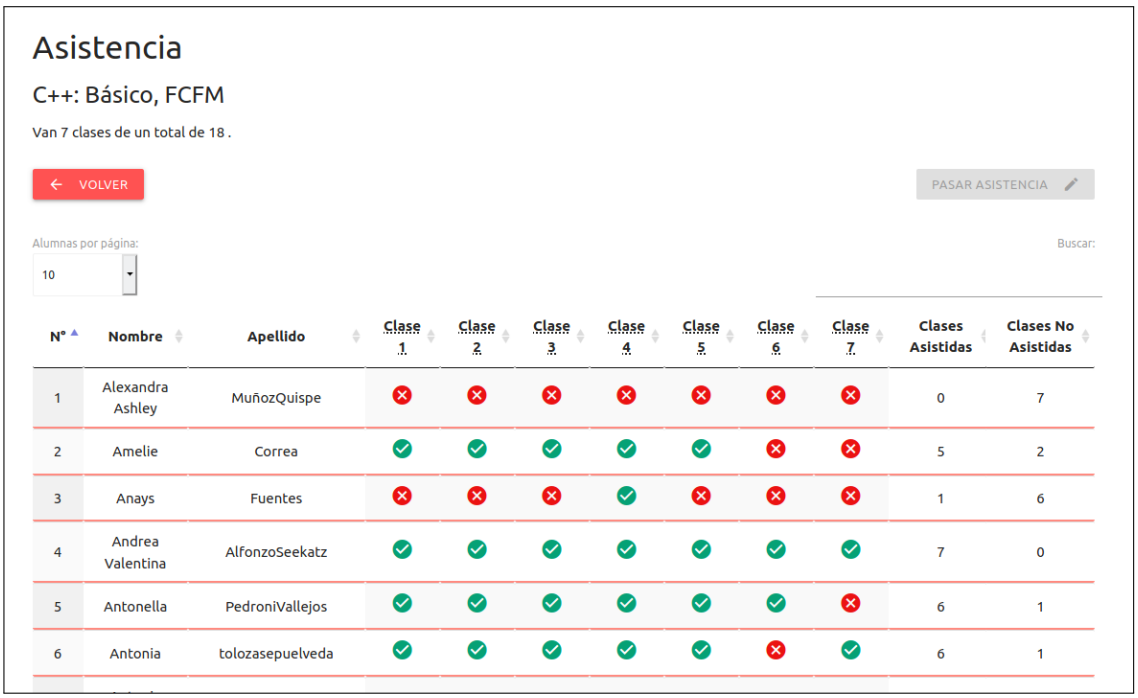


Figura 4.11: Vista de la asistencia general.

4.3. Resumen

En este capítulo se explicó cómo se realizó la implementación del sistema diseñado en el Capítulo 3. Primero se explicó como se agregó C++ en el sistema, para que este pueda revisar códigos en este lenguaje.

Luego, para la aplicación web, se explicaron aspectos generales de esta, para luego mostrar cómo se implementaron los cuatro módulos diseñados. Únicamente en la Subsección 4.2.2 se explicó detalladamente el proceso para implementar una vista, ya que para la implementación de todas las vistas se siguió el mismo proceso. Para el resto de la funcionalidad, se presentó la funcionalidad obtenida y los problemas que surgieron en la implementación.

Capítulo 5

Validación

Dados los objetivos de esta memoria, se diseñaron tres validaciones con tutoras y estudiantes del curso. La primera fue un caso de estudio exploratorio con las estudiantes, para entender si la información entregada por el sistema es útil para resolver los errores que tienen sus soluciones. Luego, se hizo una prueba de usabilidad con las tutoras para validar los módulos de cursos, feedback y seguimiento. Finalmente se hizo una prueba de usabilidad del módulo de asistencia con tutoras, organizado por las practicantes de Niñas Pro.

Cabe mencionar que parte de esta memoria se desarrolló en el contexto de la pandemia por Covid-19, por lo que algunas evaluaciones necesariamente tuvieron que ser de forma virtual. En estos casos se hizo una videollamada con cada participante por separado. Para las tres validaciones se solicitó a las participantes una autorización para grabar lo que se habló, y en el caso de las videollamadas se grabó también la pantalla de las participantes.

La metodología a utilizar en los tres casos será "Thinking Aloud" o pensar en voz alta. Este método se basa en observar a los usuarios resolviendo una tarea y se les pide que hablen en voz alta sobre lo que ven y están haciendo. Al finalizar las pruebas con usuarias, se revisaron todas las grabaciones y se tomó nota de las respuestas y comentarios de las participantes, además de tomar el tiempo que demoraron en realizar ciertas tareas.

En este capítulo se contará cómo se diseñó cada una de las validaciones, cómo se implementó y los resultados obtenidos.

5.1. Caso de estudio exploratorio con estudiantes

Para la evaluación de utilidad se realizó un caso de estudio exploratorio [16] con estudiantes del Curso Anual de Niñas Pro 2020. El propósito fue entender si la información entregada por el sistema, al dar feedback a las estudiantes, es útil para resolver los errores que tienen sus soluciones.

Para mejor comprensión de esta sección es importante recordar que una sugerencia es un

mensaje que las tutoras agregan cuando muchas soluciones generan un mismo *output* erróneo y, un caso de prueba es un conjunto entre un *input* y un *output* con que se prueba una solución a un problema.

5.1.1. Diseño del caso de estudio

Las preguntas específicas a responder fueron las siguientes:

P1 : ¿Cómo utilizan las estudiantes las sugerencias que les entrega el sistema mientras resuelven problemas?

P2 : ¿Cómo las estudiantes utilizan los casos de prueba mientras resuelven problemas?

A continuación se describen los pasos para el diseño de este caso de estudio:

1. Dado el propósito del caso de estudio, se determinó que las estudiantes que participaran debían tener experiencia en programación y en resolución de problemas de estilo competitivo, para enfocar la evaluación en cómo corrigen sus errores con la plataforma y no en ayudarlas a entender y resolver el problema. Dado esto, fueron invitadas a participar las estudiantes del nivel avanzado del Curso Anual de la Sede Santiago. En este nivel participan 11 estudiantes, de las cuales 9 participan del Curso Anual por segunda vez o más, y las otras dos participan por primera vez pero habían programado antes en otros lenguajes. De este grupo, 5 participaron del caso de estudio, y sus edades rondan entre los 15 y 18 años.
2. La validación se hizo por videollamada con cada estudiante por separado. Dado que varias de las participantes eran menores de edad, se solicitó el consentimiento de un apoderado para participar y autorización para grabar la videollamada, además de un asentimiento informado de la participante.
3. Se escogieron 2 problemas para que las estudiantes resolvieran apoyándose de la plataforma. El primer problema, llamado "Bisiesto", consistía en determinar si un año era bisiesto o no y cuenta con 10 casos de prueba. Este problema contaba con dos sugerencias al iniciar la evaluación y se agregaron nuevas sugerencias mientras las estudiantes resolvían el problema.

   El segundo problema, llamado "Robot", consistía en imprimir en pantalla una cantidad N de cuentas regresivas. Este contaba con 6 casos de prueba y 4 sugerencias previo a la evaluación.

   El enunciado y los casos de cada problema, se encuentran en detalle en el Anexo A. El problema Bisiesto tenía una dificultad menor que el problema Robot, por lo que la estudiante debía completar el problema Bisiesto antes de seguir con el otro.
4. Cuando una estudiante suba una solución al sistema, la memorista le hará ciertas preguntas con el objetivo de complementar lo que esta observa, con la percepción de las estudiantes hacia el feedback. Esto para luego responder las preguntas de investigación. Las preguntas son las siguientes:
   (a) ¿Qué ves en la plataforma?
   (b) ¿Pasaste todos los casos? ¿En qué te fijaste para saber esto?

(c) Si no los pasaste todos, ¿Qué te faltó?

(d) ¿Cómo piensas solucionar el código? ¿Harás algo con la información que te dio la plataforma? ¿Qué?

5.1.2. Realización del caso de estudio

Dado que este es un caso de estudio exploratorio y no una prueba de usabilidad, al empezar la evaluación se le enseñó a cada estudiante cómo subir una solución. Para esto, antes del inicio de cada videollamada se les envió a las estudiantes un archivo de formato cpp, para subirlo al sistema, como si lo fueran a revisar.

Luego de completar esta tarea exitosamente, las estudiantes pasaron a resolver los problemas. Cada vez que una estudiante iba a probar su solución en la plataforma, debía avisar a la memorista para ver los resultados en conjunto. Al recibir el feedback de la plataforma, la memorista dejó que las estudiantes analizaran sus resultados y luego les preguntaba cuál era su interpretación de estos. Si no completaban todos los casos del problema, se les preguntaba cómo iban a arreglarlo.

En la Tabla 5.1 se encuentra un resumen de los intentos realizados por cada estudiante. En total para el problema Bisiesto se realizaron 21 intentos, y para el problema Robot se realizaron 16. Cuatro de las cinco estudiantes lograron resolver ambos problemas, y la estudiante que no terminó el problema Robot fue debido a que se acabó el tiempo de la entrevista. Es importante destacar que el intento final, cuando la estudiante aprobó todos los casos de prueba está considerado en esta cuenta.

	E1	E2	E3	E4	E5	Total
Problema Bisiesto	5	6	5	3	2	21
Problema Robot	3	2 (No termina)	8	2	1	16

Tabla 5.1: Cantidad de intentos realizados por cada estudiante para los problemas Bisiesto y Robot hasta aprobar todos los casos.

En la Tabla 5.2 se encuentra la cantidad de veces que las estudiantes utilizaron los casos de prueba o las categorías para arreglar su solución. De esta se destaca que las estudiantes E4 y E5 utilizaron únicamente los casos de prueba para resolver el problema.

	E1	E2	E3	E4	E5	Total
Problema Bisiesto	2	2	1	2	1	8
Problema Robot	1	2	2	1	0	6

Tabla 5.2: Cantidad de veces que las estudiantes utilizaron los casos de prueba para identificar un error.

En la Tabla 5.3 se encuentra la cantidad de sugerencias que recibieron las estudiantes en el total de sus intentos para cada problema. De esta tabla se destaca que para el problema Robot, solo dos estudiantes recibieron sugerencias, pero es importante considerar que una estudiante no alcanzó a terminar el problema y que una estudiante lo resolvió al primer intento, por lo que no se le dio feedback.

	E1	E2	E3	E4	E5	Total
Problema Bisiesto	1	1	2	0	1	5
Problema Robot	1	0	2	0	0	2

Tabla 5.3: Cantidad de veces que las estudiantes recibieron una sugerencia.

En los intentos en que las estudiantes no utilizaron ni los casos de prueba ni sugerencias, la memorista fue quien las ayudó a mejorar su código. Se podrían haber subido sugerencias muy específicas para la solución, pero ese no es el objetivo de la plataforma, por lo tanto la memorista tomó el rol de "tutora".

Luego de revisar las entrevistas nuevamente, se clasificaron los errores de las soluciones de las estudiantes en 3 categorías: error de compilación, error de presentación y error de algoritmo. Un error de compilación es cuando la solución no compila y se le muestra un mensaje de error a la estudiante. Un error de presentación es cuando el *output recibido* de un caso de prueba tiene el valor correcto, pero no está escrito de la forma en que se solicita. Por ejemplo, si el *output* de la solución del estudiante dice "NO" o "no" cuando se pide escribir "No". Finalmente un error de algoritmo, es cuando algún *output* generado por la solución no es el valor esperado.

Durante las pruebas, hubo 3 intentos de diferentes estudiantes para el problema Bisiesto que dieron error de compilación. Al recibir este error las estudiantes leyeron el mensaje entregado por la plataforma (explicado en la Sección 4.1) y solicitaron ayuda de la memorista. Dos de estos errores fueron por el uso de la letra "ñ", ya que el servidor no soportaba este carácter, y el otro error fue debido a que la estudiante copió su código desde el compilador online a un archivo y olvidó copiar un carácter.

A continuación se presentan los resultados obtenidos para las dos preguntas de investigación:

5.1.2.1. P1: ¿Cómo utilizan las estudiantes las sugerencias que les entrega el sistema mientras resuelven problemas?

Como fue la primera vez que se utilizaba el sistema, no existían muchas sugerencias ingresadas de antemano. Es por esto que cuando una estudiante tenía un error en un caso de prueba y la memorista consideraba que se le podía dar una sugerencia, esta la agregaba y la estudiante subía nuevamente la solución para tener acceso a esta sugerencia. En total se añadieron 3 sugerencias para el problema Bisiesto, y para el problema Robot se añadieron 2, como se ve en la Tabla 5.4. Con esto, al final del caso de estudio, el sistema tenía 5 sugerencias para el problema Bisiesto y 6 sugerencias para el problema Robot. Por otro lado, cuando ocurrió que si existía una sugerencia, no todas las estudiantes lo notaron, ya que se fijaban más en los *outputs* o la clasificación del problema.

Un ejemplo de sugerencia es el siguiente: *El output del problema debe ser "Si" con s mayúscula o "No" con n mayúscula.* Esta sugerencia tiene relación con un error de presentación y las estudiantes que obtuvieron sugerencias como esta lograron comprenderlas rápidamente.

	# de sugerencias previo a la evaluación	# de sugerencias ingresadas durante la evaluación	Total
Bisiesto	2	3	5
Robot	4	2	6

Tabla 5.4: Cantidad de sugerencias en el sistema durante el caso de estudio.

Por otro lado, en 5 ocasiones las estudiantes obtuvieron una sugerencia que hacía referencia a un error de algoritmo. De estas, dos permitieron que las estudiantes arreglaran su error sin ayuda. Para los otros casos, las estudiantes entendieron lo que decía la sugerencia, pero esto no les permitió saber cómo cambiar su código para seguir la sugerencia. Por lo tanto, se cree que habrá que evaluar más en profundidad el contenido de estas sugerencias y cómo hacer que entreguen información útil para las estudiantes. A pesar de esto, se cree que las sugerencias sí serán un aporte al seguimiento de las estudiantes, ya que con esta información las tutoras podrán saber rápidamente dónde buscar el error del código de la estudiante.

5.1.2.2. P2: ¿Cómo las estudiantes utilizan los casos de prueba mientras resuelven problemas?

Se observó que las estudiantes hicieron mayor uso de los casos de prueba que de las sugerencias, para identificar dónde estaba el error de su solución. A continuación se describen los distintos usos que se le dieron a los casos de prueba.

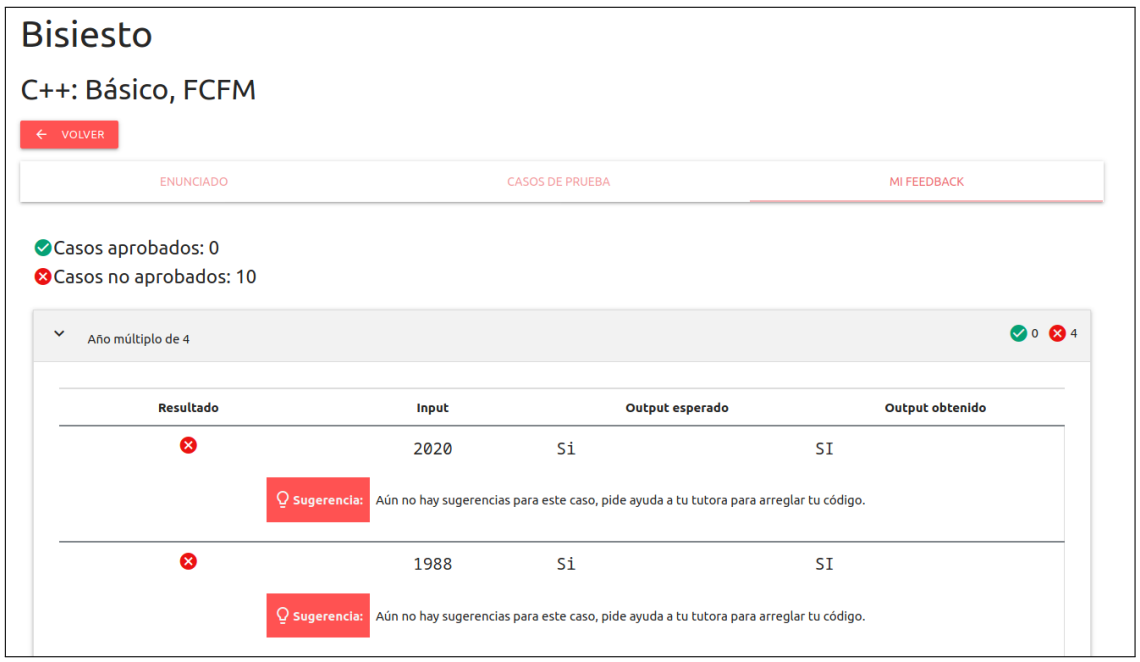


Figura 5.1: Feedback a solución de problema Bisiesto, con error de presentación.

En la Figura 5.1 se encuentra la vista del feedback a la solución de una estudiante que tuvo un error de presentación. Para feedback como este, se observó que las estudiantes notaban que el *output obtenido* y el *output esperado* tenían el mismo valor, pero el caso aparecía como no aprobado. Ante este resultado la mayoría comparaba los *outputs* hasta identificar la

diferencia entre el *output recibido* y el *output esperado*, que en el caso de la figura, es que la plataforma espera el mensaje "Si" y se recibe el mensaje "SI". En el caso del problema Robot, una estudiante también tuvo un error de presentación ya que escribió "Siguiete ejercicio" (sin n en Siguiente). Este error fue más difícil de identificar porque el *output* era más complejo. Luego de observar los casos esperados y obtenidos con atención, la estudiante logró identificar el error y arreglarlo. Luego de esto la estudiante comentó textual :

> "Me gusta, en los que me equivoqué me pude dar cuenta solita en que me equivocaba. Por ejemplo si lo hubiera tirado a Vjudge, dice error, y por más pruebas que una haga, iba a ser más difícil encontrar el error".

En el caso de tener un error de algoritmo, los casos de prueba también fueron utilizados por las estudiantes. Al probar su solución las estudiantes se fijaban específicamente en los casos que no pasaron y en el nombre de la categoría de los casos. Para el problema Bisiesto las categorías estaban definidas por las diferentes condiciones del problema (por ejemplo: Año múltiplo de 4), por lo tanto, al ver que todos los casos de una categoría fallaban se les hizo fácil identificar en qué parte del código estaba el error. Por otro lado, en el problema Robot, las categorías no estaban directamente relacionadas con una parte del algoritmo de la solución, por lo que las estudiantes no les dieron uso.

Otro uso que le dieron a la plataforma fue para ir viendo si su solución iba por "buen camino". Una estudiante subió 8 veces (el resto subió entre 3 y 5 veces) su solución ya que a medida que hacía cambios en el código iba viendo si el *output recibido* se parecía más al *output esperado*.

Se observó también que la mayoría de las estudiantes utilizaron los casos que les entregó el sistema para probarlos en su propio compilador.

5.1.3. Análisis de la validación

Las estudiantes utilizaron activamente la plataforma para identificar dónde se habían equivocado en su solución y la mayoría de las veces fueron capaces de solucionar los errores gracias a la información de la plataforma. Esto, siempre y cuando ellas tuvieran clara la solución que estaban implementando. Hubo un caso en que la estudiante no leyó con atención el enunciado o no tenía una idea clara del algoritmo a utilizar para resolver el problema y, a pesar de entender dónde fallaba su código no fue capaz de arreglarlo sin ayuda. En este sentido, el estudio permitió confirmar que este sistema de feedback será de mayor utilidad cuando una estudiante tiene claridad sobre el algoritmo a desarrollar. En caso contrario, la herramienta servirá para que las tutoras puedan apoyar fácilmente a otras estudiantes.

Respecto a la pregunta de investigación *P1: ¿Cómo utilizan las estudiantes las sugerencias que les entrega el sistema mientras resuelven problemas?* se concluye que para errores de presentación las estudiantes comprendieron la sugerencia y pudieron arreglar su código rápidamente, en cambio que para errores de algoritmo algunas estudiantes comprendieron la sugerencia, pero no lograron hacer cambios en su algoritmo en base a esta. Será importante evaluar la estructura que se le da a las sugerencias, para que las estudiantes puedan no

solamente encontrar su error, sino que arreglar su código sin ayuda de las tutoras.

Respecto a la pregunta de investigación *P2: ¿Cómo las estudiantes utilizan los casos de prueba mientras resuelven problemas?* se concluye que las estudiantes utilizan los casos de prueba tanto para arreglar errores de presentación como para arreglar errores de algoritmo. Para errores de presentación, las estudiantes comparan el *output obtenido* con el *output esperado* y logran corregir su código. Por otro lado, para errores de algoritmo, las categorías de los casos de prueba pueden ser muy relevantes, ya que pueden ayudar a orientar a las estudiantes en cómo crear el algoritmo de la solución. Además, las estudiantes utilizan los casos de prueba para probar su solución antes de subirla al sistema y saber de antemano si un caso pasa o no.

5.2. Evaluación de usabilidad módulo de asistencia

5.2.1. Diseño de la validación

La validación del módulo de asistencia fue la primera validación de usabilidad que se hizo del sistema. Dado esto se hizo una evaluación formativa del módulo, es decir se evaluó un diseño intermedio, para tener feedback de cómo seguir con la implementación. Esta evaluación fue diseñada por Karina Parga, practicante de Niñas Pro experta en usabilidad, y fue respaldada por la memorista. Los pasos para el diseño de este proceso fueron los siguientes:

- Para determinar la cantidad de usuarias para la validación del sistema se usó la metodología propuesta por Turner et al. [13] que dice que de 3 a 5 usuarios bastan para determinar la mayoría de los problemas de usabilidad. En esta validación participaron 5 tutoras del Curso Anual de la Sede de Santiago.

  En la Tabla 5.5 se encuentra una caracterización de las tutoras que participaron de esta validación, donde se ve que la experiencia de las participantes en el curso de programación es muy variada, y su experiencia con programación también, lo que aportará diferentes puntos de vista a la validación.

	T1	T2	T3	T4	T5
# veces que ha participado del curso	5	4	1	2	3
Experiencia en programación	Magíster en Ciencias de la Computación y Desarrolladora de Software	Doctorado en Ciencias de la Computación y Desarrolladora de Software	Estudiante de 3er año de Ciencias de la Computación, ex-estudiante del Curso Anual	Profesor de matemáticas, con experiencia en programación.	Ingeniera Eléctrica, estudiante de magíster en el Departamento de Ingeniería Eléctrica.

Tabla 5.5: Caracterización de las tutoras participantes de la evaluación de usabilidad del módulo de Asistencia.

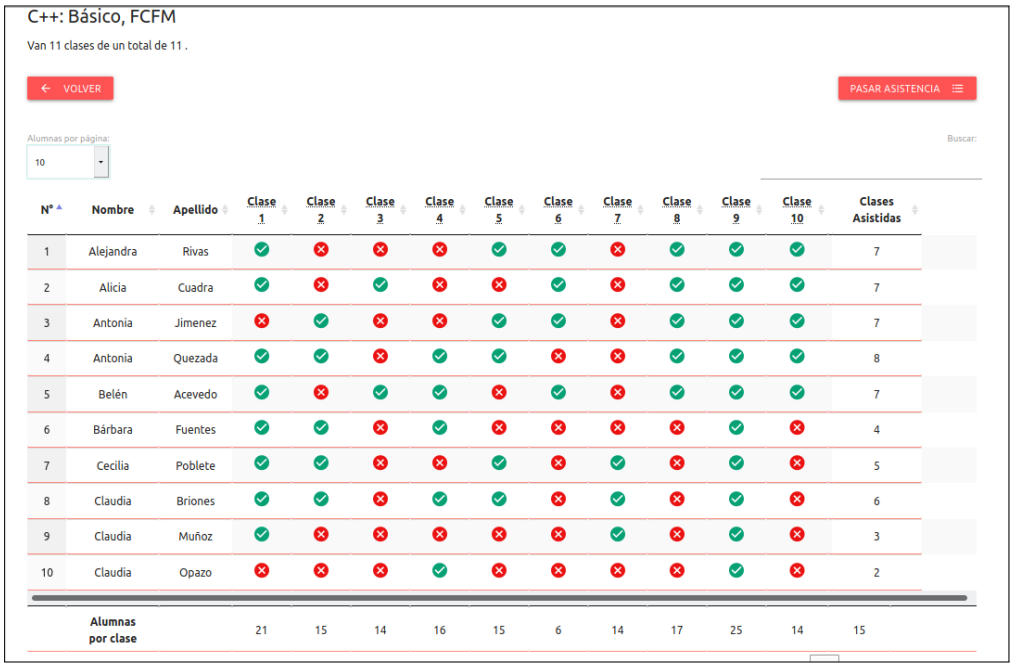
- Esta validación se hizo de forma presencial en las dependencias del Departamento de Ciencias de la Computación de la Universidad de Chile. Se entrevistó de forma separada

a cada tutora y se grabó el audio de la entrevista, con autorización de cada una.

- Se diseñaron cuatro tareas a realizar por las participantes, con las que se espera evaluar si las tutoras comprenden la información que entrega la tabla de asistencia general y si la interfaz de pasar asistencia es usable. A continuación se listan las cuatro tareas que resolvieron las tutoras en esta validación y en la siguiente sección se muestran capturas de pantalla que ayudarán a comprender las tareas:
  1. Indica cuántas niñas asistieron a las primeras 3 clases del curso "C++ Básico".
  2. Indica a cuántas clases ha faltado la alumna Antonia Jimenez
  3. En la clase 12 no se ha pasado asistencia. Pasa la asistencia teniendo en cuenta que asistieron las siguientes alumnas: Alejandra Rivas, Alicia Cuadra, Belén Acevedo y Claudia Briones.
  4. Después de pasar la asistencia, quién es la niña que más ha faltado a clases?
- Dado que la validación se realizó con un diseño intermedio, algunas funcionalidades no estaban completamente disponibles. En estos casos se utilizó la metodología "mago de oz" que consiste en que ciertas funcionalidades están controladas por un ser humano en vez del sistema, para que la persona entrevistada vea los resultados de una acción que no está implementada.

5.2.2. Implementación de la validación

En la Figura 5.2 se encuentra la vista de asistencia general que veían las tutoras en la validación. Para la primera tarea, todas reconocieron que podían ver el total de asistentes al curso al final de la tabla y responder rápidamente.



N°	Nombre	Apellido	Clase 1	Clase 2	Clase 3	Clase 4	Clase 5	Clase 6	Clase 7	Clase 8	Clase 9	Clase 10	Clases Asistidas
1	Alejandra	Rivas	✓	✗	✗	✗	✓	✓	✗	✓	✓	✓	7
2	Alicia	Cuadra	✓	✗	✓	✗	✗	✓	✗	✓	✓	✓	7
3	Antonia	Jimenez	✗	✓	✗	✗	✓	✓	✗	✓	✓	✓	7
4	Antonia	Quezada	✓	✓	✗	✓	✓	✗	✗	✓	✓	✓	8
5	Belén	Acevedo	✓	✗	✓	✓	✗	✓	✗	✓	✓	✓	7
6	Bárbara	Fuentes	✓	✓	✗	✓	✗	✗	✗	✗	✓	✗	4
7	Cecilia	Poblete	✓	✓	✗	✗	✓	✗	✓	✗	✓	✗	5
8	Claudia	Briones	✓	✓	✗	✓	✓	✗	✓	✗	✓	✗	6
9	Claudia	Muñoz	✓	✗	✗	✗	✗	✗	✓	✗	✓	✗	3
10	Claudia	Opazo	✗	✗	✗	✓	✗	✗	✗	✗	✓	✗	2
	Alumnas por clase		21	15	14	16	15	6	14	17	25	14	15

Figura 5.2: Vista de la asistencia general para validación del módulo de Asistencia.

En la segunda tarea, sobre cuántas veces ha faltado una estudiante, se les hizo más difícil

responder ya que solo existía el total de clases asistidas por cada estudiante pero no las clases a las que no han asistido. Dos tutoras contaron las inasistencias a mano, y el resto hizo la resta entre las clases totales y las clases asistidas. Gracias a esta pregunta se identificó que habrá que agregar una columna de clases no asistidas para hacer una análisis más directo de la asistencia.

La tercera tarea consistía en pasar asistencia para la siguiente clase. Al realizar esta tarea algunas tutoras tuvieron dificultad para encontrar a las estudiantes. Estas aparecen ordenadas por nombre, pero estas tutoras las buscaron por apellido lo que dificultó su búsqueda. Es importante considerar la posibilidad de ordenar a las estudiantes por nombre o apellido en una futura versión del sistema, porque se espera que pasar asistencia sea una tarea rápida y sencilla. Además, en la tabla de asistencia general si se puede ordenar por nombre o apellido por lo que la plataforma debe ser consistente.

Otra observación que se hizo de esta interfaz es que no tiene un botón de "Volver" o "Cancelar", y que la zona para hacer check de una estudiante es muy pequeña. En el primer caso la tutora estaba obligada a pasar asistencia una vez que abría esta vista, lo cual no genera una buena experiencia de usuario ni una buena navegación en la plataforma. Respecto a la zona para hacer check, es importante considerar agrandarla, ya que entorpece la acción de pasar asistencia.

Cuando se les pidió identificar a la estudiante que más había faltado a clases (cuarta tarea), algunas tutoras supieron rápidamente que debían buscar a las estudiantes que tenían el menor número de asistencia, otras no tan rápidamente, y otras intentaron contar las inasistencias en las estudiantes que parecían tener más símbolos rojos. Con este resultado se confirma la necesidad de incorporar una columna con el total de inasistencias a las clases por cada estudiante. Es importante destacar que algunas tutoras buscaron patrones de inasistencias con los colores de la tabla. Esto confirma que mostrar diferentes símbolos y colores, con esa distancia entre filas y columnas será útil para el trabajo de las tutoras.

Varias tutoras observaron que cuando la lista de estudiantes sea muy larga, el total de asistencia por clase no se verá directamente al ingresar a la vista, sino que se tendrá que bajar en la página para encontrar esta información. Frente a esto, las participantes consideraron que podría ser incómodo, sobre todo en cursos grandes con 40 o 50 estudiantes.

Otro comentario que surgió de varias tutoras respecto a la asistencia general es que falta el nombre de cada clase en la tabla, ya que no es intuitivo reconocer una clase solo por su número.

5.2.3. Análisis de la validación

Luego de todas las entrevistas, se escucharon las grabaciones de las pruebas con tutoras para identificar la frecuencia con que surgieron algunos comentarios y la importancia que estas le dieron.

De este análisis las modificaciones que se necesitaron hacer al sistema fueron las siguientes:

- Agregar botón de "Volver" al pasar asistencia para mejorar la navegación del sistema.
- Agregar columna con el total de clases no asistidas por una estudiante.
- Tener la opción de ordenar por nombre o apellido a las estudiantes cuando se pasa la asistencia.
- Agrandar la zona *clickeable* al pasar la asistencia de una estudiante.
- Agregar el nombre de cada clase a la tabla.
- Al entrar a la asistencia general mostrar los totales de alumnas por clase dentro de la ventana.

No todas estas funcionalidades fueron implementadas debido al tiempo y experiencia de las practicantes. Pero en versiones futuras de la plataforma se considerarán estos resultados.

A pesar de estas sugerencias, las tutoras lograron utilizar la plataforma sin ayuda de las entrevistadoras, pudieron extraer información de la tabla de asistencia general y pudieron pasar asistencia. Con este resultado se concluye que las historias de usuario **HD4** y **HD5** están cubiertas por el sistema.

5.3. Validación de usabilidad de módulos de Cursos, Feedback y Seguimiento

5.3.1. Diseño de la validación

El objetivo de esta validación es determinar la usabilidad de los módulos de Cursos, Feedback y Seguimiento implementados, que serán utilizados por las tutoras del Curso Anual. Los pasos para el diseño de este proceso fueron los siguientes:

- Para la selección de las tutoras que participarían de la evaluación se realizó un llamado abierto a 8 tutoras de la sede Santiago que no tenían mayor conocimiento del funcionamiento de la plataforma. De estas 8 tutoras invitadas, 5 participaron de la validación.

	T1	T2	T3	T4	T5
# veces que ha participado del curso	1	3	1	2	1
Experiencia en programación	Estudiante de 4to año ing en computación	Profesor de matemáticas con conocimientos en programación	Estudiante de 4to año ing en computación	Ingeniera civil en programación, trabaja en desarrollo web	Estudiante de 6to año de ingeniería civil en computación
Participó de la validación de asistencia	No	Si	No	Si	No

Tabla 5.6: Caracterización de las tutoras participantes de la evaluación de usabilidad de los módulos de Cursos, Feedback y Seguimiento.

En la Tabla 5.6 se encuentra el perfil de las 5 tutoras que participaron de esta validación. De esta tabla se destaca que tres tutoras están participando por primera vez del Curso Anual y las mismas tutoras no participaron antes de la validación de usabilidad del módulo de asistencia, por lo tanto la validación era la primera vez que se enfrentaban al sistema. Por otro lado, como se ve en la tabla, las tutoras participantes tienen diferentes grados de experiencia en programación, lo que le da diversidad de miradas a la evaluación.

- Esta validación se hizo en el contexto de distanciamiento social por lo tanto se hizo una videollamada con cada participante por separado. En cada videollamada se solicitó a la participante compartir su pantalla con la evaluadora. Además, cada participante autorizó ser grabada, para que luego se analizaran los resultados de cada evaluación.
- Se evaluaron 3 partes del sistema: la tabla de avances de las estudiantes, los casos de prueba alternativos y sugerencias, y la creación de problemas. Se definieron 4 tareas para que las tutoras resolvieran usando el sistema y en base a estas, este se pobló con información realista. Con estas tareas se espera evaluar si el sistema es usable sin que las usuarias tengan información previa sobre su funcionamiento.

  Además, al momento de comenzar cada validación las tutoras recibieron dos archivos; un archivo de tipo `pdf` con el enunciado de un problema y un archivo de tipo `json` con los casos de prueba para ese problema, para ser utilizados luego en la validación.

  A continuación se encuentras las tareas que las tutoras realizaron y en la siguiente sección se encuentran capturas de pantalla de la información que las tutoras vieron en la plataforma :

  1. Revisar la tabla de avance de las estudiantes e identificar qué estudiantes han resuelto menos problemas. ¿Hay alguna estudiante que vaya más adelantada que las otras? ¿Cómo lo identificas? .
  2. Ir a los casos del problema "Tipos de triángulo" de la clase de condicionales. ¿Detectas casos de prueba donde se puedan agregar sugerencias? ¿Qué crees que significa esto? .
  3. Agregar una sugerencia para el caso cuyo *intput* es `6 6 10` para que las estudiantes puedan arreglar su código si tienen este error. La sugerencia puede ser: "Hay un error de presentación. Las letras debe ir en minúsculas y con las tildes correspondientes."
  4. El próximo sábado es la clase de *ciclos while* y falta agregar un problema, agrega el problema "Watermelon", con el enunciado y el archivo de casos de prueba que se te entregaron.
- Para complementar las observaciones de la memorista, al terminar cada prueba se le entregó a cada tutora un cuestionario SUS [2] para medir cuantitativamente la usabiliad del sistema. SUS es una escala de usabilidad donde se genera un puntaje respecto a las respuestas que un usuario deja en un cuestionario que consta de 10 preguntas con 5 opciones de respuesta en escala Linkert. Las preguntas son del estilo "Me gustaría usar este sistema frecuentemente" o "Encontré el sistema innecesariamente complejo". A este cuestionario se le asigna un puntaje que será entre 0 y 100. Jeff Sauro [11] hizo un estudio de más de 500 pruebas de usabilidad usando SUS y concluyó que el puntaje promedio entre los 500 sistemas fue 68 puntos y que sobre 80.3 puntos se estará entre el 10 % de los mejores sistemas.

En esta evaluación se considerará el puntaje que entregue cada tutora por separado, ya que la cantidad de participantes no fue grande.

A continuación se mostrarán los resultados de la validación para las diferentes partes que se evaluaron y finalmente se hará un análisis de estos resultados.

5.3.2. Implementación de la validación

Dado que se evaluó la usabilidad de 3 partes diferentes del sistema, los resultados de esta evaluación se presentan para cada parte.

5.3.2.1. Tabla de avance

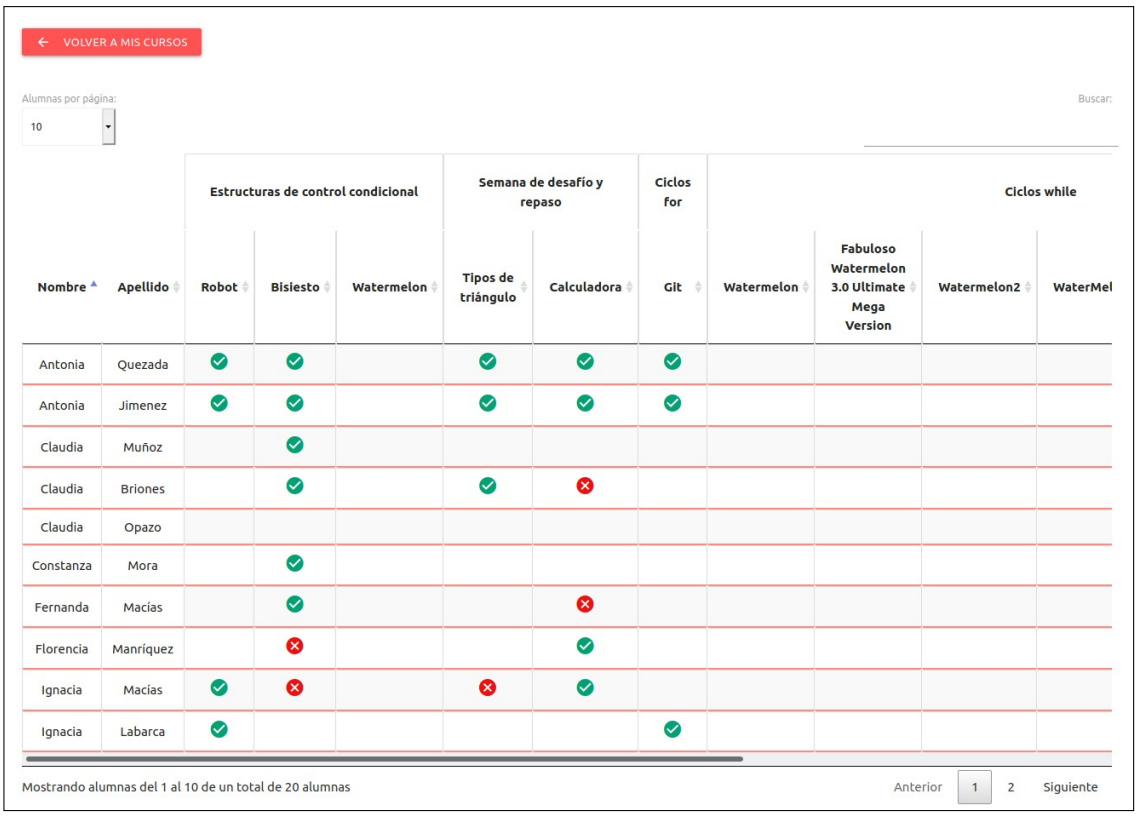


Figura 5.3: Tabla de avance de estudiantes utilizada para pruebas de usabilidad con tutoras, página 1.

Lo primero que se les solicitó a las tutoras fue revisar la tabla de avance del curso básico (Figuras 5.3 y 5.4). En la primera figura se ve que las estudiantes Antonia Quezada y Antonia Jimenez tienen solo tickets verdes; en ambas figuras hay estudiantes que no han subido ningún problema y en la segunda figura hay una estudiante que solo tiene cruces rojas. Al entrar en esta vista las tutoras tuvieron un tiempo para acostumbrarse a ella y luego debían identificar qué estudiantes creían ellas que estaban más atrasadas o cuáles necesitaban más ayuda.

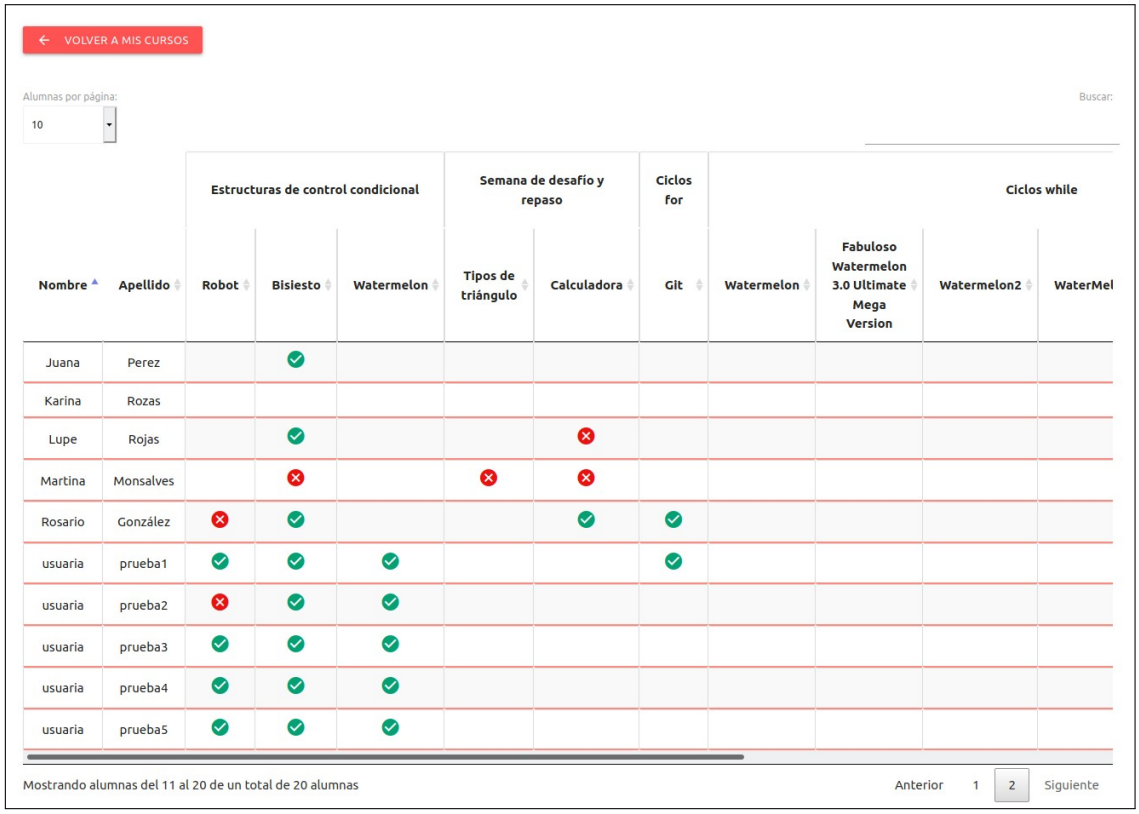


Figura 5.4: Tabla de avance de estudiantes utilizada para pruebas de usabilidad con tutoras, página 2.

Como resultado a esta tarea se observó lo siguiente:

- Todas las tutoras demoraron alrededor de 30 segundos en responder.
- Todas las tutoras identificaron que las estudiantes que no habían subido ninguna solución son las que van más atrasadas.
- 3 de las 5 tutoras identificaron que también le pondrían atención a la estudiante Martina Monsalves ya que, a pesar de haber intentado resolver varios problemas, no había completado ninguno y podría tener altos niveles de frustración.
- Algunas tutoras observaron que si una estudiante deja de participar en el curso, aparecerá en el sistema como una estudiante que va atrasada, pero no debería aparecer.

Luego de identificar a las estudiantes que necesitaran apoyo, debían identificar a las estudiantes que fueran más avanzadas. Como resultado a esta tarea se observó lo siguiente:

- Todas las tutoras identificaron que las estudiantes Antonia Quezada y Antonia Jimenez son las que van más adelantadas, dado que han resuelto correctamente la mayoría de los problemas.
- Una tutora identificó que Rosario González también va rápido, ya que a pesar de tener un problema incompleto, ha resuelto varios problemas y lo está intentando.

Además de las observaciones anteriores, se identificaron algunos problemas de usabilidad en la vista de la tabla. El primer problema es que no se ven todas las estudiantes al mismo

tiempo, sino que se dividen en páginas, lo cual generó ruido en las tutoras. Por otro lado, dos tutoras buscaron al final de la tabla información sobre la cantidad de estudiantes que resolvieron un problema, o sobre la cantidad de problemas que resolvió cada estudiante. Por último, dos tutoras intentaron hacer click sobre el nombre de un problema con la intención de acceder a la información de ese problema.

5.3.2.2. Outputs alternativos y sugerencias

Para evaluar los outputs alternativos y las sugerencias, la tutora tenía que acceder al curso básico e ingresar al problema "Tipos de triángulos", para luego ver los casos de prueba del problema. En la Figura 5.5 se encuentra la vista a la que accedía la tutora. Se le dio un tiempo para explorar la plataforma y luego se le asignaron 2 tareas. La primera fue responder a la pregunta: "¿Detectas casos de prueba donde se puedan agregar sugerencias?, y la segunda tarea fue agregar una sugerencia a un *output recibido*.

Figura 5.5: Vista de los casos de prueba del problema Tipos de triangulo para prueba de usabilidad con tutoras.

Como se ve en la Figura 5.5 todos los casos tienen una notificación de "nuevos outputs" lo cual significa que hubo soluciones de estudiantes que no dieron el *output esperado* para ese caso y se les puede asignar una nueva sugerencia. Dado que las tutoras no conocían el concepto de sugerencias o *output alternativos*, les tomó un tiempo poder entender a qué se refería la pregunta y poder responder.

Al inicio de las pruebas se le dijo a cada tutora lo siguiente sobre los *outputs alternativos*: "para los casos que no den el *output esperado*, las tutoras podrán identificar patrones

que generan estos errores y sugerirles a las estudiantes como mejorar". Sabiendo esto, e interactuando con la plataforma, cada tutora dedujo lo que son las sugerencias y para qué sirven. Al hacer la revisión de los vídeos de las pruebas con tutoras se midió el tiempo que les tomó entender esta parte de la plataforma y la tutora que más demoró, le tomó menos de 3 minutos, y el resto de las tutoras demoraron entre uno y dos minutos. Esta demora es razonable debido a que el sistema está incorporando un nuevo concepto y las tutoras necesitan aprenderlo antes de usar la plataforma sin problemas. La memorista observó que luego de que las tutoras comprendían el concepto de *outputs alternativos* y sugerencias, estas no tenían problemas para desenvolverse en la plataforma y la mayoría quiso agregar sus propias sugerencias.

Como segunda tarea para probar las sugerencias y los *outputs alternativos* las tutoras debían entrar en la vista de la Figura 5.6 y agregar una sugerencia, entregada por la memorista, al primer *output recibido*. Todas las tutoras pudieron completar esta tarea sin problemas, sin embargo la memorista observó que todas volvieron al modal de *outputs alternativos* para confirmar si se había agregado su sugerencia. Esto se debe a que no se incorporó en la plataforma ningún tipo de confirmación después de agregar una sugerencia. Es importante considerar esto en una futura versión de la plataforma, debido a que afecta la seguridad que sienten las usuarias al utilizarla.

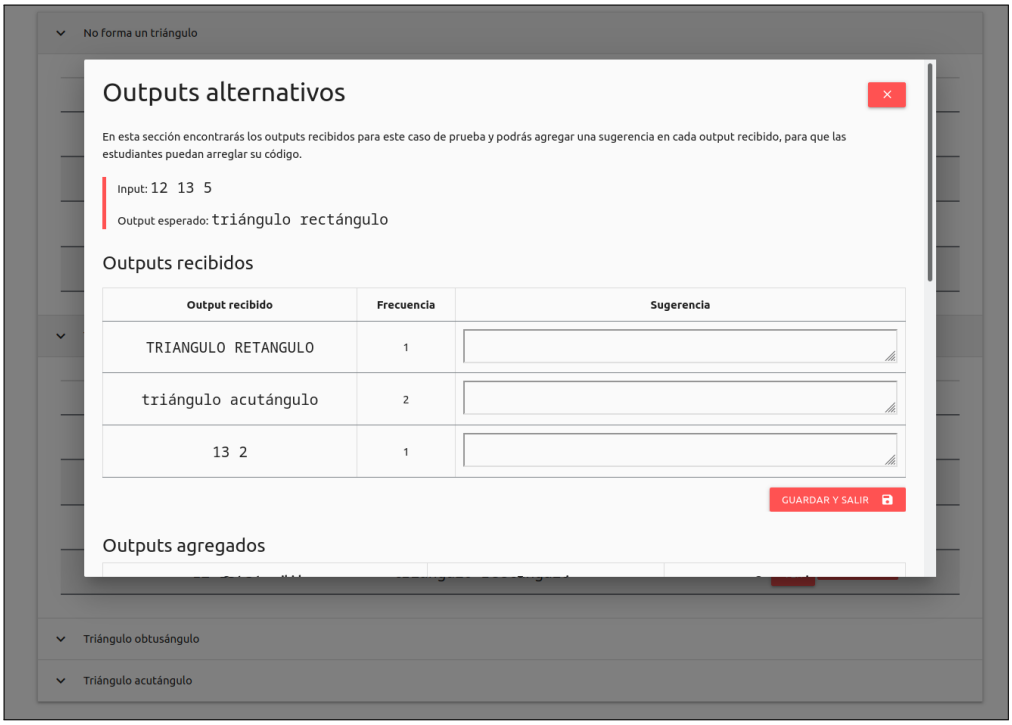


Figura 5.6: Vista de los *outputs sugeridos* del problema Tipos de triangulo para prueba de usabilidad con tutoras.

En general hubo un buen recibimiento del concepto de *output alternativo* y sugerencia, y esto se concluye debido a los siguientes comentarios textuales de dos tutoras:

> "Al principio cuesta cachar que los outputs recibidos son los outputs de las niñas pero se entiende fácil después. "

> "Súper útil. para ver la frecuencia de los errores, para ver qué tipo de errores están teniendo."

A pesar del buen recibimiento, se identificaron algunas funcionalidades que serían útiles para mejorar la experiencia de las tutoras con el sistema, estas se listan a continuación:

- Agregar *outputs alternativos* manualmente.
- Descartar *outputs recibidos*.
- Saber qué niñas han generado el *output recibido* y ver su código.
- Notificar a las estudiantes al crear una sugerencia.

5.3.2.3. Creación de problemas

En la Figura 5.7 se encuentra la vista de las clases del curso al que tenían acceso las tutoras, donde se encuentra la clase de *ciclos while* sin ningún problema aún. Para evaluar la usabilidad de la creación de nuevos problemas en la plataforma, se le entregó a cada tutora el enunciado de un problema en formato `pdf` y un archivo de casos de prueba en formato `json`, con esto la tutora debía agregar el problema al sistema en la clase de *ciclos while*. Para completar esta tarea, cada tutora debía hacer click en el símbolo + de la clase llamada *ciclos while* para acceder a la vista de creación de problemas (Figura 5.8) y crear el problema.

Figura 5.7: Vista de clases y problemas para prueba de usabilidad con tutoras.

La memorista observó que todas las tutoras supieron que debían hacer click sobre el símbolo + que se encuentra junto a la clase de *ciclos while* para agregar un problema. Estas lograron completar la tarea y demoraron entre 40 segundos y 2 minutos. En general las que más demoraron fue porque no tenían los archivos descargados previamente. Al igual que en las tareas anteriores, el tiempo que demoraron en completarla es bajo, dado que era la primera vez que interactuaban con esta parte de la plataforma.

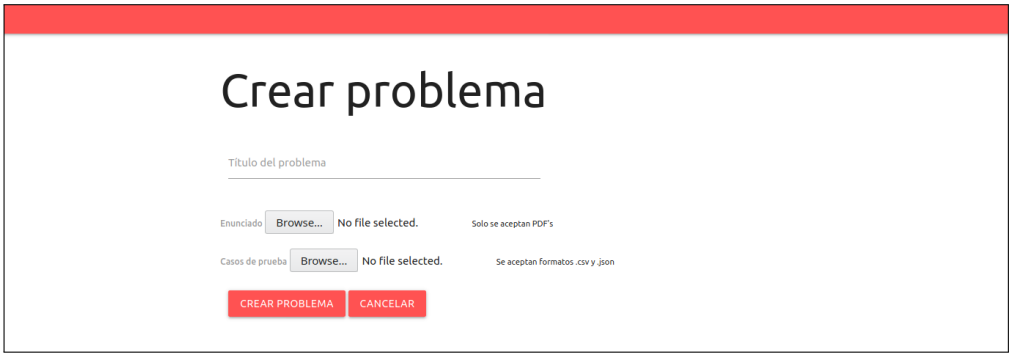


Figura 5.8: Vista para agregar problema.

A pesar que las tutoras pudieron acceder a la vista de la creación de problemas fácilmente, tres de las cinco tutoras no le pusieron título al problema y solo agregaron los archivos. Este error no tuvo consecuencias negativas, ya que el sistema les indicó que debían poner un título, pero para una siguiente versión del sistema se debe trabajar en que los campos obligatorios de un formulario sean más claros.

Luego de completar la tarea, las tutoras se encontraban en la vista del curso (Figura 5.7) y se les preguntó lo siguiente: "¿Qué te parece esta vista y el proceso de subir un problema? De esta pregunta se obtuvieron múltiples observaciones que se listan a continuación:

- La mayoría consideró que la creación de problemas es fácil, intuitiva y simple.
- Dos tutoras observaron que no era posible editar ni eliminar un problema. Esto no se implementó debido a que esta memoria es un piloto, pero para la siguiente versión de la plataforma si existirá esta funcionalidad.
- El símbolo para editar una clase no es intuitivo, ya que el cursor no toma la forma de un elemento *clickeable* al pasar por ahí.
- Ordenar las clases desde la primera hasta la última, puede resultar incómodo cuando hay muchas clases.

5.3.2.4. Resultados cuestionario SUS

Cuando la tutora completó las tareas asignadas se le entregó el cuestionario SUS. En la Tabla 5.7 están los puntajes calculados para el cuestionario SUS respondido por cada tutora. En la tabla se ve que todos los cuestionarios entregaron un resultado mayor o igual a 80 puntos (de un total de 100). Como se explicó anteriormente, este puntaje está sobre el promedio del estudio hecho por Jeff Sauro en [11] y en entre el 10 % de los sistemas con mejor nota, es decir tiene un buen nivel de usabilidad.

	T1	T2	T3	T4	T5
Total	95	90	82.5	80	97.5

Tabla 5.7: Resultados cuestionario SUS con tutoras.

Las preguntas del cuestionario que tuvieron peor resultado fueron *Creo que necesitaría*

ayuda para poder usar el sistema y *Las personas aprenderían rápidamente cómo usar el sistema*. Esto tiene sentido ya que el sistema introduce conceptos que no se han utilizado antes en la organización, entonces las tutoras notaron que su uso requiere aprendizaje. A pesar de esto todas las tutoras comprendieron el funcionamiento del sistema rápidamente y evaluaron bien los otros aspectos del cuestionario.

5.3.3. Análisis de la validación

De la evaluación de la tabla de avance se observó que las tutoras son capaces de extraer información útil sobre las estudiantes y su avance en el curso en un tiempo razonable gracias a la tabla de avance. Comparando esta acción con las formas en que se hacía seguimiento antes, esta es más rápida ya que la tabla se genera automáticamente. Por otro lado, se observó que las tutoras creen que las estudiantes que más necesitan apoyo son aquellas que no han subido ninguna solución, sin embargo hay muchos factores que pueden influir en esto; puede ser que ya no se encuentre activa en el curso, que no entienda cómo funciona la plataforma o que no se atreva a subir una solución por creer que fallará. Dado esto, para futuras versiones del sistema, se evaluarán formas de distinguir los factores que generan que una estudiante no envíe soluciones al sistema y así darles apoyo personalizado.

Respecto a los *outputs alternativos* y las sugerencias, las tutoras lograron entender el concepto rápidamente y utilizar la plataforma libremente. Esto refleja que a pesar de que la plataforma introduce un nuevo concepto, el diseño de las interfaces permite que las usuarias aprendan rápidamente a utilizarla.

Por otro lado, el proceso de creación de problemas es fundamental para este sistema ya que estos son la base para la funcionalidad del módulo de feedback y el módulo de avance. Para las participantes de la validación, este proceso fue fácil e intuitivo, lo cual da una buena evaluación del sistema.

Finalmente, del cuestionario SUS se desprende que el sistema tiene un nivel de usabilidad alto, y complementado con el análisis realizado anteriormente se confirma este resultado. También se concluye que las historias de usuario **HD1**, **HD2** y **HD3** están cubiertas por el sistema. A pesar de esto hay algunos elementos que habrá que integrar en la plataforma, como mensajes de confirmación en ciertas acciones o la posibilidad de editar elementos en caso de cometer un error, para asegurar que las tutoras tengan una buena experiencia. Estos elementos serán incluidos en la próxima versión del sistema, además de las funcionalidades que se propusieron en las secciones anteriores.

5.4. Resumen

En el presente capítulo se presentaron las validaciones de la plataforma generada en esta memoria. La primera validación fue un caso de estudio exploratorio con estudiantes del Curso Anual, donde el propósito fue entender si la información entregada por el sistema, al dar feedback a las estudiantes, es útil para resolver los errores que tienen sus soluciones.

De este se concluye que las estudiantes logran identificar los errores de su código con las sugerencias y casos de prueba, pero en algunos casos esto no basta para arreglar su código, por lo tanto se deberá evaluar cómo entregar las sugerencias de forma que sean útiles para que las estudiantes puedan corregir sus errores.

La segunda evaluación fue dirigida a las tutoras, donde se realizaron dos validaciones de usabilidad de los cuatro módulos implementados. Las tutoras indicaron que la plataforma les parecía simple y fácil de utilizar, y se observó que estas fueron capaces de extraer de cada uno de los módulos información útil para hacer seguimiento del curso y sus estudiantes. Sin embargo se encontraron algunos problemas de usabilidad que deberán ser mejorados antes de utilizar oficialmente la plataforma.

Capítulo 6

Conclusión

En esta memoria se extendió un sistema de revisión de código como de apoyo al Curso Anual de Niñas Pro, donde las estudiantes del curso pueden obtener feedback de sus soluciones a los problemas de cada clase; y donde las tutoras pueden ver los resultados de las estudiantes para darles sugerencias de cómo mejorar, y hacer seguimiento de su avance y asistencia. La creación de este sistema era importante ya que las herramientas de feedback utilizadas actualmente no están pensadas para uso pedagógico, sino que competitivo, generando altos niveles de deserción en el curso.

Para el logro de los objetivos planteados, primero se evaluaron dos herramientas de generación de feedback para elegir cual extender en este proyecto. Luego, se diseñó un módulo de manejo de cursos, un módulo para dar feedback a las estudiantes, un módulo para hacer seguimiento y un módulo para llevar asistencia. Una vez diseñados se implementaron los 4 módulos para luego validar la usabilidad y utilidad del sistema con usuarias reales.

Gracias al nuevo sistema, desarrollado en el marco de la presente memoria, las estudiantes obtendrán feedback con más información que la que obtienen hoy en día con los jueces en línea. Por su parte, las tutoras tendrán a disposición los errores que cometen las estudiantes para añadir sugerencias al sistema. Por otro lado, gracias a los módulos de asistencia y seguimiento, las tutoras tendrán información fidedigna del avance y participación de las estudiantes de los cursos.

6.1. Conclusiones del trabajo

Hacer validación de usabilidad y utilidad de la plataforma fue fundamental para identificar elementos a mejorar en futuras versiones del sistema. Por otro lado, hacer un caso de estudio exploratorio permitió ver cómo las estudiantes interactúan con la plataforma, para validar que el feedback entregado sería un aporte para ellas. Gracias a las instancias de validación se confirma que es necesario seguir adelante con este proyecto para mejorar la forma en que se enseña programación, y generar instancias de educación más personalizadas para disminuir la deserción en cursos introductorios de programación.

Durante la presente memoria quien escribe este documento tuvo la oportunidad de guiar la práctica profesional de tres estudiantes de Ingeniería en Computación, de la cual se aprendió que la comunicación dentro de un equipo de trabajo y la buena relación entre sus integrantes es fundamental, para que todas y todos tengan la confianza de pedir ayuda en caso de necesitarla, y para potenciar las cualidades de cada integrante para lograr un mejor producto. Por otro lado, se aprendió que el uso de metodologías ágiles como "pair programming", el uso de "kanban" y los "Daily meetings", mejoran la forma de trabajar del equipo. Al comparar el tiempo que la memorista desarrolló el proyecto sola, en relación al tiempo con el equipo de práctica, se concluye que el trabajo con un equipo es mejor ya que aporta diferentes perspectivas a la hora de resolver problemas, generando mejores soluciones.

6.2. Trabajo futuro

Considerando los resultados obtenidos en la validación y teniendo en cuenta que el sistema implementado es un prototipo, a continuación plantea la proyección que tiene este trabajo de memoria, tanto en el corto, como en el largo plazo.

6.2.1. Corto plazo

Para que el sistema generado en la presente memoria sea utilizado oficialmente en el Curso Anual de Niñas Pro, se deberán resolver las fallas de usabilidad encontradas en las pruebas con usuarias, de manera que estas tengan una buena experiencia con la plataforma, y que sea utilizada cómodamente. Por otro lado, hay que agregar algunas funcionalidades mínimas para que cada sede administre sus cursos de forma independiente, sin necesitar acudir a quien administre el sistema; algunas de estas son la existencia de una usuaria administradora, carga masiva de usuarias y edición o eliminación de problemas.

Otra funcionalidad que es importante mejorar es que al subir un archivo de casos de prueba inválido, el sistema no notifica a la usuaria que este tiene un error, por lo tanto, se crean problemas sin casos de prueba. Es indispensable reparar esta funcionalidad y que la plataforma ayude a las tutoras a crear archivos con casos de prueba válidos, ya que es fácil cometer errores al crear casos de prueba y las profesoras no deberían verse afectadas por esto.

Por otro lado, es necesario someter a pruebas de carga y concurrencia al módulo que corregirá ejercicios. Esto se debe hacer para saber si el sistema podrá soportar que varias estudiantes envíen una solución a la vez y asegurar que se genere bien el feedback al recibir varias soluciones. Esto se deberá hacer antes de poner la plataforma a disposición de las estudiantes y tutoras.

Finalmente, queda pendiente hacer pruebas de usabilidad y utilidad con usuarias de otras sedes de Niñas Pro. Esto es relevante para tener diversas perspectivas de la plataforma y así tener un sistema inclusivo.

6.2.2. Largo plazo

En el largo plazo se deberá evaluar los efectos de la plataforma en el curso. Se podría evaluar los niveles de deserción en los cursos de las diferentes sedes, antes y después de la plataforma. Por otro lado, se puede evaluar nuevamente cómo interactúan las estudiantes con la plataforma, una vez que ya están acostumbradas a ella, para saber si las sugerencias les son útiles, las entienden y evaluar si hay sugerencias que les sean más fáciles de entender que otras. En esta misma línea, se podría implementar la herramienta en cursos mixtos que tengan un objetivo parecido al Curso Anual de Niñas Pro, y observar cómo interactúan hombres y mujeres con la plataforma.

Por el lado de las tutoras, se puede evaluar si efectivamente la plataforma les permite mejorar el seguimiento que le hacen a las estudiantes de sus cursos. Estos estudios deberían realizarse de forma multidisciplinar considerando profesionales de áreas como la pedagogía, ciencias sociales y ciencias de la computación.

Otra línea de investigación que se desprende de este trabajo es el nivel de frustración que muestran las estudiantes al resolver problemas de este tipo. En este sentido se propone evaluar su experiencia con los jueces en línea que se utilizan actualmente en el curso, y luego ver cómo cambian los niveles de frustración y su tolerancia a la frustración, al utilizar el nuevo sistema.

Otra arista de trabajo a largo plazo es la implementación de nuevas visualizaciones que permitan hacer seguimiento. Como se mencionó en el informe, se pueden agregar variables como el tiempo que interactúan las estudiantes con la plataforma, la asistencia y la cantidad de intentos para resolver un ejercicio.

Por otro lado, para mejorar el feedback entregado por la plataforma se podría estudiar cómo automatizar el feedback relacionado a errores de presentación. Así, si el algoritmo de una estudiante calcula bien el resultado a un problema pero imprime algún símbolo de más, se podrían generar sugerencias automatizadas que la guíen a presentar el resultado como se espera.

Bibliografía

[1] Luciana Benotti, Federico Aloi, Franco Bulgarelli, and Marcos J Gomez. The Effect of a Web-based Coding Tool with Automatic Feedback on Students' Performance and Perceptions. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pages 2–7, 2018.

[2] John Brooke. SUS: a "quick and dirty" usability scale. *Usability evaluation in industry*, page 189, 1996.

[3] Olimpiada Chilena de Informática. `https://www.olimpiada-informatica.cl/acercade`, 2018. En línea; último acceso 16 de agosto 2020.

[4] Soumitra Dutta, Bruno Lanvin, and Silja Baller. The global information technology report 2015. In *World Economic Forum*. Citeseer, 2016.

[5] World Economic Forum. The global gender gap report. World Economic Forum Genebra, 2020.

[6] Sumit Gulwani, Ivan Radiček, and Florian Zuleger. Automated clustering and program repair for introductory programming assignments. *ACM SIGPLAN Notices*, 53(4):465–480, 2018.

[7] Nicolás Jara. Programadores: Escasez en Chile y en el mundo. `https://crealab.cl/programadores-escasez-chile-y-el-mundo/`. En línea; último acceso 6 julio 2020.

[8] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. A systematic literature review of automated feedback generation for programming exercises. *ACM Trans. Comput. Educ.*, 19(1), September 2018.

[9] Comunidad Mujer. Mujer y trabajo: Brecha de género en STEM, la ausencia de mujeres en Ingeniería y Matemáticas, 2017. En línea; último acceso 16 de agosto 2020.

[10] R.S. Pressman, J.E.M. Murrieta, V.C. Olguín, and E.P. Rojas. *Ingeniería del software: un enfoque práctico*. McGraw-Hill, 2006.

[11] Jeff Sauro. Measuring Usability with the System Usability Scale (SUS). `https://measuringu.com/sus/`. En línea; último acceso 21 de julio 2020.

[12] Rishabh Singh, Sumit Gulwani, and Armando Solar-Lezama. Automated feedback ge-

neration for introductory programming assignments. In *Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation*, pages 15–26, 2013.

[13] Carl W Turner, James R Lewis, and Jakob Nielsen. Determining Usability Test Sample Size. In *International Encyclopedia of Ergonomics and Human Factors*, pages 3084–3088, 2006.

[14] Ke Wang, Rishabh Singh, and Zhendong Su. Search, align, and repair: data-driven feedback generation for introductory programming exercises. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 481–495, 2018.

[15] Jooyong Yi, Umair Z Ahmed, Amey Karkare, Shin Hwei Tan, and Abhik Roychoudhury. A feasibility study of using automated program repair for introductory programming assignments. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pages 740–751, 2017.

[16] R.K. Yin. *Case Study Research: Design and Methods*. SAGE Publications, 2013.

Apéndice A

Anexos

A. Problemas utilizados en caso de estudio exploratorio

A.1. Problema Bisiesto

A continuación se presenta el enunciado del problema bisiesto y los casos de prueba de este problema en el caso de estudio exploratorio.

Enunciado

Un año es bisiesto si es múltiplo de 4 y no es múltiplo de 100. Además, si un número es múltiplo de 400 también es bisiesto. Crea un programa que reciba un año y que diga si el año es bisiesto o no.

Input El input contiene un entero que representa un año.

Output El programa deberá entregar "Si" si el año es bisiesto, y "No" si el año no es bisiesto.

* Recuerda agregar un salto de línea al final de cada línea.

Ejemplos En la Figura A.1 se encuentran ejemplos de entrada y salida para el problema Bisiesto.

Casos de prueba

En el Código A.1 se encuentran los casos de prueba del problema Bisiesto, utilizados en el caso de estudio exploratorio.

```
[{
```

Input	Output
1988	Si
2100	No
2020	Si
2021	No

Figura A.1: Ejemplos de entrada y salida para el problema Bisiesto.

```
    "Input":"2020",
    "Output": "Si\n",
    "Categoria": "Anio multiplo de 4"},
{
    "Input":"1988",
    "Output": "Si\n",
    "Categoria": "Anio multiplo de 4"},
{
    "Input":"2004",
    "Output": "Si\n",
    "Categoria": "Anio multiplo de 4"},
{
    "Input":"1992",
    "Output": "Si\n",
    "Categoria": "Anio multiplo de 4"},
{
    "Input":"2100",
    "Output": "No\n",
    "Categoria": "Anio multiplo de 4 y de 100"},
{
    "Input":"1800",
    "Output": "No\n",
    "Categoria": "Anio multiplo de 4 y de 100"},

{
    "Input":"2000",
    "Output": "Si\n",
    "Categoria": "Anio multiplo de 400"},

{
    "Input":"1995",
    "Output": "No\n",
    "Categoria": "Anio no es multiplo de 4 ni de 400"},
{
    "Input":"2001",
    "Output": "No\n",
```

```
    "Categoria": "Anio no es multiplo de 4 ni de 400"},
]
```

Código A.1: Ejemplos de casos de prueba para problema Bisiesto.

A.2. Problema Robot

A continuación se presenta el enunciado del problema bisiesto y los casos de prueba de este problema en el caso de estudio exploratorio.

Enunciado

UR2277 es un robot que quiere hacer ejercicio. Ha visto que muchos humanos ocupan un "timer" que les dice cuanto tiempo queda y al final emite un sonido ("BIIIIIIP"), indicando que terminó el ejercicio. Como R2277 es un robot, en un segundo puede hacer muchas más repeticiones que un humano normal, por lo que no le sirve un "timer" normal, pero se dio cuenta que el tiempo que se demora un computador en escribir números es el ideal para él. R227 te dirá cuántos ejercicios quiere hacer y el tiempo que quiere realizar cada ejercicio. Tu programa debe imprimir la cuenta regresiva para cada ejercicio, si el ejercicio no es el último, al finalizar se debe imprimir "Siguiente ejercicio", si el ejercicio es el último se debe imprimir "BIIIIIIP".

Input Se entregará un entero N>0, que dirá cuántos ejercicios hará el robot y luego se entregarán N números positivos que indicarán el tiempo que hará cada ejercicio.

Ejemplos En la Figura A.2 se encuentran ejemplos de entrada y salida para el problema Bisiesto.

Casos de prueba

En el Código A.2 se encuentran los casos de prueba del problema Bisiesto, utilizados en el caso de estudio exploratorio.

```
[{
    "Input":"1 3",
    "Output": "3\n2\n1\nBIIIIIIP\n",
    "Categoria": "Solo un ejercicio"},
{
    "Input":"1 10",
    "Output": "10\n9\n8\n7\n6\n5\n4\n3\n2\n1\nBIIIIIIP\n",
    "Categoria": "Solo un ejercicio"},
{
    "Input":"2 3 3",
```

Entrada	Salida
3 3 2 1	3 2 1 Siguiente ejercicio 2 1 Siguiente ejercicio 1 BIIIIIIP
1 3	3 2 1 BIIIIIIP
3 1 2 5	1 Siguiente ejercicio 2 1 Siguiente ejercicio 5 4 3 2 1

Figura A.2: Ejemplos de entrada y salida para el problema Robot.

```
    "Output": "3\n2\n1\nSiguiente ejercicio\n3\n2\n1\nBIIIIIIP\
      n", "Categoria": "2 ejercicios"},
{
    "Input":"2 3 5",
    "Output": "3\n2\n1\nSiguiente ejercicio\n5\n4\n3\n2\n1\
      nBIIIIIIP\n", "Categoria": "2 ejercicios"},
{
    "Input":"5 3 5 10 2 1",
    "Output": "3\n2\n1\nSiguiente ejercicio\n5\n4\n3\n2\n1\
      nSiguiente ejercicio\n10\n9\n8\n7\n6\n5\n4\n3\n2\n1\
      nSiguiente ejercicio\n2\n1\nSiguiente ejercicio\n1\
      nBIIIIIIP\n",
    "Categoria": "Mas de 2 ejercicios"},
{
    "Input":"10 3 5 10 2 1 3 5 10 2 1",
    "Output": "3\n2\n1\nSiguiente ejercicio\n5\n4\n3\n2\n1\
      nSiguiente ejercicio\n10\n9\n8\n7\n6\n5\n4\n3\n2\n1\
      nSiguiente ejercicio\n2\n1\nSiguiente ejercicio\n1\
      nSiguiente ejercicio\n3\n2\n1\nSiguiente ejercicio\n5\n4\
      n3\n2\n1\nSiguiente ejercicio\n10\n9\n8\n7\n6\n5\n4\n3\n2
```

```
        \n1\nSiguiente ejercicio\n2\n1\nSiguiente ejercicio\n1\
        nBIIIIIIP\n",
    "Categoria": "Mas de 2 ejercicios"}
]
```

Código A.2: Ejemplos de casos de prueba para problema Robot.