Aprender a Programar: Role Playing Learn

Autores:

93419 - Alonso, Juan Manuel
juan6691@gmail.com
93764 - Desseno, Carlos
cdesseno@gmail.com
93479 - Lew, Kevin Martin
kevin.martin.lew@gmail.com
Fecha de entrega: 21/07/2017

Tutor: Prof. Dr. Mariano Méndez

"Facultad de Ingeniería - Universidad de Buenos Aires" 2017

Índice general

I Informe **5**

1. General **6**
1.1. Introducción 6
1.2. Motivación 7
1.3. Objetivo 8
1.3.1. Objetivos adicionales 9

2. Trabajos Relacionados **10**
2.1. Moodle 10
2.2. HackerRank 11
2.3. Codewars 12
2.4. Conclusión 13

3. Trabajo **14**
3.1. Descripción de la Plataforma 14
3.1.1. Uso de la plataforma 15
3.2. Metodología y Proceso de Desarrollo de Software 23
3.2.1. Horas reales vs. horas estimadas 24
3.3. Requerimientos 26
3.3.1. Requerimientos funcionales 26
3.3.2. Requerimientos no funcionales 28
3.4. Descripción de la Arquitectura 28
3.4.1. Componentes de la arquitectura 29
3.4.2. Ejecución de una actividad 30
3.4.3. Tecnologías utilizadas 33
3.5. Diagramas 35
3.5.1. Esquema de la base de datos 35
3.5.2. Diagrama de las clases 36
3.6. Casos de Uso 37
3.6.1. Inscripción en un curso 37
3.6.2. Aceptación de un alumno en un curso 37
3.6.3. Creación de una actividad 38
3.6.4. Envío de la solución de una actividad 38

3.7. Extensibilidad de la Plataforma 38
3.7.1. Front-end 38
3.7.2. Back-end 41

4. Caso de Estudio **43**
4.1. Contexto 43
4.2. Servidor 43
4.3. Participación de la Cátedra 44
4.4. Participación de los Alumnos 44
4.4.1. Patrón detectado 44
4.4.2. Relación entre la participación y la nota 49
4.5. Personalización del Curso 51
4.6. Recepción de la Plataforma 51

5. Conclusiones **53**
5.1. Trabajos Futuros 54

6. Anexo **57**
6.1. API 57
6.1.1. Autenticación 57
6.1.2. API 57

II Manual Técnico **66**

7. Instalación y Despligue **67**
7.1. Introducción 67
7.2. Dependencias Necesarias 67
7.3. Generación de Ejecutables y Archivos Extras 69
7.4. Despliegue de la Aplicación 70

III Manual de Usuario **72**

8. Funcionalidades **73**
8.1. Introducción 73
8.2. Inicio de Sesión 73
8.3. Registro 73
8.4. Mi Perfil 74
8.5. Cerrar Sesión 74
8.6. Como Administrador 75
8.6.1. Administrar cursos 75
8.6.2. Usuarios 76
8.6.3. Copiar actividades 78
8.7. Como Profesor o Ayudante de un Curso 78

8.7.1. Actividades 79
8.7.2. Categorías 81
8.7.3. Alumnos 82
8.7.4. Personalización 83
8.7.5. Reportes 84
8.8. Como Alumno 89
8.8.1. Inscripción en un curso 89
8.8.2. Dentro de un curso 89
8.8.3. Resolución de una actividad 90

9. Ejemplos de Actividades **92**
9.1. Introducción 92
9.2. C 92
9.2.1. Pruebas 92
9.2.2. Entrada/Salida 93
9.3. Java 94
9.3.1. Pruebas 94
9.3.2. Entrada/Salida 95
9.4. Python 96
9.4.1. Pruebas 96
9.4.2. Entrada/Salida 97

Parte I

Informe

Capítulo 1

General

1.1. Introducción

La plataforma desarrollada tiene como fin mejorar la interacción entre los alumnos y los profesores durante los cursos introductorios de programación de cualquier carrera afín a la informática. Los profesores a través de esta herramienta pueden realizar un seguimiento y obtener estadísticas de las actividades de programación asignadas a sus alumnos. Por su parte, los alumnos tienen la posibilidad de interactuar mediante la integración de un marco de trabajo para la solución de problemas exclusivos de programación. A medida de que el alumno logra la resolución de dichos problemas, el conocimiento adquirido se ve materializado a través de puntos que el docente puede utilizar en la calificación final del curso. Asimismo, la herramienta colabora con el docente permitiendo verificar la correcta resolución de las actividades mediante la ejecución de pruebas automatizadas realizadas por el docente.

Una de las ideas principales en juego es incentivar a los alumnos para que realicen las actividades y se mantengan al día con los contenidos de la materia. En este punto es donde toman importancia los juegos de Rol o Role-Playing Games. Un juego de rol se trata de un conjunto de jugadores que deben actuar, jugar o desempeñar un determinado rol o papel, en otras palabras, es un juego interpretativo-narrativo en el que los jugadores asumen el "rol" de personajes imaginarios a lo largo de una historia o trama en la que interpretan sus diálogos y describen sus acciones. Uno de los juegos de rol más conocidos fue creado en 1974 por Gary Gygax y es conocido como "Dungeons and Dragons" (Calabozos y Dragones), que utiliza únicamente papel, lápiz, dados y la imaginación del grupo de personajes. Dentro del grupo de personajes se encuentra el director del juego que en este caso sería el docente, quien articula, fija las reglas y la narrativa del juego.

Esta aplicación puede integrar un conjunto de Role-Playing Games determinados por los profesores de cada curso en el cual se desarrollen las

actividades de enseñanza. Uno de los puntos más importantes de un juego de Rol es que permite a sus jugadores acumular Puntos de Experiencia o XP o Karma Points; que conceptualmente es una puntuación acumulativa según el esclarecimiento y/o participación y/o resolución de una actividad planteada.

El resultado del trabajo práctico, es decir, la plataforma desarrollada, fue puesto en práctica en el curso cuatro de la cátedra de "Algoritmos y Programación I" de la Facultad de Ingeniería, Universidad de Buenos Aires durante el primer cuatrimestre de 2017. Se contó con la participación tanto de los profesores como de los ayudantes, quienes fueron brindando sugerencias y mejoras para la plataforma.

1.2. Motivación

La importancia de la programación como competencia dentro de una carrera de informática es fundamental. También es fundamental la forma en que se evalúa dicha competencia.

Dentro de la problemática del dictado de un curso de programación se encuentra la forma de incentivar a los alumnos a realizar los ejercicios que se requieren para conseguir las habilidades y competencias que éstos deben adquirir durante el dictado del mismo. La asignación, planificación, control y corrección de los ejercicios asignados a un grupo de alumnos, que puede rondar un promedio de veinte o más personas, es una tarea compleja y que requiere mucha dedicación de tiempo por parte del docente. Esta tarea es aún más compleja si la misma debe ser realizada manualmente.

Hoy en día se encuentran disponibles una serie de plataformas en línea. Por un lado podemos encontrar las plataformas en línea del tipo de *Ambiente Educativo Virtual* que generalmente se usan para la gestión de cursos y aprovechan la metodología de *blended learning*. Por ejemplo, Moodle [12], Gradiance [8] o Piazza [13]. Por otro lado, podemos encontrar un tipo de plataformas pensadas para practicar a través de ejercicios de programación (por ejemplo, HackerRank [9], codewars [5]).

El primer tipo de plataformas presentado, es decir, del tipo de *Ambiente Educativo*, está principalmente orientado a la gestión de cursos o resolución de tareas. En general, cuentan con distintos módulos para hacer tareas, evaluaciones y compartir información. Este tipo de plataforma requiere de una configuración y gestión que se encuentra en manos de terceros y las asignaciones no permiten que el estudiante realice los ejercicios, en el caso de la programación, en la misma plataforma. Son meramente contenedores de información que no proporcionan ninguna forma automatizada de evaluar dichos conocimientos.

El segundo tipo de plataformas tiene un fin muy diferente ya que consiste en realizar ejercicios y competir contra otros participantes. La mayoría de

estos sitios apuntan al armado de competencias para la contratación de personas o al uso por parte de usuarios autodidactas como medio de adquisición de experiencia. Estas plataformas no cuentan con un enfoque académico ya que lo que se evalúa siempre es el resultado.

En resúmen, el problema a abordar es crear una plataforma que permita combinar las cualidades de los dos tipos mencionados previamente: que permita administrar cursos y que también permita realizar ejercicios de programación con un enfoque académico. Todo esto, incentivando al alumno mediante la posibilidad de acumular puntos que le servirán para aprobar la materia, utilizando la metodología de Role-Playing Games.

Teniendo en cuenta la importancias que tienen las TICs (Tecnologías de la Información y la Comunicación) en la vida académica, en este trabajo se busca dar con una solución que ataque los problemas descritos anteriormente, con un enfoque dirigido a la automatización, gestión y evaluación de contenidos necesarios de los alumnos de un curso de programación de distintos niveles. A su vez, se busca la posibilidad de determinar el grado de conocimiento adquirido por parte de los alumnos durante el curso mediante la obtención de datos, normalmente en forma de reportes.

1.3. Objetivo

El objetivo principal de este trabajo es realizar una plataforma en línea para:

- Organizar y gestionar cursos de materias de programación.
- Proporcionar un único recurso integrado, en una sola herramienta para que los alumnos que estén cursando los primeros años de alguna carrera de Informática desarrollen sus actividades.
- Incentivar mediante la utilización de la herramienta la adquisición del conocimiento, habilidades y competencias correspondientes a dicha asignatura.
- Implementar una herramienta que proporcione la dinámica de un Role Playing Game como excusa para el proceso de aprendizaje.
- Desarrollar un sistema automatizado de evaluación de los ejercicios escritos en algún lenguaje de programación mediante la utilización de tests automatizados y edición on-line de los ejercicios propuestos apuntando a la construcción de un "banco de trabajo" en el cual los alumnos deban probar sus habilidades.
- Mantener un historial del desempeño del alumno que pueda ser utilizado por distintos docentes para determinar cuáles son los puntos que dicho alumno necesita mejorar.

1.3.1. Objetivos adicionales

- Facilitar la tarea del docente debido a la corrección automática de las actividades.
- Que los alumnos dispongan de una herramienta para practicar que les brinde un feedback inmediato.
- Realizar un seguimiento de los alumnos para ver el nivel de dedicación con el curso.

Capítulo 2

Trabajos Relacionados

2.1. Moodle

Moodle[12] es un software gratuito y de código abierto que se utiliza actualmente en la Facultad de Ingeniería de la Universidad de Buenos Aires como soporte para los cursos, no sólo del departamento de Computación sino también de otros departamentos.

Si bien dispone de varios módulos que se pueden instalar, *Moodle* fue desarrollado con un fin más general pero sin dejar de lados ciertos principios pedagógicos. Algunos de estos principios fueron la *educación a distancia*, la *clase invertida* o el *aprendizaje semipresencial*.

La *educación a distancia*, como lo dice su nombre, le ofrece a los estudiantes la posibilidad de seguir un curso sin estar físicamente en el lugar. Antiguamente correspondía a la famosa educación vía correo postal.

Por otro lado, se encuentra el término de *clase invertida*. Este tipo de clases se caracteriza con la realización de tareas y actividades antes de la clase. Además de resolver las actividades, los alumnos pueden visualizar clases anteriores en forma de video, por ejemplo, o utilizar cualquier otro tipo de recurso disponible en la plataforma. Luego, en la clase presencial resuelven y plantean sus dudas con los profesores y los demás estudiantes. Se llama *clase invertida* porque se invierte el orden de las clases tradicionales donde el profesor dicta una clase y luego el alumno resuelve las actividades.

Por último, se encuentra el *aprendizaje semipresencial*. En este caso se combinan los cursos en línea con cursos dictados por el profesor de manera presencial. Los cursos pueden ser complementados por actividades en líneas, como en los otros casos.

Figura 2.1: Ejemplo de un curso en la plataforma Moodle.

2.2. HackerRank

HackerRank[9] es un sitio web que presenta un enfoque basado en la programación y la competencia.

Los usuarios resuelven distintos desafíos de programación y obtienen una puntuación de acuerdo al tiempo que insumió la solución y a cuánto se parece a la solución esperada. En base a los puntajes obtenidos por cada usuario, hay un ranking global. También existen rankings particulares para ciertos concursos que se realizan.

HackerRank aprovecha el uso de la técnica llamada Gamificación. Según la Wikipedia:

> *"La ludificación —en ocasiones traducido al español como gamificación, jueguización o juguetización— es el uso de técnicas, elementos y dinámicas propias de los juegos y el ocio en actividades no recreativas con el fin de potenciar la motivación, así como de reforzar la conducta para solucionar un problema, mejorar la productividad, obtener un objetivo, activar el aprendizaje y evaluar a individuos concretos."*

HackerRank también tiene una variante comercial que se le ofrece a las empresas. La solución le permite a cada empresa desarrollar sus propios desafíos de programación, que generalmente son utilizados a la hora de contratar nuevos programadores.

Figura 2.2: Ejemplo de actividad de la plataforma HackerRank.

Problem | Submissions | Leaderboard | Discussions | Editorial

Given an array of N integers, can you find the sum of its elements?

Input Format

The first line contains an integer, N, denoting the size of the array.
The second line contains N space-separated integers representing the array's elements.

Output Format

Print the sum of the array's elements as a single integer.

Sample Input

```
6
1 2 3 4 10 11
```

Sample Output

```
31
```

Explanation

We print the sum of the array's elements, which is: $1 + 2 + 3 + 4 + 10 + 11 = 31$.

Submissions: 465419
Max Score: 10
Difficulty: Easy
Rate This Challenge:
More

Current Buffer (saved locally, editable) C

```c
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <assert.h>
#include <limits.h>
#include <stdbool.h>

int simpleArraySum(int n, int ar_size, int* ar) {
    // Complete this function
}

int main() {
    int n;
    scanf("%i", &n);
    int *ar = malloc(sizeof(int) * n);
    for(int ar_i = 0; ar_i < n; ar_i++){
       scanf("%i",&ar[ar_i]);
    }
    int result = simpleArraySum(n, n, ar);
    printf("%d\n", result);
    return 0;
}
```

Line: 1 Col: 1

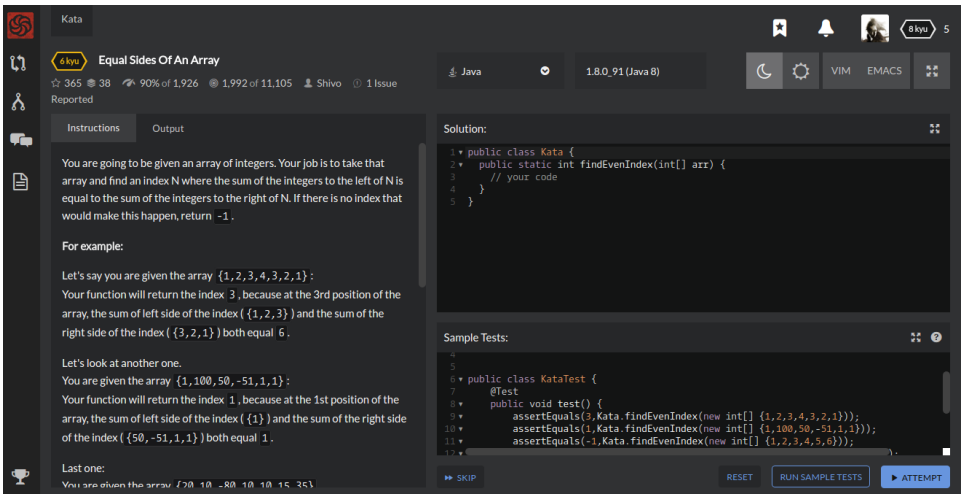
Upload Code as File | Test against custom input | Run Code | Submit Code

2.3. Codewars

Codewars[5] es un sitio muy similar a *HackerRank*. El sitio hace énfasis en resolver desafíos o *katas*, permitiéndole al usuario ganar honores y mejorar en el ranking.

El sitio permite ver las soluciones de otros usuarios y compararlas. La idea principal de esta funcionalidad es comentar las soluciones de otros usuarios con el objetivo de aplicar las mejores prácticas y compartir el conocimiento entre los usuarios de la comunidad.

Figura 2.3: Ejemplo de actividad de la plataforma Codewars.



2.4. Conclusión

De las tres herramientas enumeradas se puede concluir que para el ámbito académico no existe una plataforma que disponga de todas las funcionalidades necesarias para que el alumno tenga un banco de pruebas integrado. Este banco de pruebas que le permite al alumno ejercitar a través de actividades de programación es análogo a un laboratorio para cualquier estudiante de física. En física sería inentendible un enfoque sólo teórico, sin prácticas.

Por otro lado, si bien la implementación de TyCs (Tecnologías de la información y la comunicación) en el ámbito educativo ha avanzado mucho desde sus comienzos y es sabido que ayudan a mejorar el proceso de aprendizaje, no ha explotado todas las posibilidades. No es suficiente con dotar a las salas de informática con equipos, hace falta también contar con recursos de software, por ejemplo, que sirvan de complemento.

La plataforma de banco de pruebas integrado facilita y promueve la gestión de cursos, el aprendizaje a distancia, la clase invertida y los desafíos de programación. Estos conceptos mencionados existen pero no en un sola herramienta y de manera integrada.

Capítulo 3

Trabajo

3.1. Descripción de la Plataforma

La plataforma cuenta con cursos que serán creados por un administrador. Este administrador puede crear un curso definiéndole un nombre y asignándole profesores y ayudantes. Asimismo, puede administrar usuarios.

Un curso corresponde al dictado de una materia particular en un determinado período de tiempo. Los cursos están conformados por alumnos, ayudantes y profesores. Cada uno con distintos roles dentro de la plataforma de aprendizaje y distintos roles dentro de la narrativa del juego.

Un alumno puede inscribirse en un curso donde debe ser aceptado por uno de los profesores del mismo. A su vez, los profesores pueden asignarle a cada alumno un determinado ayudante quien podrá realizar un seguimiento de los mismos. El alumno, una vez aceptado, puede ingresar al curso y ver todas las actividades disponibles agrupadas en categorías. También puede ver el ranking del curso, es decir, una lista de los alumnos del curso ordenada por puntaje de mayor a menor.

El alumno puede resolver las actividades del curso y ver el historial con los envíos realizados y sus resultados. En caso de que el envío de la solución haya sido exitoso puede marcarla como solución definitiva y visualizar las soluciones de los otros alumnos para corroborar y comparar con la suya.

Desde el punto de vista de un profesor, se puede crear y subir una tarea a la plataforma en línea. Subir una actividad consiste en asignarle una cierta cantidad de puntos por su resolución, elegir el lenguaje de programación en el que se va a ejecutar la actividad (la plataforma soporta actividades en C, Python y Java) y, por último, seleccionar cómo se va a comprobar ese resultado: si es a través de pruebas unitarias o verificando el resultado del ejercicio según la entrada/salida de la actividad. Además deberá proporcionar los datos necesarios para que dicha tarea sea solucionada i.e. datos de entrada, datos de salida, pre-condiciones, post-condiciones, enunciado del problema, pruebas unitarias, archivos necesarios, entre otras cosas, según

corresponda.

Las actividades van a estar divididas en categorías que el mismo profesor puede crear dentro de cada curso. Esto le va a permitir más tarde, por ejemplo, relevar los puntos débiles y puntos fuertes de cierto alumno, o ver si el alumno resuelve los ejercicios propuestos en clase. Sin embargo, esta herramienta no ayuda sólo al docente, al alumno también ya que él mismo también puede ver sus puntos.

Desde el punto de vista del ayudante, se pueden crear actividades y categorías al igual que los profesores pero no pueden gestionar los alumnos del curso.

Otro punto a destacar de la plataforma es que permitirá la personalización de cada curso en cuanto a la estética (colores, imágenes, fotos de perfil, etc.) y el punto del alumno según sus puntos acumulados.

Por último, tanto los profesores como los ayudantes tienen la posibilidad de ver ciertos reportes sobre el trabajo realizado de los alumnos.

3.1.1. Uso de la plataforma

Se detallan a continuación algunas de las funcionalidades principales de la plataforma.

Crear un curso

El administrador crea un nuevo curso ingresando el nombre del mismo. Luego puede asignar profesores y ayudantes para dicho curso.

Figura 3.1: Crear curso.

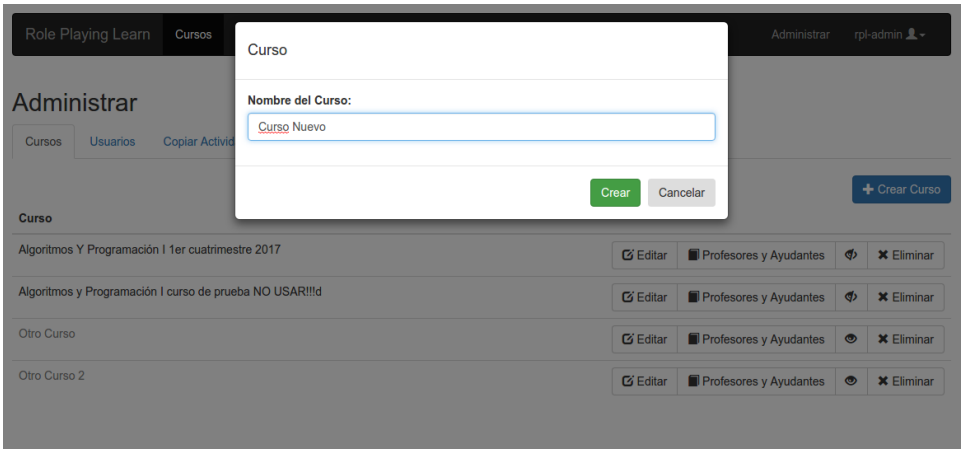
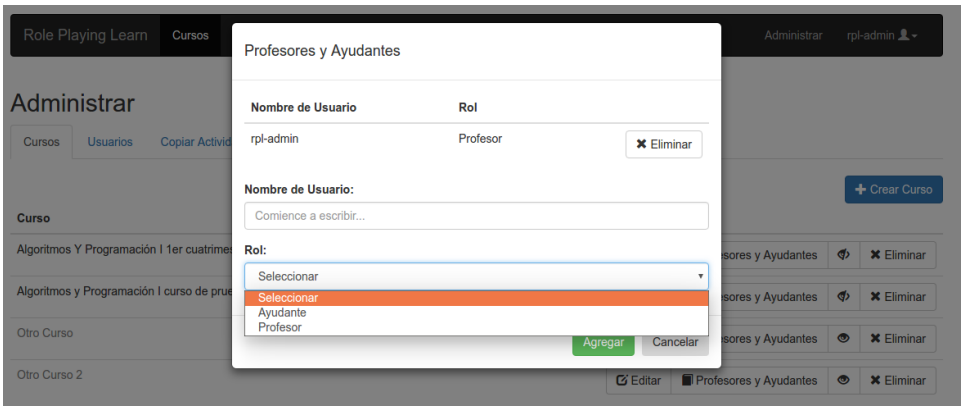


Figura 3.2: Asignar profesores y ayudantes a un curso.

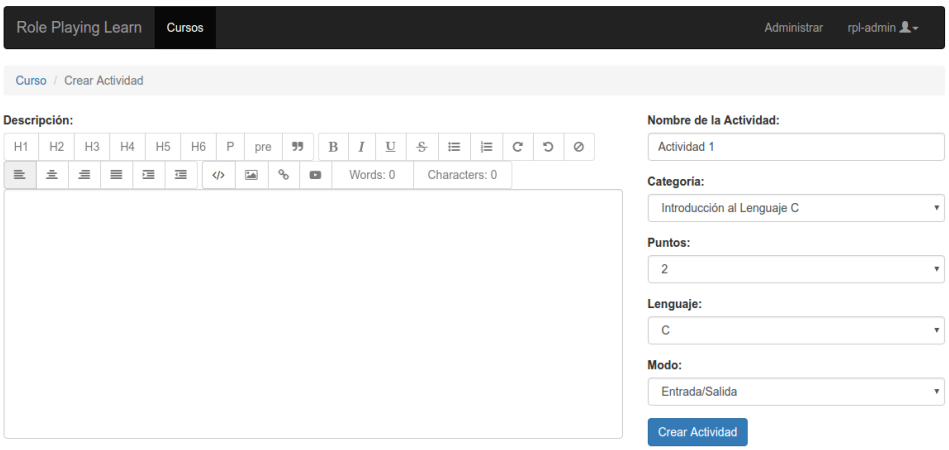


Crear actividad

La creación de la actividad está a cargo de un profesor o ayudante del curso.

El profesor o ayudante debe ingresar la información correspondiente a la descripción de la actividad y el nombre de la misma. Asimismo debe ingresar la cantidad de puntos que otorga si se resuelve correctamente y el lenguaje de programación elegido. También se debe seleccionar la categoría a la que pertenece la actividad. Adicionalmente, debe seleccionar entre los dos modos disponibles para validar la solución del alumno.

Figura 3.3: Información de la actividad y selección de modo.



En el primer modo (entrada/salida) la salida de la solución del alumno se va a comparar con una esperada. En caso de seleccionarse esta opción se debe especificar la entrada que se le va a proporcionar al programa (*standard input*) y la salida esperada (por *standard output*).

Figura 3.4: Información necesaria en el modo entrada/salida.

Template

```
1  #include <stdio.h>
2  int main() {
3      // Codigo del alumno
4      return 0;
5  }
```

Entrada/Salida

Entrada:

```
2
3
```

Salida:

```
5
```

El segundo modo es el que ejecuta prueba unitarias. En este modo el profesor o ayudante debe escribir las pruebas unitarias que se deben correr en cada caso.

La actividad una vez creada ya está disponible para ser resuelta por los alumnos. Luego de la creación existe la opción de deshabilitarla si se quiere que temporalmente no sea visible y la opción de subir archivos adicionales que se crean necesarios para la resolución de la actividad.

Figura 3.5: Información necesaria en el modo pruebas.

Template

```
1  #include <stdio.h>
2  int suma(int numero1, int numero2) {
3      // Codigo del alumno
4      return 0;
5  }
```

Pruebas

```
1  #include <criterion/criterion.h>
2  #include "solution.c"
3
4  Test(misc, test1) {
5      cr_assert(suma(2,3) == 5);
6  }
7  Test(misc, test2) {
8      cr_assert(suma(10,15) == 25);
9  }
```

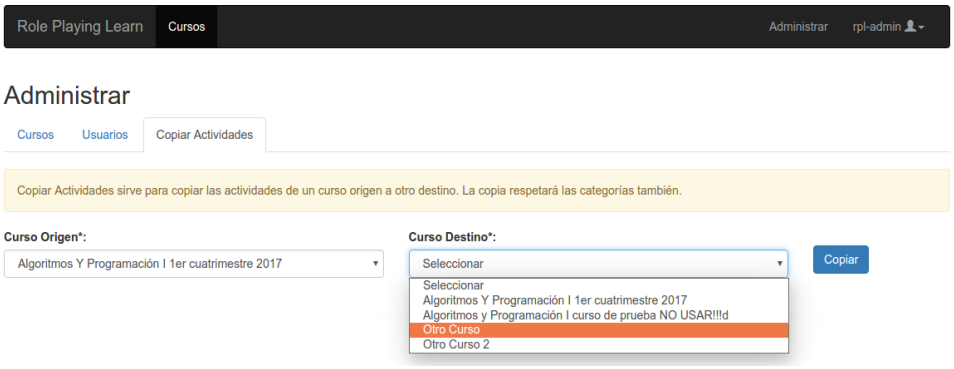
Sin importar el modo que se haya elegido, se debe completar el *Template* con el código por defecto que le va a aparecer al alumno.

Copiar actividades

Es muy normal que al finalizar un curso muchas actividades vayan a ser reutilizadas en un curso nuevo. Es por eso que el administrador puede copiar las categorías y actividades de un curso a otro.

Gracias a esto, los profesores no tendrán que crear todas las actividades cada vez que generen un curso nuevo. Esto facilita el trabajo del profesor ya que al copiar automáticamente las actividades del curso previo, solo deben agregar, editar o eliminar las actividades que deseen, ahorrándoles tiempo de trabajo.

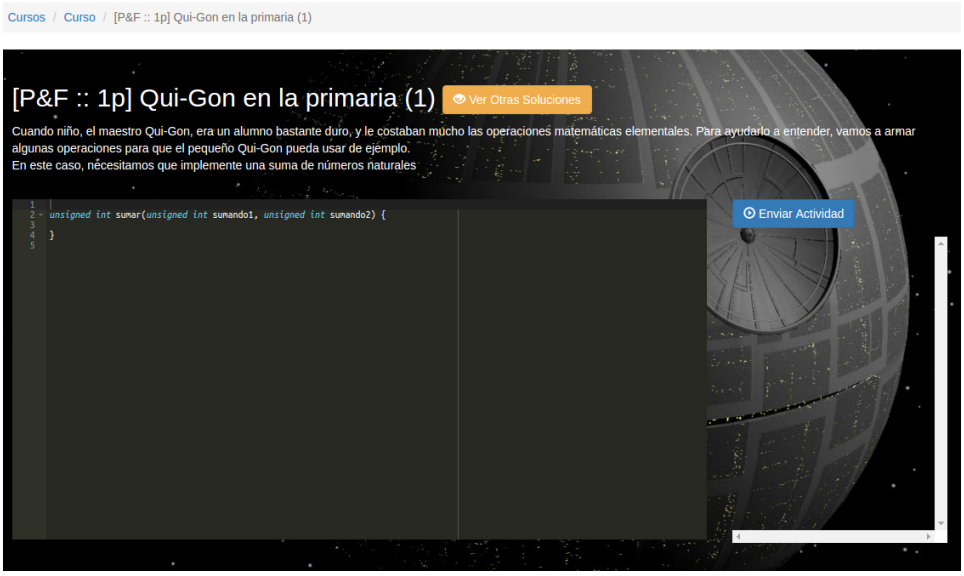
Figura 3.6: Copiar actividades.



Enviar solución

El alumno de un curso, al seleccionar una actividad para resolver, verá una pantalla con el nombre de la actividad, la descripción y el *template* con el código generado por el profesor.

Figura 3.7: Pantalla para enviar actividad.



Una vez resuelta, el alumno debe enviar la solución. Al enviarla, verá una pantalla que le indica que la actividad se está ejecutando y, una vez finalizada, verá el resultado de la misma.

Figura 3.8: Ejecución de la actividad en proceso.

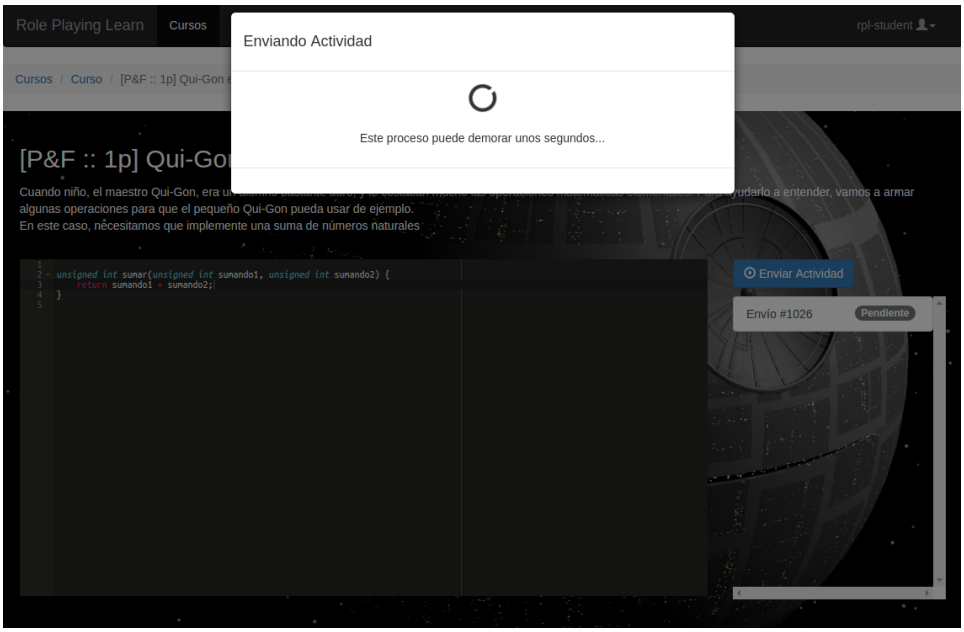
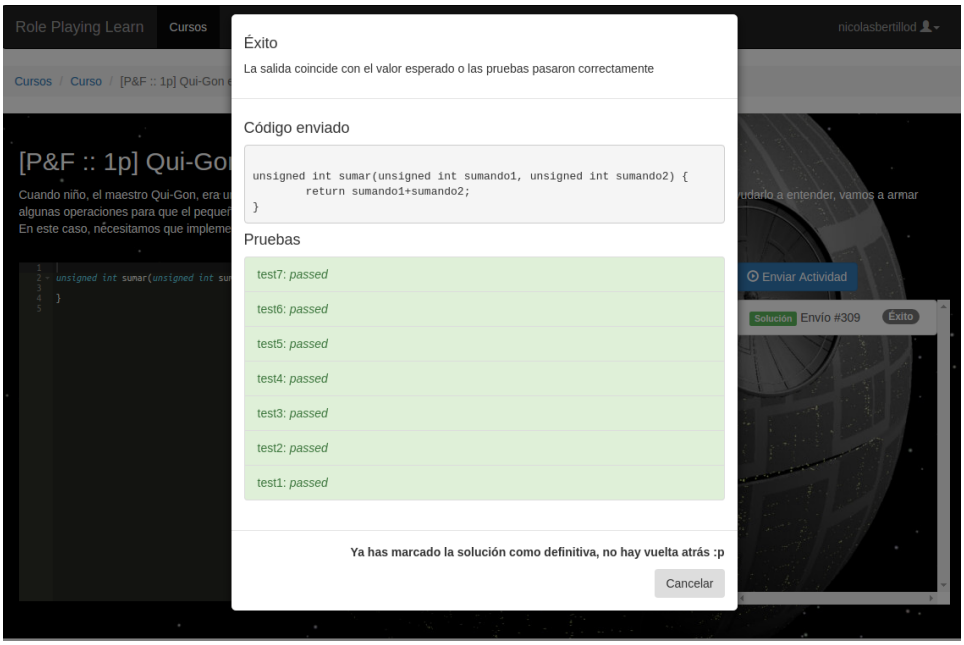


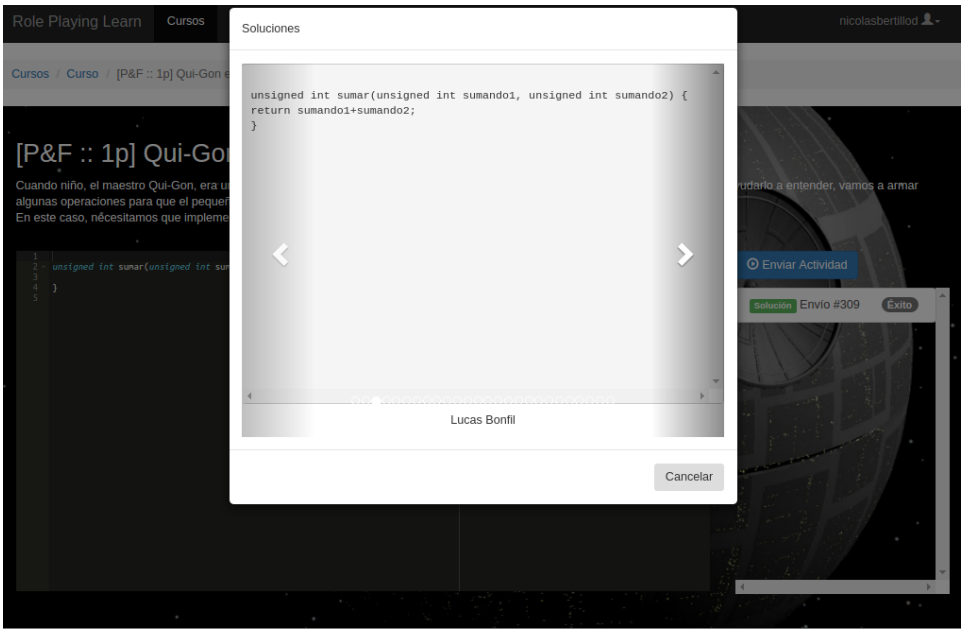
Figura 3.9: Actividad resuelta correctamente.



En caso de que la solución sea exitosa, el alumno puede marcarla como solución definitiva. El alumno al marcar la solución como definitiva ya no puede cambiarla pero le otorga la oportunidad de ver las resoluciones de otros alumnos. Así puede comparar su respuesta con la de otros alumnos y notar que no siempre hay única manera de resolver los problemas.

En particular, en el área de la programación y los algoritmos es frecuente que existan varias maneras de resolver un problema. También es común que haya dos soluciones que resuelvan el mismo problema pero una sea más eficiente. En definitiva, si bien la solución brindada ya no se puede cambiar, el alumno puede aplicar los conocimientos adquiridos en la resolución de la próxima actividad.

Figura 3.10: Ver otras soluciones.

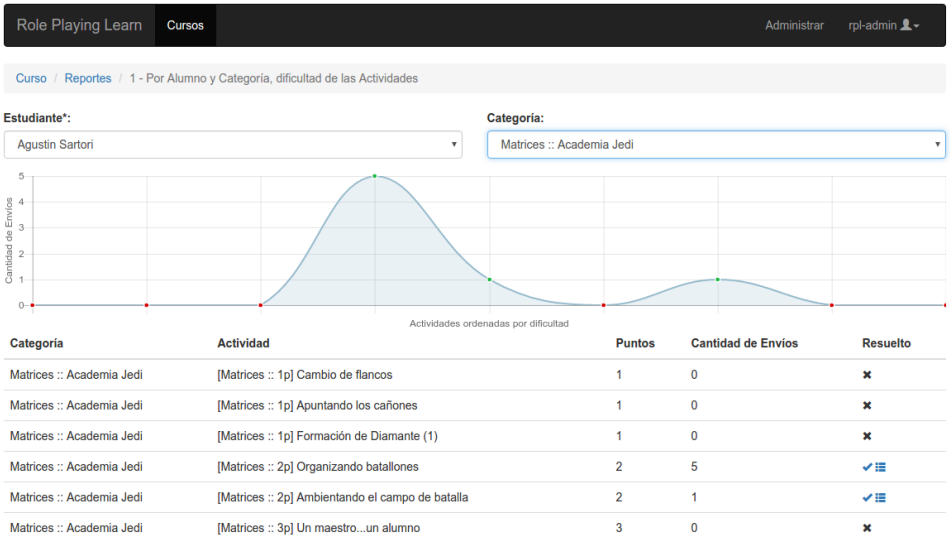


Ver reportes

Si se es profesor o alumno o ayudante de un curso es posible ver ciertos reportes. Los datos que se pueden ver en los reportes están limitados en el caso de ser ayudante según los alumnos que tiene asignados.

Por ejemplo, uno de los reportes disponibles presenta la actividad de un alumno con respecto a una categoría. El reporte permite visualizar las actividades del alumno en esa categoría, si fueron resueltas o no y el historial de envíos. En general, los reportes se complementan con gráficos.

Figura 3.11: Reporte por alumno y categoría.



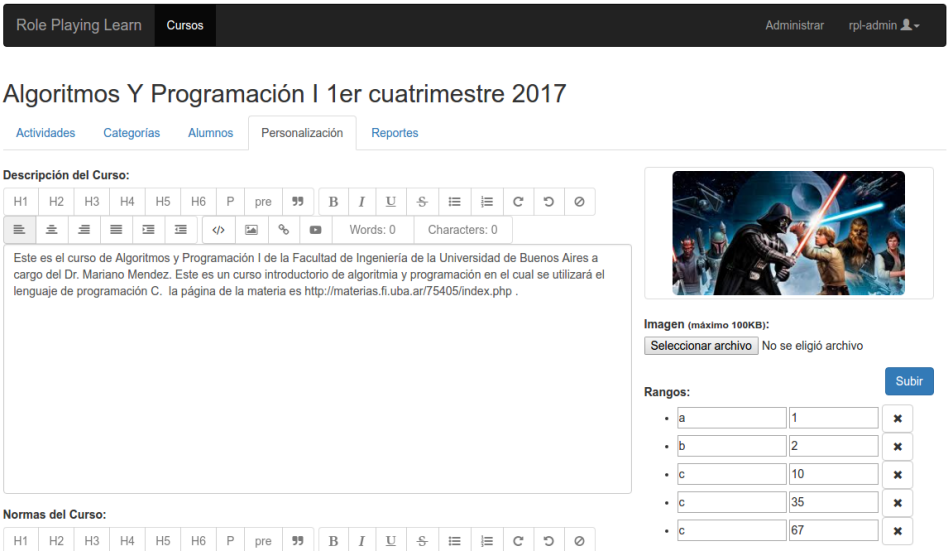
En el manual de usuario se especifican todos los reportes disponibles y detallados.

Personalización de un curso

Se pueden personalizar varios aspectos del curso. El profesor puede definir la descripción, las normas del curso (el alumno debe aceptarlas al momento de la inscripción) y una imagen del mismo. Además, puede definir *Rangos* que representan el nivel en el que están los alumnos según su puntaje en el curso.

Por otro lado, el profesor puede especificar un CSS para modificar cómo los alumnos ven la pantalla del curso.

Figura 3.12: Personalización.



3.2. Metodología y Proceso de Desarrollo de Software

Recurso	Roles
Alonso, Juan Manuel	Programador Backend
Desseno, Carlos	Programador Backend, Programador Frontend
Lew, Kevin	Programador Backend

Cuadro 3.1: Equipo de trabajo

- **Programador Backend:** rol a cargo del desarrollo del servidor.
- **Programador Frontend:** rol a cargo de la interfaz web y la interacción a través de la API de la plataforma.

A lo largo del trabajo profesional se trabajó con una metodología de trabajo iterativa e incremental. De esta manera, cuando fue necesario introducir alguna modificación se la pudo hacer fácilmente sin tener que esperar hasta el final del desarrollo.

En cada iteración se hizo una mejora sobre el producto, priorizando según lo que le aportaba más valor al mismo.

En cuanto a la forma de comunicación, las reuniones presenciales fueron nuestro foco principal. Tuvimos con el cliente varias de estas reuniones presenciales donde cada vez refinábamos más los requerimientos iniciales. Al

final de cada reunión establecíamos las prioridades, especialmente si surgía algún requerimiento no especificado inicialmente.

Para mantener una comunicación fluída con el cliente también utilizamos una herramienta llamada *Slack* que nos permitió responder cualquier duda del cliente y también recibir nuevas sugerencias sobre la marcha.

Unos de los desafíos más importantes que tuvimos fue cumplir con una fecha determinada (principio del primer cuatrimestre de 2017) para tener al menos los requerimientos iniciales de la plataforma. Esto fue como consecuencia de que nuestro cliente deseaba utilizar la plataforma de manera productiva ya para esa fecha.

Se comenzó por implementar los requerimientos necesarios para la puesta en producción de la plataforma, y luego se fueron agregando nuevas funcionalidades a lo largo del cuatrimestre. Esto implicó la necesidad de contar con una instancia de la plataforma corriendo en la nube.

3.2.1. Horas reales vs. horas estimadas

Se hace una comparación entre la estimación original dada en la propuesta del trabajo profesional y las horas insumidas reales en el trabajo. Es claro que las horas reales superan a las horas estimadas inicialmente. Esto se debe, en gran medida, a que surgieron nuevos requerimientos a lo largo del trabajo.

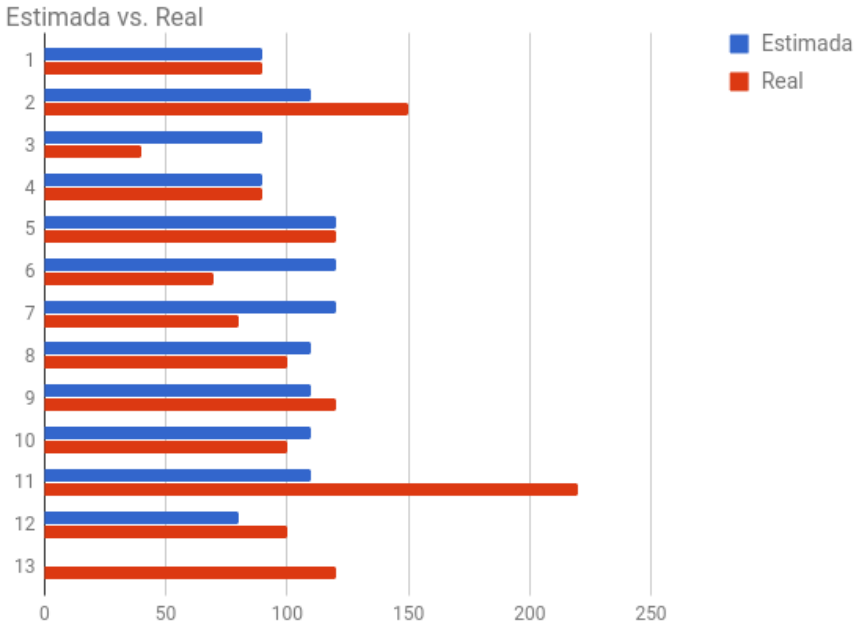
Un ejemplo de esto son los reportes que no estaban especificados en un comienzo o el hecho de poder copiar actividades de un curso a otro. A estos se sumaron pedidos de cambios que iban desde pequeñas modificaciones estéticas a cambios en el flujo de cierta funcionalidad (por ejemplo, cómo se debía esperar el resultado de una actividad).

En cierta medida, también se subestimó la redacción de la documentación (informe, documentación técnica y manual de usuario).

	Descripción	**H. estimadas**	**H. reales**
1	Investigación sobre herramientas a utilizar Propuesta Profesional	90	90
2	Diseño de la arquitectura Diseño del modelo Diseño del modelo de persistencia	110	150
3	Configuración del entorno Pruebas de concepto de compilación en varios lenguajes	90	40
4	Compilación programática en varios lenguajes	90	90
5	Gestión de cursos y actividades básica Servicio de compilación	120	120
6	Comunicación entre el servicio y el servidor	120	70
7	Persistencia de los resultados en el modelo	120	80
8	Gestión de usuarios Vistas de la web para subir actividades y administrar cursos	110	100
9	Web funcional para subir actividades	110	120
10	Web personalizable	110	100
11	Web funcional	110	220
12	Manual de usuario Documentación técnica	80	100
13	Reportes	0	120
		1260	**1400**

Cuadro 3.2: Horas estimadas vs horas reales insumidas

Figura 3.13: Gráfico horas estimadas vs reales.



3.3. Requerimientos

El desarrollo de la plataforma contempla los requerimientos funcionales y no funcionales descritos a continuación.

3.3.1. Requerimientos funcionales

- **Administración de cursos.** Los diferentes cursos puede ser creados y modificados.
- **Administración de usuarios.** La plataforma permite gestionar a los alumnos y a los profesores. En particular, los alumnos pueden tener un nombre de fantasía y un avatar, además de sus datos reales.
- **Administración de categorías.** Los profesores y los ayudantes pueden crear y editar categorías para las actividades dentro de un curso. Pueden también eliminarlas o deshabilitarlas.
- **Administración de actividades.** Los profesores y los ayudantes pueden crear actividades dentro de un curso. Pueden también eliminarlas o deshabilitarlas.
- **Elección de categoría.** Los profesores y los ayudantes pueden seleccionar la categoría de la actividad.

- **Lenguaje de programación.** Los profesores y los ayudantes pueden elegir el lenguaje de programación para cierta actividad.
- **Tipo de prueba.** Los profesores y los ayudantes pueden elegir el tipo de prueba a ejecutar sobre la actividad, ya sea entrada/salida o pruebas unitarias.
- **Asignación de puntos.** Los profesores y ayudantes pueden asignarle puntos a cada actividad.
- **Resolución**. Los alumnos pueden subir la resolución de sus actividades a la plataforma.
- **Pruebas automatizadas**. Las verificación de cada actividad se realiza automáticamente mediante pruebas automatizadas si corresponde.
- **Resultados de la actividad.** Los alumnos pueden ver los resultados de la actividad, sumando puntos por cada una. Los profesores y los ayudantes también puuedeen ver los resultados de cada actividad y realizar un seguimiento de ellas.
- **Otras resoluciones**. Los alumnos pueden ver las resoluciones de otros alumnos si ya completaron la actividad.
- **Personalización estética.** Cada curso puede tener una temática diferente definida por un profesor.
- **Normas y descripción del curso.** Los profesores puede escribir una descripción para el curso y una normas que los alumnos deben aceptar antes de inscribirse a uno.
- **Reportes.** Los profesores pueden ver los reportes de todos los alumnos de un curso mientras que los ayudantes pueden ver los reportes de los alumnos que les fueron asignados.
- **Reporte por alumno y categoría.** Los profesores y los ayudantes pueden ver el desempeño de un alumno en todas o en una categoría en particular.
- **Reporte de la dificultad de las actividades.** Los profesores y los ayudantes pueden ver la cantidad de envíos necesarios para resolver una actividad de cierto alumno en particular.
- **Reporte de promedio de envíos necesarios por actividad.** Los profesores y ayudantes pueden ver el promedio de envíos necesarios para la solución de las actividades de una determinada categoría.

- **Reporte por categoría.** Los profesores y ayudantes pueden ver para una determinada categoría si los alumnos resolvieron o no las actividades de la misma.
- **Reporte en forma de calendario.** Los profesores y ayudantes pueden ver la cantidad de envíos diarios de todos los alumnos en forma de calendario.
- **Reporte de porcentaje de completado.** Los profesores y ayudantes pueden ver para una determinada categoría un listado ordenado de los alumnos según el porcentaje de completitud de las actividades de dicha categoría.
- **Reporte de ranking.** Los profesores y ayudantes pueden ver un ranking de los alumnos según el puntaje en cierta fecha determinada.

3.3.2. Requerimientos no funcionales

- **Seguridad.** Las actividades deben correr en un ambiente seguro.
- **Rendimiento.** La plataforma debe soportar varios alumnos al mismo tiempo.
- **Escalabilidad.** La plataforma debe soportar el aumento de usuarios a lo largo de tiempo sin mayores modificaciones.
- **Costo.** El costo de operaciones debe ser el mínimo posible ya que se cuenta con un presupuesto acotado de apróximadamente 20USD por mes.

3.4. Descripción de la Arquitectura

El trabajo se puede dividir en dos partes: la interfaz web y las operaciones que se realizan en el servidor.

La interfaz web interactúa con el servidor a través de una *API RESTful*[3]. El término REST está asociado a un estilo arquitectural que debe cumplir con ciertos requisitos: los mensajes que son enviados deben permitir que ni el cliente ni el servidor recuerden un estado entre cada comunicación y los recursos son identificados únicamente por una URI.

El término REST también se asocia simplemente a la definición de una interfaz que hace uso del protocolo HTTP. A través de HTTP se envía el contenido del mensaje que, en nuestro caso, va a estar en formato JSON[11]. Los mensajes se envían mediante operaciones definidas de HTTP como POST, GET, PUT, DELETE, etc.

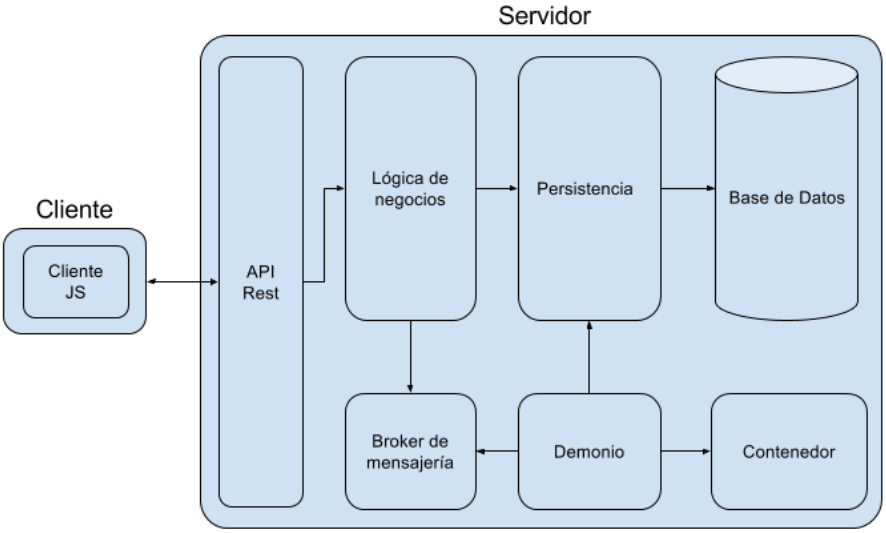
Simplificando, vía web se cuenta con un cliente que obtiene la información del servidor y, de ser necesario, que también envía cierta información.

Por otro lado, en el servidor hay un servicio que se encarga de responder a las peticiones de la *API Rest*. Estas peticiones pueden estar vinculadas a dos tipos de acciones: realizar una simple modificación en la base de datos o compilar y correr una actividad en un lenguaje de programación determinado.

Si es necesario realizar una modificación en la base de datos, el servidor se comunica con la misma a través de la capa de persistencia.

En caso de necesitar compilar una actividad, utilizamos una cola de mensajería. El servidor envia una petición de compilación a la cola. El *Demonio* es un proceso que está esperando las peticiones de compilación. El mismo atiende esa petición y utiliza una imagen de Docker para realizar la compilación y la ejecución. Docker es un contenedor que nos permite ejecutar la compilación de manera segura y sin tener el *overhead* de una virtualización completa. Dentro de la imagen de Docker utilizamos una herramienta para compilar y ejecutar código en diferentes lenguajes. Una vez finalizada la compilación y ejecución se actualiza el estado del envío de la solución.

Figura 3.14: Diagrama de la arquitectura.



3.4.1. Componentes de la arquitectura

- **Cliente JS.** Es la interfaz que se le muestra al usuario y que dispone de la menor cantidad de lógica posible. Se comunica con el servidor

a través de la API Rest. El cliente JS se basa en el patrón *MVC* y utiliza *AngularJS*.

- **API Rest.** Capa que une al cliente con la lógica de negocios y los servicios disponibles. Define las interfaces que serán usadas por el cliente. Ver sección 6.1.
- **Lógica de negocios.** Ejecuta las reglas de negocios y responde a los servicios que solicita la *API Rest*.
- **Broker de mensajería.** Se encarga de intermediar y rutear los mensajes entre la capa de negocios y el demonio. Se utiliza *RabbitMQ*.
- **Persistencia.** Es la capa que otorga un nivel de abstracción para la comunicación con la base de datos.
- **Demonio.** Es un proceso que lee los mensajes del broker de mensajería, realiza la ejecución de la actividad recibida a través del contenedor y analiza el resultado del mismo.
- **Contenedor.** El contenedor es la instancia de una imagen de *Docker* y el lugar seguro donde se corren las actividades. Siempre se parte de una imagen base y una vez finalizada la ejecución, el contenedor se destruye.
- **Base de datos.** Donde se almacena la información de manera relacional, materializado por *PostgreSQL*.

3.4.2. Ejecución de una actividad

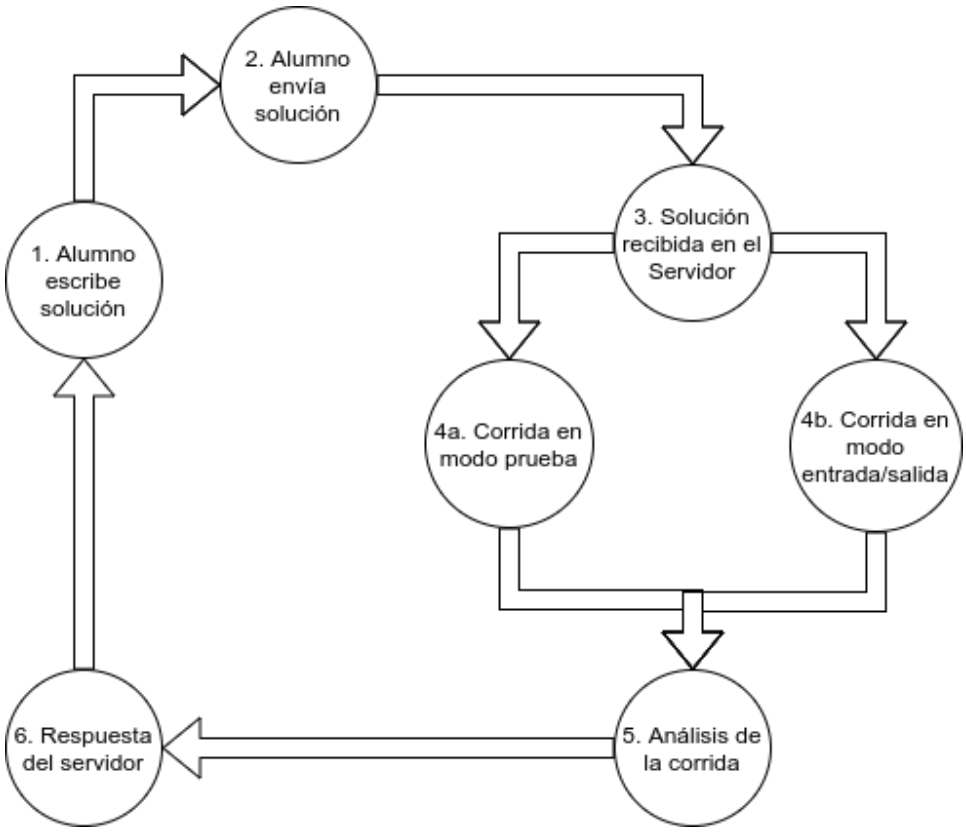
Ciclo de negocio

1. **Alumno escribe solución.** El alumno intenta resolver la actividad escribiendo el código necesario.
2. **Alumno envía solución.** El alumno envía el código escrito presionando el botón *Enviar Actividad*.
3. **Solución recibida en el Servidor.** El servidor recibe el código realizado por el alumno y encola la petición para ejecutar la solución del alumno.
4. a **Corrida en modo prueba.** Desencola la solución enviada por el alumno, compila y ejecuta las pruebas definidas por el profesor en un entorno controlado.
   b **Corrida en modo entrada/salida.** Desencola la solución enviada por el alumno, compila y ejecuta el código enviado por el alumno pasándole como entrada la definida por el profesor en un entorno controlado.

5. **Análisis de la corrida.** Se analiza cómo fue la ejecución de la actividad. En caso de que se haya corrido en modo prueba se analiza si las mismas fueron ejecutadas correctamente o si fallaron. En caso de que se haya corrido en modo entrada/salida se compara la salida obtenida con la definida por el profesor.

6. **Respuesta del Servidor.** El servidor brinda una respuesta indicando si la actividad fue resuelta correctamente o no. En caso de que no haya sido resuelta correctamente indica qué sucedió (error de compilación, error de ejecución, no pasó las pruebas, etc.)

Figura 3.15: Ciclo de envío de solución.



Paso a paso

1. El alumno envía una posible solución o *submission* de la actividad.

2. El servidor recibe la petición con la solución del alumno.

3. El servidor llama a un servicio que guarda esa posible solución en la base de datos.

4. El servidor envía el ID de la *submission* a la cola de mensajería.

5. Mientras tanto hay un *daemon* corriendo y procesando los mensajes de esa cola de mensajería.

6. El *daemon* obtiene el ID de la *submission* de la cola y la busca en las base de datos con su respectiva actividad.

7. El *daemon* ejecuta el contenedor de Docker con ciertos paramétros. Estos pueden ser el lenguaje de programación, el tipo (entrada/salida o pruebas), código enviado por el alumno y la información necesaria (entrada o pruebas). Adicionalmente existe un parámetro para enviar archivos extras que sean requeridos.

8. El contenedor de Docker contiene el *runner* quien crea los archivos con el código enviado por el alumno y las pruebas enviadas por el profesor, si corresponde, dentro del contenedor. Si es necesario, genera los archivos extras proporcionados por el profesor.

9. El *runner* compila el código de dos maneras diferentes, de acuerdo a si se trata de pruebas (en este caso es necesario referenciar las librerías para pruebas unitarias) o de entrada/salida.

10. En caso de que falle en el momento de compilación, el *runner* imprime por pantalla el resultado y finaliza la ejecución del contenedor.

11. Si no sucedió lo anterior, el *runner* ejecuta el código, también teniendo en cuenta si se trata del modo entrada/salida o pruebas. La compilación y la ejecución varía de acuerdo al lenguaje (C, Java y Python).

    En el caso de pruebas unitarias se usaron los siguientes *frameworks*: Criterion[18] (C), JUNIT[19] (Java) y unittest[20] (Python). Los archivos necesarios de cada librería o *framework* forma parte de la imagen de *Docker*.

12. El *runner* imprime por pantalla el resultado.

13. El *daemon* obtiene el resultado, lo procesa y lo guarda en la base datos

14. Mientras tanto el alumno ve que la actividad está siendo ejecutada gracias a la ayuda de un servicio que se llama hasta que indique que la actividad fue ejecutada.

15. El servidor devuelve la respuesta.

16. El alumno ve el resultado.

Estados posibles de una *submission*

- **Pendiente.** La *submission* aún no fue procesada por el *daemon*.
- **Error de compilación.** Falló en el paso de la compilación del código..
- **Error de ejecución.** Hubo un error en tiempo de ejecución, se superó el tiempo máximo definido para una ejecución o la memoria usada fue mayor a la definida.
- **Falló.** Error en las pruebas definidas por el profesor en caso de que la actividad sea con pruebas unitarias o no coinciden la salida obtenida con la esperada en caso de que el modo sea entrada/salida.
- **Éxito.** La solución propuesta pasó la etapa de compilación y se ejecutó correctamente, pasando las pruebas o la comparación con la salida esperada, según corresponda.

3.4.3. Tecnologías utilizadas

Para el trabajo se utilizaron las siguientes tecnologías:

- La plataforma corre en un entorno GNU/Linux. La versión elegida de GNU/Linux es Ubuntu 16.04 LTS.
- El backend fue desarrollado en el lenguaje de programación Java.
- Una parte del backend corre sobre WildFly[17]. WildFly es un servidor de aplicaciones Java EE de código abierto, flexible, confiable y que cumple con los estándares.
- Hibernate[10]. Hibernate es una herramienta para Java que facilita el mapeo de objetos de dominio a su representación en una base de datos. La estructura del mapeo puede ser declarada mediante anotaciones o xml. Permite al usuario abstraerse del código SQL subyacente. Además, provee la funcionalidad de *lazy/eager loading*.
- RESTEasy[16]. RESTEasy es una herramienta que simplifica el desarrollo de una API RESTful.

(a) WildFly

(b) Hibernate

(c) RESTEasy

- La interfaz web está desarrollada con la ayuda de Bootstrap[4]. Boostrap es un framework frontend open source y gratuito que permite crear un sitio web con una estética moderna de una manera sencilla. Bootstrap es responsive, lo que permite que el sitio se adapte al dispositivo desde donde se acceda, sin realizar modificaciones en el código.

- Ciertas funcionalidades de la web están desarrolladas con AngularJS[2]. AngularJS es un framework de Javascript, también de código abierto que permite y favorece el desarrollo basado en el patrón de MVC (Modelo Vista Controlador).

(a) Bootstrap

(b) AngularJS

- Como motor de base de datos se usa PostgreSQL[14]. PostgreSQL es un sistema de gestión de bases de datos relacional orientado a objetos y libre.

- Se utiliza Docker[7] como contenedor de software. Una de las ventajas principales de Docker con respecto a la virtualización tradicional es el ahorro en recursos: con Docker hay un sólo kernel corriendo compartido entre todos los contenedores. De esto se desprende que un contenedor puede arrancar mucho más rápido que una máquina virtual (las máquinas virtuales no comparten recursos). Todas estas características son posibles gracias al soporte nativo que provee el kernel de Linux en cuanto al aislamiento de recursos.

- Se utiliza RabbitMQ[15] como broker de mensajería. RabbitMQ es un software de código abierto que implementa el protocolo Advanced Message Queuing Protocol (AMQP) [1].

(a) PostgreSQL

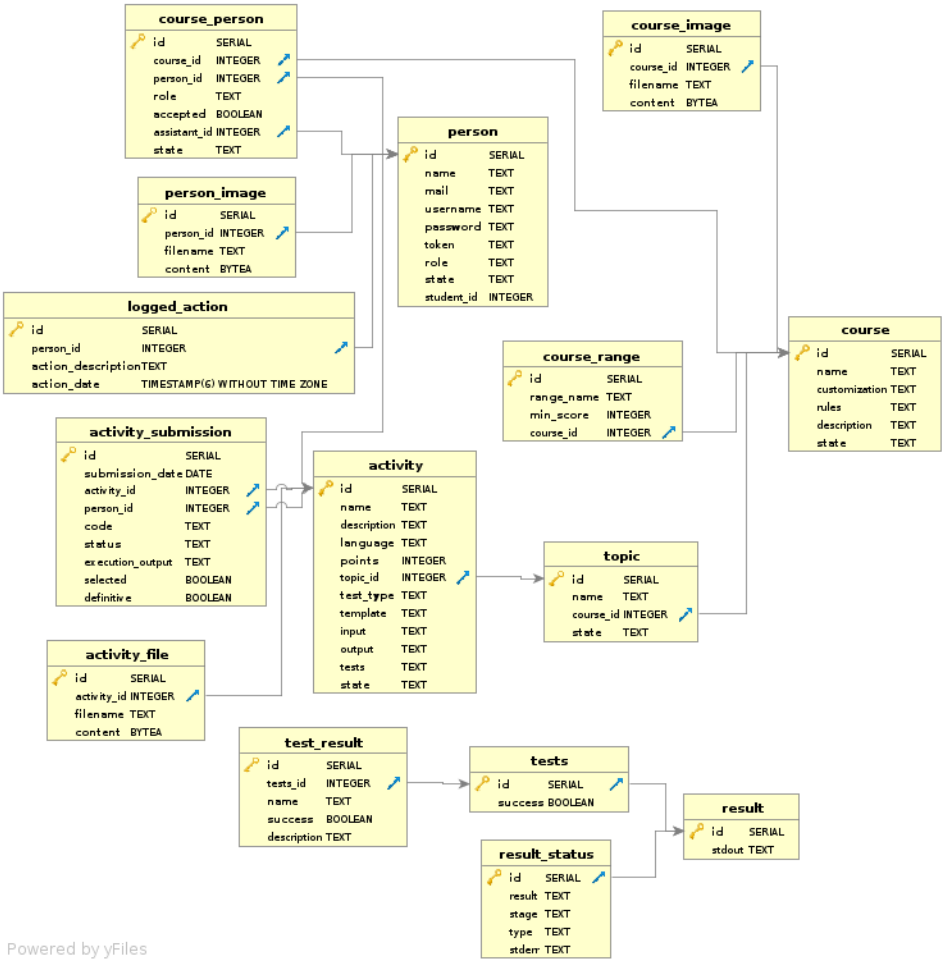
(b) Docker

(c) RabbitMQ

3.5. Diagramas

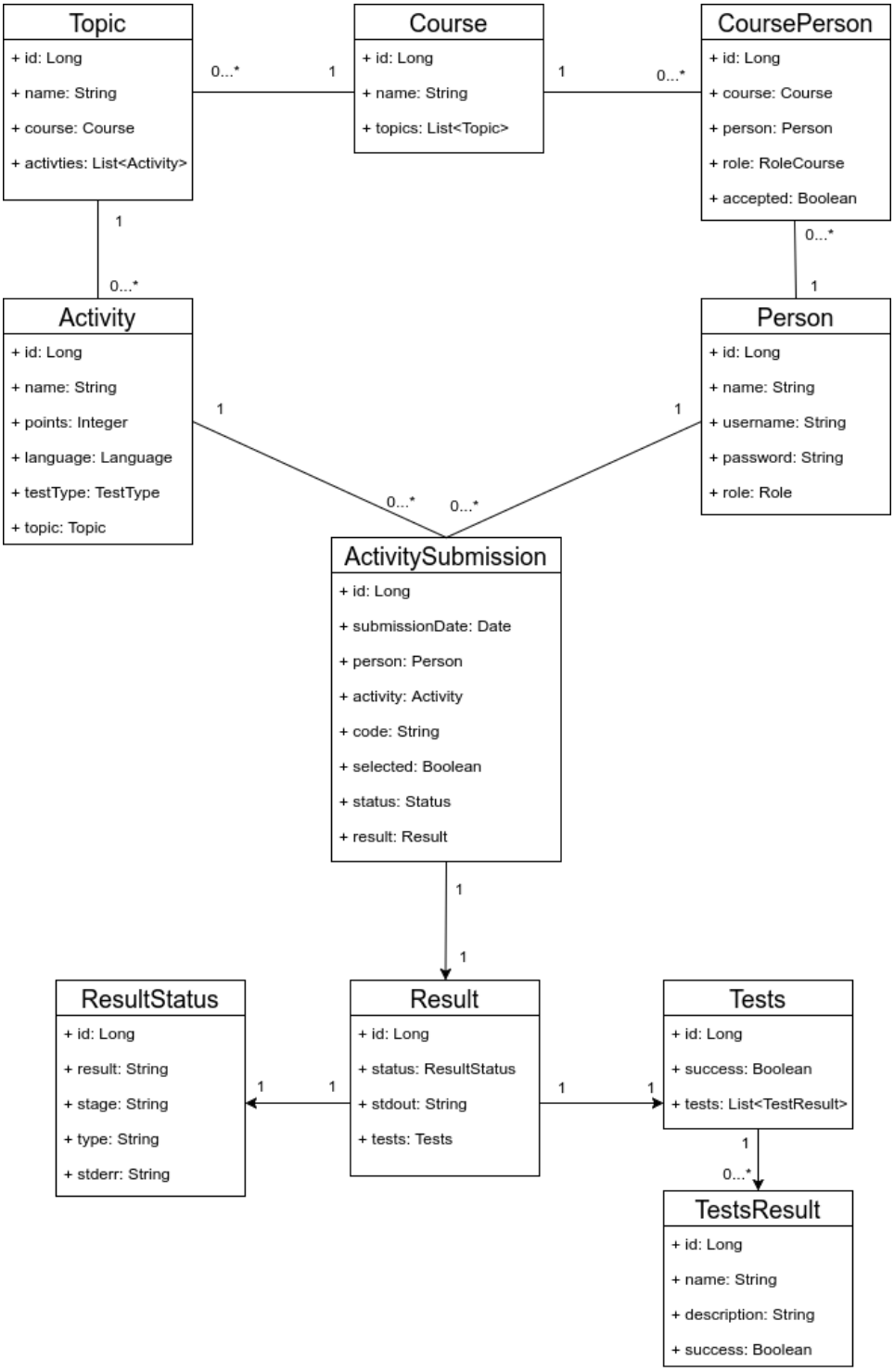
3.5.1. Esquema de la base de datos

Figura 3.19: Esquema de la base de datos.



3.5.2. Diagrama de las clases

Figura 3.20: Diagrama de las clases.



3.6. Casos de Uso

En esta sección se describen los casos de uso principales.

3.6.1. Inscripción en un curso

Actor	Alumno
Precondición	El curso debe existir y no estar previamente inscripto.
Postcondición	La inscripción del alumno queda pendiente.
Flujo	1. El alumno inicia sesión en la plataforma. 2. El alumno elige un curso donde inscribirse. 3. El alumno lee y acepta las normas del curso.

3.6.2. Aceptación de un alumno en un curso

Actor	Profesor
Precondición	El curso debe existir y la inscripción del alumno debe estar pendiente.
Postcondición	La inscripción del alumno fue aceptada.
Flujo	1. El profesor inicia sesión en la plataforma. 2. El profesor selecciona uno de sus cursos. 3. El profesor ve la lista de alumnos con inscripciones pendientes. 4. El profesor acepta la inscripción del alumno.

3.6.3. Creación de una actividad

Actor	Profesor
Precondición	El curso y la categoría deben existir.
Postcondición	La actividad fue creada.
Flujo	1. El profesor inicia sesión en la plataforma. 2. El profesor selecciona uno de sus cursos. 3. El profesor elige crear una actividad. 4. El profesor selecciona el nombre de la actividad, la descripción, los puntos que otorga, la categoría y el lenguaje. 5. El profesor selecciona el modo (pruebas o entrada/salida) e ingresa los datos según corresponda.

3.6.4. Envío de la solución de una actividad

Actor	Alumno
Precondición	El alumno debe estar inscripto en el curso.
Postcondición	El alumno envió una actividad.
Flujo	1. El alumno inicia sesión en la plataforma. 2. El alumno selecciona uno de sus cursos. 3. El alumno selecciona una actividad. 4. El alumno resuelve y envía la solución de una actividad. 5. El alumno espera el resultado.

3.7. Extensibilidad de la Plataforma

En esta sección se describe cómo extender la plataforma brindando, a modo de ejemplo, como añadir una sección con un reporte nuevo.

3.7.1. Front-end

Se asume que se trabaja en la carpeta *RPL-Web-App/src/main/webapp*.

1. Crear una carpeta nueva en *sections/reports* llamada *reportX* que va a representar a la nueva sección.

2. Crear el controlador y llamarlo *sections/reports/X/reportX.ctrl.js*.

```
'use strict';
angular
    .module('app.core')
    .controller('ReportXController', function($scope,
    ↪  $routeParams, RPL, $translate) {
        $scope.idCourse = $routeParams['idCourse']; // Id
        ↪  del curso
        $scope.reports = [];

                RPL.getReportX($scope.idCourse).query(function(data)
                ↪  {
                        $scope.reports = data;
                });

    });
```

3. Crear la vista de la nueva sección y llamarla *sections/reports/X/reportX.tpl.html*.

```
<ol class="breadcrumb">
    <li><a
    ↪  href="#!/manage/course/{{idCourse}}/">{{'COURSE'
    ↪  | translate}}</a></li>
    <li class="active"><a
    ↪  href="#!/manage/course/{{idCourse}}/?tab=4">{{'REPORTS'
    ↪  | translate}}</a></li>
    <li class="active">{{'REPORT_X' | translate}}</li>
</ol>

<div ng-show="(reports.length != 0)">
    <table class="table table-hover">
        <thead>
        <tr>
            <th></th>
            <th>{{'D1' | translate}}</th>
            <th>{{'D2' | translate}}</th>
        </tr>
        </thead>
        <tbody>
        <tr ng-repeat="data in reports">
            <td>{{data.d1}}</td>
            <td>{{data.d2}}</td>
        </tr>
        </tbody>
```

```
    </table>
</div>
```

4. Agregar el valor de los nuevos campos internacionalizables editando el archivo. *languages/locale-es.json*.

```
{
...
"REPORT_X": "Reporte X",
"D1": "dato 1",
"D1": "dato 2",
...
}
```

5. Definir el servicio en *services/rpl.srv.js*.

```
'use strict';

angular
    .module('app.services')
    .factory('RPL', service);

function service($resource, API_URL) {
    return {
        ...
        'getReportX': getReportX,
        ...
    };

    ...

    function getReportX(courseId) {
        return $resource(API_URL + "reports/X/" +
        ↪  courseId, null, {'query':  {method:'GET',
        ↪  isArray:true}});
    }

}
```

6. Agregar nueva ruta en el archivo *app.routes.js*.

```
'use strict';

angular
    .module('app.routes', ['ngRoute'])
```

```
    .config(config);

function config ($routeProvider) {
    $routeProvider.

        when('/course/:idCourse/activity/:idActivity', {
           templateUrl:
            ↪  'sections/activity/activity.tpl.html',
           controller: 'ActivityController as activity'
        })

        ...

        .when('/manage/course/:idCourse/reports/X', {
            templateUrl:
            ↪  'sections/reports/reportX/reportX.tpl.html',
            controller: 'ReportXController as reportX'
        })

                ...

        .otherwise({
            redirectTo: '/login'
        });
}
```

7. Especificar en el *index.html* el nuevo controlador.

```
<!-- CONTROLLERS -->
    ...
<script
 ↪  src="sections/reports/reportX/reportX.ctrl.js"></script>
...
```

3.7.2. Back-end

Se asume que se trabaja en la carpeta *RPL-Server*.

1. Crear en el modelo una nueva clase que represente el nuevo reporte en el que los atributos son los datos que se desean mostrar: *rpl-model/src/main/java/com/rpl/model/reports/ReportX.java*

```
public class ReportX {
    private String d1;
    private String d2;
        ...
```

2. Crear la consulta de base de datos y agregar el método que la utilice en *rpl-persistence/src/main/java/com/rpl/persistence/ReportDAO.java*. Los nombres de las columnas devueltas por la consulta deben ser igual a los atributos de la clase creada en el modelo.

```
public List<ReportX> getReportX(Long courseId) {
        return entityManager.createNativeQuery("SELECT
        ↪  ...", "reportX").getResultList();
}
```

3. Relacionar la consulta definida el ReportDAO con el modelo en *rpl-datasource/src/main/resources/META-INF/orm.xml*.

```
<sql-result-set-mapping name="reportX">
        <constructor-result
        ↪  target-class="com.rpl.model.reports.ReportX">
        <column name="d1"/>
        <column name="d2"/>
        </constructor-result>
</sql-result-set-mapping>
```

4. Definir nuevo servicio para obtener el reporte en *rpl-service/src/main/java/com/rpl/service/Rep*

```
public List<ReportX> getReportX(Long courseId);
```

5. Implementar el servicio para obtener el reporte en *rpl-service/src/main/java/com/rpl/serviceIm*

```
public List<ReportX> getReportX(Long courseId) {
        return reportDAO.getReportX(courseId);
}
```

6. Crear nuevo método en *rpl-server-api/src/main/java/com/rpl/endpoint/ReportEndpoint*.java definiendo la interfaz que será llamada por el cliente. Este método llamará al servicio correspondiente y devolverá al cliente el modelo creado.

```
@GET
@Path("/X/{courseId}")
@Produces(MediaType.APPLICATION_JSON)
public Response getReportX(@PathParam("courseId") Long
↪  courseId) {
        ...
}
```

Capítulo 4

Caso de Estudio

4.1. Contexto

El trabajo fue puesto a prueba en una cátedra de "Algoritmos y Programación I" de la Facultad de Ingeniería, Universidad de Buenos Aires durante el primer cuatrimestre de 2017.

La materia "75.40 Algoritmos y Programación I" está presente en el tercer cuatrimestre de la carreras de Ingeniería en Informática y Licenciatura en Análisis de Sistemas, otorgando seis créditos. Los créditos otorgados implican un carga horaria de seis horas presenciales a la que luego hay que sumarle la gran dedicación que hay que prestarle fuera de la Facultad. Esta dedicación implica la realización de distintas actividades, lectura de apuntes, entre otras tareas.

La cátedra [1] en cuestión fue la que se encuentra a cargo del profesor Mariano Méndez.

4.2. Servidor

La plataforma fue instalada en un servidor de *DigitalOcean*[6]. *DigitalOcean* es un proveedor de servidores virtuales. Para la instalación de la plataforma se necesitó disponer de un servidor de 2GB de memoria RAM y dos procesadores. *DigitalOcean* donó una suma de dinero para poder afrontar los gastos que lleva disponer un equipo de esas características.

Figura 4.1: DigitalOcean

[1] http://materias.fi.uba.ar/75405/index.php

4.3. Participación de la Cátedra

La participación de los profesores y los ayudantes de la cátedra fue fundamental.

Por un lado, los ayudantes y los profesores participaron activamente en la creación de los contenidos, es decir, crearon las categorías y las actividades. La creación de las actividades implicaba imaginación y trabajo para que el nivel sea el adecuado.

Por otro lado, también participaron en el desarrollo de la aplicación brindando continuamente sugerencias y reportando errores.

La cátedra cuenta con once ayudantes que colaboran en la práctica y un profesor que dicta las clases teóricas.

4.4. Participación de los Alumnos

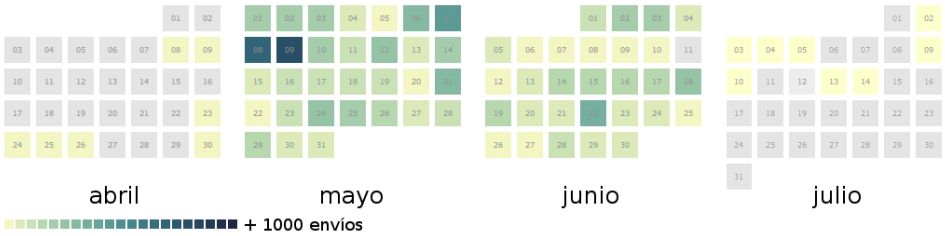
En la figura 4.2 se muestra la participación de los alumnos en la plataforma a través de la cantidad de envíos realizados. Llamamos envío a una posible solución que fue enviada por un alumno.

En la figura se puede apreciar, en colores más oscuros, mayor participación. Notar que el día previo al parcial (10 de mayo) hay un pico de participación.

También se puede remarcar que la participación en la plataforma siguió aún luego de que el cuatrimestre haya finalizado.

En la plataforma RPL participaron alrededor de cincuenta alumnos, notándose cierta correlación entre el puntaje obtenido y la nota en los parciales.

Figura 4.2: Calendario y actividad de los alumnos.



4.4.1. Patrón detectado

A través de los reportes y el historial de envíos de cada alumno en las actividades notamos un patrón recurrente en la resolución de las actividades.

En general, el alumno resolvía una actividad en tres etapas. Primero solía tener errores de compilación (en general, errores de sintaxis). Es claro que estos errores de compilación eran menos frecuentes con el paso del tiempo. Al

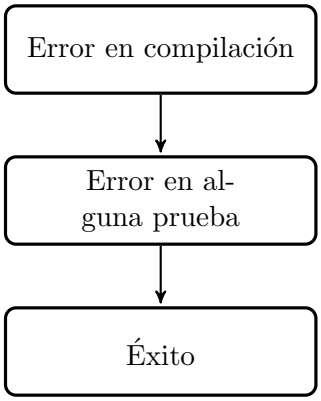


Figura 4.3: Patrón detectado

solucionar este tipo de errores se encontraba con que una o más pruebas no pasaban. Allí refinaban la solución para pasar las demás pruebas y a veces se notaba una regresión con respecto a las anteriores. El siguiente paso, entonces, consistía en revertir la regresión y cumplir con todas las pruebas. Cuando cumplían con esto, llegaban al éxito en la solución.

En las figuras 4.4, 4.5, 4.6 y 4.7 se puede observar tal progresión. En la figura 4.4 se ve el error de compilación (variable no declarada). Más tarde, en la figura 4.5 pasan todas las pruebas menos una. Luego, en la figura 4.6 pasa la prueba complementaria. En la figura 4.7 se llega al éxito.

Figura 4.4: Progresión del alumno: compilación falló.

Envío #526 Error en Compilación

```
#include<stdbool.h>

#define TAMANIO_UNIDAD 5

int formacionEsValida(char tropa[TAMANIO_UNIDAD][TAMANIO_UNIDAD]) {
    // Trasponer
    int cantTrooperBienPosicionado;
    int cantTotalTrooper = TAMANIO_UNIDAD - 1;
    int posicionXdeCabo, posicionYdeCabo;
    int coordEnXTrooper[TAMANIO_UNIDAD], coordEnYTrooper[TAMANIO_UN
    int distanciaXaC, distanciaYaC;
    bool encontroPrimerTrooper = false;
    // Consigo todas las coordenadas de los trooper y del cabo
    for (int i = 0; i < TAMANIO_UNIDAD; i++) {
        for (int j = 0; j < TAMANIO_UNIDAD; j++) {
            if (tropa[i][j] == 'T') {
                coordEnXTrooper[trooperCargados] = i;
                coordEnYTrooper[trooperCargados] = j;
            }
            if (tropa[i][j] == 'C') {
                posicionXdeCabo = i;
```

Error

```
                In file included from test.c:2:0:
            solution.c: In function 'formacionEsValida':
solution.c:17:33: error: 'trooperCargados' undeclared (first use in th
                        coordEnXTrooper[trooperCargados] = i;
                                                        ^
solution.c:17:33: note: each undeclared identifier is reported only o
```

Figura 4.5: Progresión del alumno: pasan algunas pruebas.

Envío #528 Falló

```
#include<stdbool.h>

#define TAMANIO_UNIDAD 5

int formacionEsValida(char tropa[TAMANIO_UNIDAD][TAMANIO_UNIDAD]) {
    // Trasponer
    int cantTrooperBienPosicionado;
    int cantTotalTrooper = 0;
    int posicionXdeCabo, posicionYdeCabo;
    int coordEnXTrooper[TAMANIO_UNIDAD], coordEnYTrooper[TAMANIO_UN
    int distanciaXaC, distanciaYaC;
    bool encontroPrimerTrooper = false;
    // Consigo todas las coordenadas de los trooper y del cabo
    for (int i = 0; i < TAMANIO_UNIDAD; i++) {
        for (int j = 0; j < TAMANIO_UNIDAD; j++) {
            if (tropa[i][j] == 'T') {
                coordEnXTrooper[cantTotalTrooper] = i;
                coordEnYTrooper[cantTotalTrooper] = j;
                cantTotalTrooper++;
            }
            if (tropa[i][j] == 'C') {
```

Pruebas

- test5: *failure*
- test4: *passed*
- test3: *passed*
- test2: *passed*
- test1: *passed*

Figura 4.6: Progresión del alumno: pasan algunas pruebas.

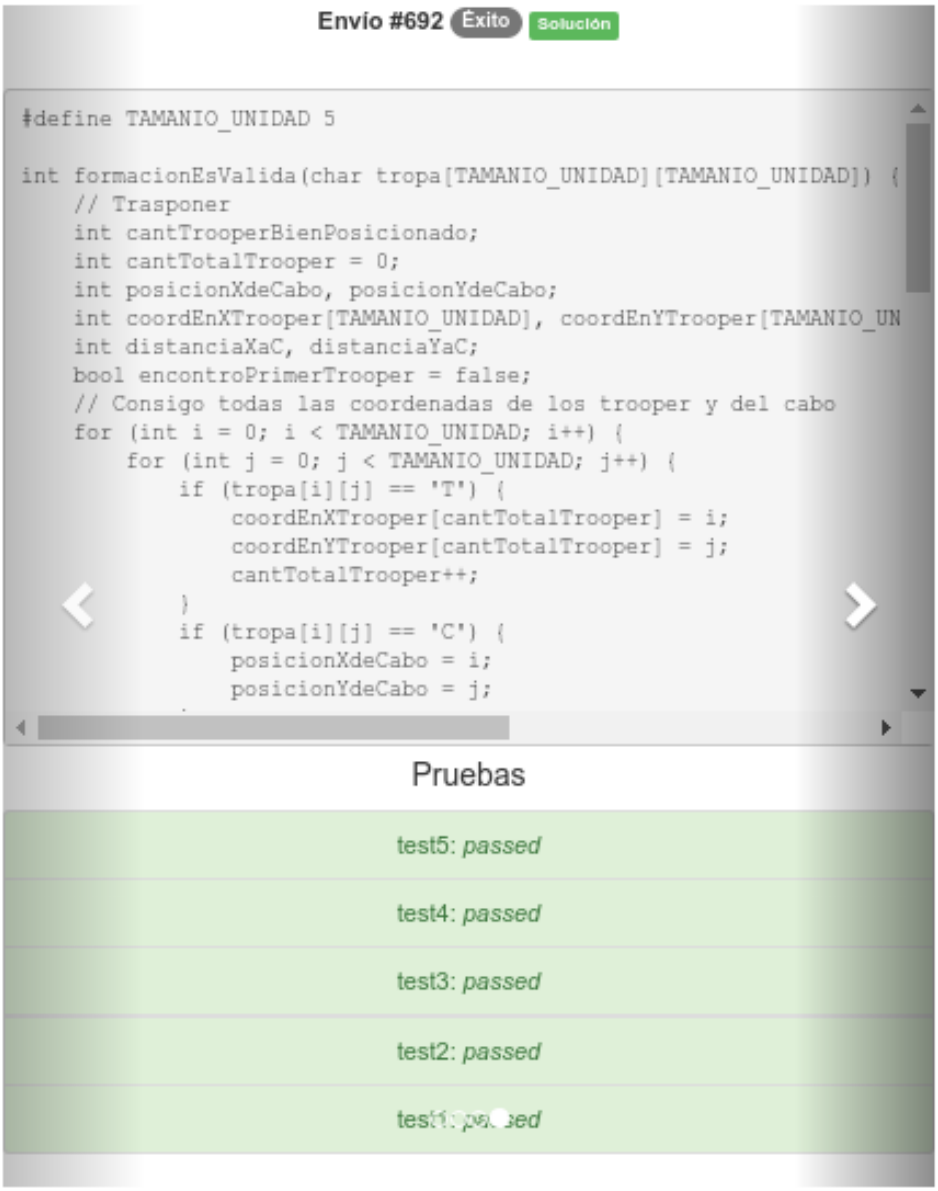
Envío #529 Falló

```
#include<stdbool.h>

#define TAMANIO_UNIDAD 5

int formacionEsValida(char tropa[TAMANIO_UNIDAD][TAMANIO_UNIDAD]) {
    // Trasponer
    int cantTrooperBienPosicionado;
    int cantTotalTrooper = 0;
    int posicionXdeCabo, posicionYdeCabo;
    int coordEnXTrooper[TAMANIO_UNIDAD], coordEnYTrooper[TAMANIO_UN
    int distanciaXaC, distanciaYaC;
    bool encontroPrimerTrooper = false;
    // Consigo todas las coordenadas de los trooper y del cabo
    for (int i = 0; i < TAMANIO_UNIDAD; i++) {
        for (int j = 0; j < TAMANIO_UNIDAD; j++) {
            if (tropa[i][j] == 'T') {
                coordEnXTrooper[cantTotalTrooper] = i;
                coordEnYTrooper[cantTotalTrooper] = j;
                cantTotalTrooper++;
            }
            if (tropa[i][j] == 'C') {
```

Pruebas

- test5: *passed*
- test4: *failure*
- test3: *failure*
- test2: *failure*
- test1: *failure*

Figura 4.7: Progresión del alumno: pasan todas las pruebas.



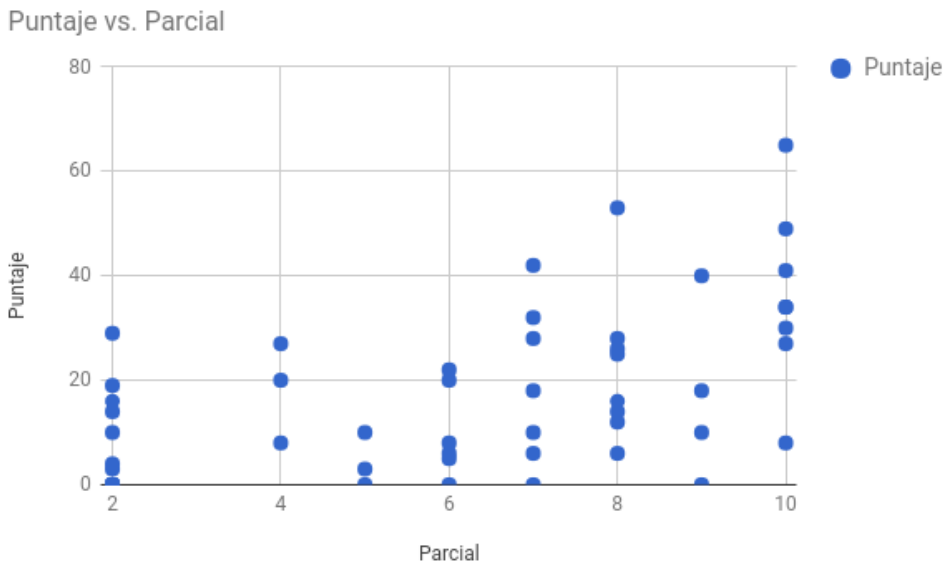
4.4.2. Relación entre la participación y la nota

Se analizó la relación entre la nota en los parciales de los alumnos y su participación en la plataforma y se obtuvo la siguiente información:

- La mayoría que hizo más de 20 puntos antes del parcial aprobó.
- Todos los que hicieron más de 40 puntos en los ejercicios se sacaron 7 o más en el parcial.

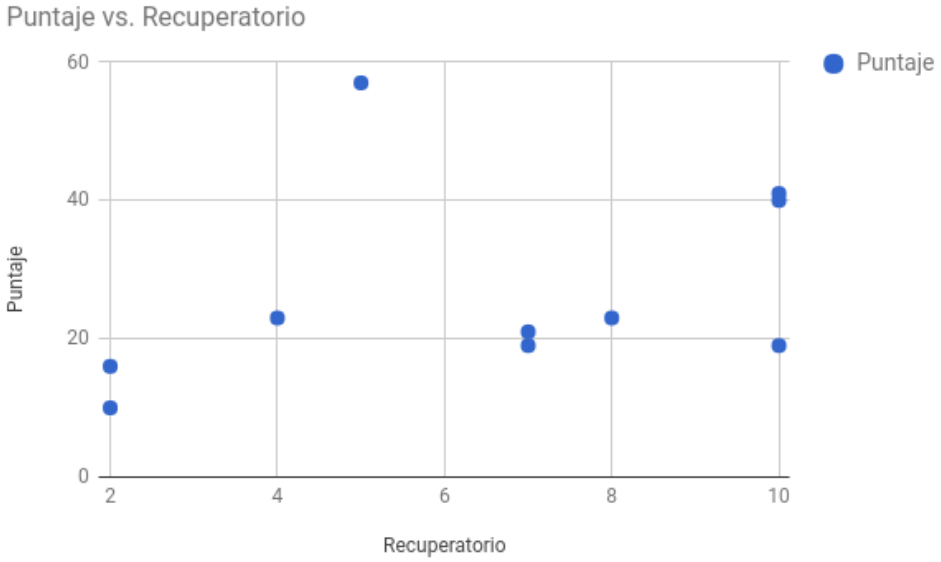
- Todos los que hicieron más de 50 puntos en los ejercicios se sacaron 8 o más en el parcial

Figura 4.8: Dispersión de la nota en el primer parcial y el puntaje obtenido hasta esa fecha.



- Los que no aprobaron el primer parcial e hicieron por lo menos 20 puntos después, aprobaron el recuperatorio.

Figura 4.9: Dispersión de la nota en el segundo parcial y el puntaje obtenido hasta esa fecha.



4.5. Personalización del Curso

En este primer curso de prueba se utilizó la temática de *"Star Wars"*. Eso incluyó el estilo y la forma estética del curso, así como también la narración personalizada de las actividades.

La personalización del curso favoreció el concepto de *Gamificación* antes mencionado. El orden de las actividades era importante y describía una historia.

4.6. Recepción de la Plataforma

Al finalizar el cuatrimestre los alumnos enviaron ciertos comentarios, algunos correspondientes a la plataforma.

Alguno de ellos fueron:

> *"Me pareció excelente la inclusión de la plataforma en la cursada"*
>
> *"Muy útil y entretenida"*
>
> *"Super"*
>
> *"Una de las cosas que más me ayudó a practicar"*
>
> *"Excelente"*

"Es una buena herramienta"

"Una de las cosas que más me ayudo para practicar"

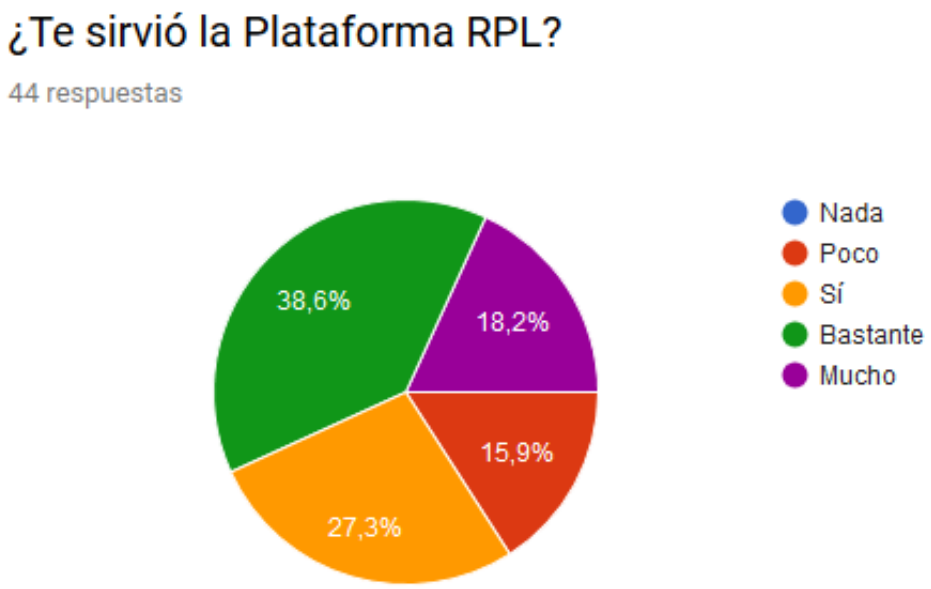
"Que los mensajes de error sean más descriptivos"

"Excelente, fue lo mejor que tuvo el curso, cambiaría un poco la redacción del ejercicio y ser un poco concreto en que hacer, ya que algunos ejercicios me costaron entenderlos"

"La ando haciendo medio a las corridas pero veo sirve para dejar en claro ejercicios simples (algunos) para el momento del examen"

También hubo una encuesta acerca de la utilidad de la plataforma de RPL.

Figura 4.10: Encuesta sobre la plataforma RPL.



Capítulo 5

Conclusiones

A lo largo de nuestra carrera universitaria no tuvimos la posibilidad de encontrar una herramienta interactiva que nos permita ejercitar nuestra habilidad para programar. Al realizar los ejercicios en papel no hay manera de verificar el correcto funcionamiento de los algoritmos, haciéndolos más tediosos y desviando la atención de lo realmente importante.

De lo anterior surgió la necesidad de crear una plataforma que permita realizar ejercicios de programación con pruebas automatizadas y haciendo énfasis en un enfoque educativo.

Se implementó un sistema que cumplió con esta necesidad, basado en las guías y las experiencias de una cátedra real de la facultad. Se obtuvo una plataforma que incentiva a los alumnos a resolver ejercicios a través de la ludificación.

La plataforma se puso a prueba en un curso real y el feedback recibido, tanto de los alumnos como el de los profesores, fue positivo. A partir de los reportes generados se pudo observar que los alumnos utilizaban la plataforma para practicar para los exámenes, ya que la actividad en la misma creció notablemente en las fechas previas a las instancias evaluatorias.

Se pudo observar que la plataforma fue útil tanto para los profesores como para los alumnos. Con esto, los profesores obtuvieron una herramienta para poder evaluar y realizar un seguimiento mas detallado de los alumnos. Los profesores pudieron ver el progreso individual de cada alumno, el estado del curso en general a través del ranking y por cada tema en especial.

Por el lado de los alumnos, la plataforma les permitió tener una opción realizar ejercicios de programación y practicar para los parciales. En el curso que utilizó la plataforma, se observó que los alumnos que usaron la misma para practicar antes del parcial obtuvieron una nota alta. A demás, en los casos en que los alumnos no realizaron suficientes ejercicios y no aprobaron el parcial, también se observa que los que luego comenzaron a practicar para el recuperatorio aprobaron el mismo.

5.1. Trabajos Futuros

A continuación se mencionan algunas posibles mejoras sobre la plataforma.

- Que el profesor pueda hacer comentarios sobre las soluciones enviadas.
- Facilitar la prueba de las actividades antes de crearlas definitivamente.
- Chequear que no haya dos soluciones similares entre distintos alumnos.
- Análisis del código en tiempo real.
- Solicitar ayuda en línea de un profesor o un ayudante.

Bibliografía

[1] Advanced Message Queuing Protocol. https://www.amqp.org/, 2016. [Online; accedida 15-octubre-2016].

[2] AngularJS. https://angularjs.org/, 2016. [Online; accedida 15-octubre-2016].

[3] API Rest. https://en.wikipedia.org/wiki/Representational_state_transfer, 2016. [Online; accedida 3-octubre-2016].

[4] Bootstrap. http://getbootstrap.com/, 2016. [Online; accedida 15-octubre-2016].

[5] codewars. https://www.codewars.com/, 2016. [Online; accedida 1-octubre-2016].

[6] DigitalOcean. https://www.digitalocean.com/, 2016. [Online; accedida 19-julio-2017].

[7] Docker. https://www.docker.com/, 2016. [Online; accedida 3-octubre-2016].

[8] Gradiance. http://www.newgradiance.com/, 2016. [Online; accedida 1-octubre-2016].

[9] HackerRank. https://www.hackerrank.com/, 2016. [Online; accedida 1-octubre-2016].

[10] Hibernate. http://hibernate.org/, 2016. [Online; accedida 15-octubre-2016].

[11] JSON. https://es.wikipedia.org/wiki/JSON, 2016. [Online; accedida 20-octubre-2016].

[12] Moodle. http://www.moodle.org/, 2016. [Online; accedida 1-octubre-2016].

[13] Piazza. https://www.piazza.com/, 2016. [Online; accedida 1-octubre-2016].

[14] PostgreSQL. https://www.postgresql.org/, 2016. [Online; accedida 15-octubre-2016].

[15] RabbitMQ. https://www.rabbitmq.com/, 2016. [Online; accedida 15-octubre-2016].

[16] RESTEasy. http://resteasy.jboss.org/, 2016. [Online; accedida 15-octubre-2016].

[17] WildFly. http://wildfly.org/, 2016. [Online; accedida 15-octubre-2016].

[18] Criterion. https://github.com/Snaipe/Criterion, 2017. [Online; accedida 19-julio-2017].

[19] JUNIT. http://junit.org/junit4/, 2017. [Online; accedida 19-julio-2017].

[20] unittest. https://docs.python.org/2/library/unittest.html, 2017. [Online; accedida 19-julio-2017].

Capítulo 6

Anexo

6.1. API

6.1.1. Autenticación

Para los endpoints que requieren autenticación es necesario enviar el header *Authorization* junto al token provisto al iniciar sesión. Por ejemplo, *Authorization:Bearer XXX* (XXX debe ser reemplazado por el token).

6.1.2. API

GET /courses?role=:role

Requiere autenticación. Devuelve una lista de cursos en los que está inscripto con un determinado rol. Si no se especifica el rol, se devuelven todos los cursos.

GET /courses/:courseId

Requiere autenticación. Devuelve la información de un curso. Debe ser profesor, alumno o ayudante del curso.

GET /courses/:courseId/customization.css

Devuelve el CSS correspondiente a un curso.

DELETE /courses/:courseId

Requiere autenticación. Elimina un curso. Debe ser administrador.

PUT /courses/:courseId/customization

Requiere autenticación. Permite editar el CSS correspondiente a un curso. Debe ser profesor del curso.

PUT /courses/:courseId

Requiere autenticación. Permite editar el nombre de un curso. Debe ser administrador.

POST /courses/:courseId/activities

Requiere autenticación. Crea una nueva actividad dentro del curso. Debe ser profesor o ayudante del curso.

POST /courses/activity/:activityId/file

Requiere autenticación. Agrega archivos en una actividad. Debe ser profesor o ayudante del curso.

GET /courses/:courseId/image

Devuelve la imagen de un curso.

POST /courses/:courseId/image

Requiere autenticación. Cambia la imagen de un curso. Debe ser profesor del curso.

POST /courses/:courseId/topics

Requiere autenticación. Crea una nueva categoría en un curso. Debe ser profesor o ayudante del curso.

POST /courses/:courseId/join

Requiere autenticación. Inscribe a una persona en un curso como alumno a la espera de ser aceptado. Debe ser profesor del curso.

POST /courses/:courseId/assistant

Requiere autenticación. Asigna a un alumno del curso un ayudante determinado. Debe ser profesor del curso.

GET /courses/:courseId/ranking

Requiere autenticación. Devuelve el ranking de puntajes de un curso. Debe ser profesor, alumno o ayudante del curso.

GET /courses/:courseId/range

Requiere autenticación. Devuelve una lista con los rangos (nombre y puntaje minimo) definidos por el profesor. Debe ser profesor del curso.

PUT /courses/:courseId/range

Requiere autenticación. Crea nuevos rangos en un curso. Debe ser profesor del curso.

POST /courses/:courseId/hide

Requiere autenticación. Marca un curso como deshabilitado. Debe ser administrador.

POST /courses/:courseId/unhide

Requiere autenticación. Marca un curso como habilitado. Debe ser administrador.

GET /topics/:topicId

Requiere autenticación. Devuelve la información de una categoría. Debe ser profesor o ayudante del curso.

DELETE /topics/:topicId

Requiere autenticación. Elimina una categoría. Debe ser profesor o ayudante del curso.

PUT /topics/:topicId

Requiere autenticación. Permite editar el nombre de una categoría. Debe ser profesor o ayudante del curso.

GET /topics/:topicId/activities

Requiere autenticación. Devuelve una lista con la información de las actividades de una categoría. Debe ser profesor, alumno o ayudante del curso.

POST /topics/:topicId/hide

Requiere autenticación. Marca una categoría como deshabilitada. Debe ser profesor o ayudante del curso.

POST /topics/:topicId/unhide

Requiere autenticación. Marca una categoría como habilitada. Debe ser profesor o ayudante del curso.

GET /activities/:activityId

Requiere autenticación. Devuelve la información de la actividad. Debe ser profesor, alumno o ayudante del curso.

GET /activities/:activityId/edit

Requiere autenticación. Devuelve la información editable de una actividad. Debe ser profesor o ayudante del curso.

DELETE /activities/:activityId

Requiere autenticación. Elimina una actividad. Debe ser profesor o ayudante del curso.

PUT /activities/:activityId

Requiere autenticación. Permite editar la información de la actividad. Los campos posibles son nombre, descripción, lenguaje, puntos, categoría, tipo de tests, el template, el input, output y los tests en sí. Debe ser profesor o ayudante del curso.

GET /activities/:activityId/submissions

Requiere autenticación. Muestra todos los envíos que tiene la actividad.

POST /activities/:activityId/submission

Requiere autenticación. Envía una nueva solución para la actividad. Debe ser estudiante del curso.

GET /activities/:fileId/files

Requiere autenticación. Obtiene los archivos correspondientes a una actividad. Debe ser profesor o ayudante del curso.

DELETE /activities/:fileId/file

Requiere autenticación. Borra un archivo de la actividad. Debe ser profesor o ayudante del curso.

POST /activities/:activityId/hide

Requiere autenticación. Deshabilita una actividad. Debe ser profesor o ayudante del curso.

POST /activities/:activityId/unhide

Requiere autenticación. Habilita una actividad. Debe ser profesor o ayudante del curso.

GET /activities/:activityId/solutions

Requiere autenticación. Muestra las soluciones enviadas para la actividad.

GET /submissions/:submissionId

Requiere autenticación. Muestra la información de un envío. Debe ser alumno del curso y haber enviado el envío.

GET /submissions/activity/:activityId/person/:userId

Requiere autenticación. Muestra los envíos de un usuario correspondientes a una actividad. Debe ser profesor o ayudante del curso.

POST /submissions/:submissionId/select

Requiere autenticación. Marca el envío como solución seleccionada. Debe ser alumno del curso y haber enviado el envío.

POST /submissions/:submissionId/definitive

Requiere autenticación. Marca el envío como solución definitiva. Debe ser alumno del curso y haber enviado el envío.

GET /admin/courses

Requiere autenticación. Obtiene la información de todos los cursos. Debe ser administrador.

POST /admin/courses

Requiere autenticación. Crea un nuevo curso. Debe ser administrador.

POST /admin/courses/:courseId/person

Requiere autenticación. Agrega una persona al curso. Debe ser administrador.

GET /admin/:courseId/assistants

Requiere autenticación. Muestra la información de los ayudantes de un curso. Debe ser administrador.

GET /admin/:courseId/professors

Requiere autenticación. Muestra la información de los profesores de un curso.

POST /admin/:courseId/person/:personId/leave

Requiere autenticación. Elimina a la persona de un curso

GET /admin/persons/:personId/information

Requiere autenticación. Obtiene la información de una persona.

PUT /admin/persons/:personId/information

Modifica la información de una persona. Se puede modificar el nombre, el mail y el rol.

PUT /admin/persons/:personId/password

Requiere autenticación. Modifica la contraseña de una persona.

DELETE /admin/persons/:personId

Requiere autenticación. Elimina a una persona

POST /admin/courses/copy/source/:sourceCourseId/dest/:destCourseId

Requiere autenticación. Crea una copia de un curso fuente a el curso destino.

GET /reports/submission/:id

Requiere autenticación. Obtiene el código de la solución enviada. Debe ser profesor o ayudante del curso.

GET /reports/1/:courseId/:personId/?topic=:topicId

Requiere autenticación. Devuelve por alumno y por actividad la cantidad de submissions hasta que se marcó como definitiva y si fue resuelta. Se debe seleccionar un alumno, y opcionalmente, la categoría. Debe ser profesor o ayudante del curso.

GET /reports/2/:topicId

Requiere autenticación. Devuelve por actividad, cuántas submissions realizó cada alumno hasta llegar a la solución definitiva promediadas. Si no hubo solución definitiva, no se toma en cuenta. Debe ser profesor o ayudante del curso.

GET /reports/3/:topicId

Requiere autenticación. Muestra las actividades de la categoría seleccionada y si fue o no resuelta por cada alumno. Debe ser profesor o ayudante del curso.

GET /reports/4/:courseId

Requiere autenticación. Muestra la actividad, es decir, envíos de soluciones, que tuvo el curso en el último año. Debe ser profesor o ayudante del curso.

GET /reports/5/:topicId

Requiere autenticación. Devuelve los datos del alumno y el porcentaje de completitud de la categoría correspondiente. Debe ser profesor o ayudante del curso.

GET /reports/6/:topicId/:personId

Requiere autenticación. Devuelve las actividades ordenadas por dificultad y la cantidad de intentos enviada por alumno. Debe ser profesor o ayudante del curso.

GET /reports/ranking/:courseId/?date=:date

Requiere autenticación. Devuelve el ranking de un curso a la fecha indicada. Debe ser profesor o ayudante del curso.

POST /authentication

Loguea un usuario, verificando sus credenciales.

PUT /authentication/password

Actualiza el password del usuario logueado.

POST /authentication/logout

Cierra la sesión del usuario logueado.

POST /authentication/register

Registra un nuevo usuario.

GET /manage/courses/:courseId/activities

Requiere autenticación. Devuelve todas las actividades de un curso (habilitadas y deshabilitadas). Debe ser profesor o ayudante del curso.

GET /manage/courses/:courseId/topics

Requiere autenticación. Devuelve todas las categorías de un curso (habilitadas y deshabilitadas). Debe ser profesor o ayudante del curso.

GET /manage/courses/:courseId/students

Requiere autenticación. Devuelve una lista con los alumnos de un curso. Debe ser profesor o ayudante del curso.

GET /manage/courses/:courseId/assistants

Requiere autenticación. Devuelve una lista con los ayudantes de un curso. Debe ser profesor del curso.

POST /manage/courses/:courseId/person/:personId/accept

Requiere autenticación. Acepta a un alumno en un curso. Debe ser profesor del curso.

POST /manage/courses/:courseId/person/:personId/leave

Requiere autenticación. Borra a un alumno de un curso. Debe ser profesor del curso.

GET /persons

Requiere autenticación. Devuelve una lista de todos los usuarios de la plataforma. Debe ser administrador.

GET /persons/information

Requiere autenticación. Devuelve la información del usuario que realizó la llamada.

GET /persons/:personId/image

Requiere autenticación. Devuelve la imagen de un usuario.

POST /persons/image

Requiere autenticación. Sube una imagen para el usuario que realizó la llamada.

PUT /persons/information

Requiere autenticación. Edita la información del usuario que realizó la llamada.

Parte II

Manual Técnico

Capítulo 7

Instalación y Despligue

7.1. Introducción

En este manual se indicará como instalar un servidor con la aplicación RPL corriendo. El manual se basa en la utilización de Ubuntu 16.04 LTS como sistema operativo de referencia.

7.2. Dependencias Necesarias

Antes de empezar es necesario contar con las dependencias que utilizaremos para compilar y ejecutar el código.

- RabbitMQ
  1. Instalación

     ```
     echo 'deb http://www.rabbitmq.com/debian/ testing
      ↪  main' | sudo tee
      ↪  /etc/apt/sources.list.d/rabbitmq.list
     wget -O-
      ↪  https://www.rabbitmq.com/rabbitmq-release-signing-key.asc
      ↪  | sudo apt-key add -
     sudo apt-get update
     sudo apt-get install rabbitmq-server
     ```

  2. Crear usuario y obtener permisos

     ```
     sudo rabbitmqctl add_user rpl rpl
     sudo rabbitmqctl set_permissions -p / rpl ".*" ".*"
      ↪  ".*"
     ```

- PostgreSQL
  1. Instalación

     ```
     sudo apt-get install postgresql postgresql-contrib
     ```

2. Crear base de datos, usuario y obtener permisos

```
sudo -u postgres psql
create user rpl with password 'rpl';
create database rpldb;
grant all privileges on database rpldb to rpl;
```

El nombre de base de datos y las credenciales de usuario descriptas son las default de la aplicación. Si se quiere utilizar otros, será necesario cambiarlo en la configuración de la aplicación, en el archivo *RPL-Server/rpl-datasource/src/main/resources/META-INF/persistence.xml*.

- Docker

1. Instalación

```
sudo apt-key adv --keyserver
↪ hkp://p80.pool.sks-keyservers.net:80 --recv-keys
↪ 58118E89F3A912897C070ADBF76221572C52609D
sudo apt-add-repository 'deb
↪ https://apt.dockerproject.org/repo ubuntu-xenial
↪ main'
sudo apt-get update
sudo apt-get install docker-engine
```

- Maven

1. Instalación

```
sudo apt-get install maven
```

- Bower

1. Instalación

```
curl -sL https://deb.nodesource.com/setup_4.x | sudo
↪ -E bash -
sudo apt-get install -y nodejs
npm install -g bower
```

- Wildfly

1. Instalación

```
wget
↪ http://download.jboss.org/wildfly/10.1.0.Final/wildfly-10.1.0.Final
.tar.gz
tar -xvzf wildfly-10.1.0.Final.tar.gz
```

2. Ejecución

```
wildfly-10.1.0.Final/bin/standalone.sh -b=0.0.0.0
↪ -bmanagement=0.0.0.0
```

7.3. Generación de Ejecutables y Archivos Extras

- rpl-server-api.war

```
cd RPL-Server/
mvn package -PwebApp
cp rpl-server-api/target/rpl-server-api.war
↪ {INSTALLATION_DIRECTORY}
```

- web-app.war

```
cd RPL-Web-App/
cd src/main/webapp/assets/ && bower install
cd RPL-Web-App/
mvn package
cp target/web-app.war {INSTALLATION_DIRECTORY}
```

- rpl-daemon-0.0.1.jar

```
cd RPL-Server/
mvn package -PstandaloneApp
cp rpl-daemon/target/rpl-daemon-0.0.1.jar
↪ {INSTALLATION_DIRECTORY}
```

- rpl-runner-0.0.1.jar

```
cd RPL-Server/
mvn package -PstandaloneApp
cp rpl-runner/target/rpl-runner-0.0.1.jar
↪ {INSTALLATION_DIRECTORY}
```

- Dockerfile

```
cp RPL-Server/rpl-runner/Dockerfile
↪ {INSTALLATION_DIRECTORY}
```

- extras/

```
cp -r RPL-Server/rpl-runner/extras/
↪ {INSTALLATION_DIRECTORY}
```

- scripts.sql

```
cp
↪ RPL-Server/rpl-datasource/src/main/resources/scripts.sql
↪ {INSTALLATION_DIRECTORY}
```

7.4. Despliegue de la Aplicación

- Generación del esquema de la base de datos.

```
psql -d rpldb -U rpl < scripts.sql
```

- Generación de imagen de Docker.

```
docker build -t rpl .
```

- Iniciar el demonio.

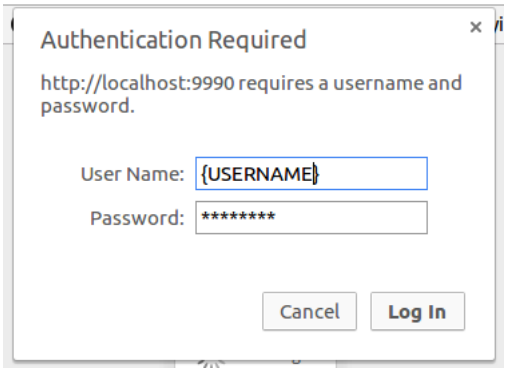
```
java -jar {INSTALLATION_DIRECTORY}/rpl-daemon-0.0.1.jar
```

- Creación de un usuario en Wildfly para desplegar.

```
./{INSTALLATION_DIRECTORY}/bin/add-user.sh
What type of user do you wish to add?
 a) Management User (mgmt-users.properties)
 b) Application User (application-users.properties)
(a): a

Enter the details of the new user to add.
Using realm 'ManagementRealm' as discovered from the
 ↪  existing property files.
Username :{USERNAME}
Password recommendations are listed below. To modify
 ↪  these restrictions edit the add-user.properties
 ↪  configuration file.
 - The password should be different from the username
 - The password should not be one of the following
 ↪  restricted values {root, admin, administrator}
 - The password should contain at least 8 characters, 1
 ↪  alphabetic character(s), 1 digit(s), 1
 ↪  non-alphanumeric symbol(s)
Password : {PASSWORD}
Re-enter Password : {PASSWORD}
What groups do you want this user to belong to? (Please
 ↪  enter a comma separated list, or leave blank for
 ↪  none)[  ]:
About to add user '{USERNAME}' for realm
 ↪  'ManagementRealm'
Is this correct yes/no? yes
Is this new user going to be used for one AS process to
 ↪  connect to another AS process?
```

```
e.g. for a slave host controller connecting to the master
↪  or for a Remoting connection for server to server EJB
↪  calls.
yes/no? yes
```

- Despliegue de empaquetado en Wildfly. Para iniciar el despliegue, nos conectaremos a http://localhost:9990 Primero es necesario iniciar se-



  sión con el usuario de despliegue que creamos en el paso anterior.

Una vez terminado el despliegue ya se puede empezar a utilizar la aplicación desde la URL *HOST:PORT/web-app*.

Parte III

Manual de Usuario

Capítulo 8

Funcionalidades

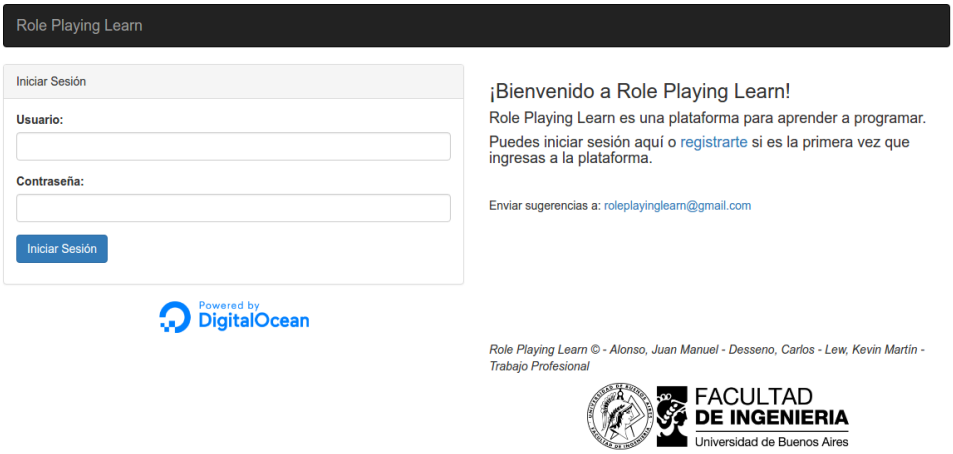
8.1. Introducción

El objetivo de esta sección es describir el uso de las funcionalidades principales de la plataforma de Role Playing Learn.

8.2. Inicio de Sesión

La pantalla de inicio presenta un formulario para iniciar sesión a la plataforma donde se debe ingresar el usuario y la contraseña.

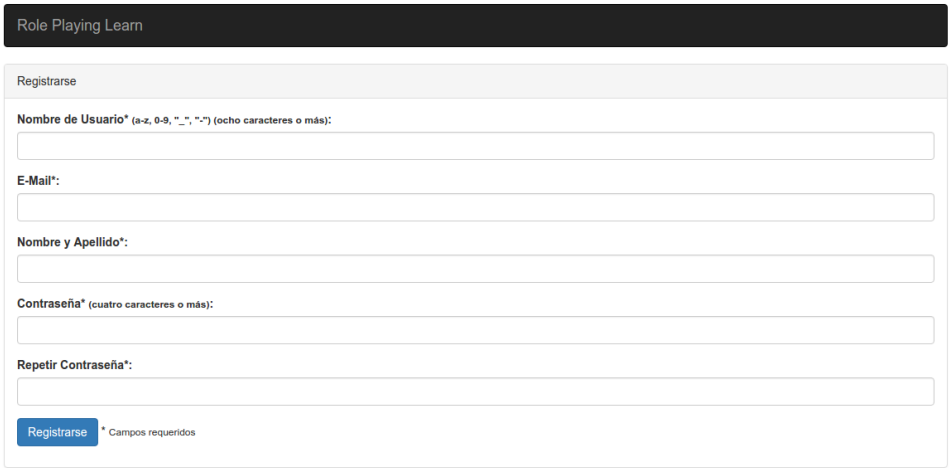
Figura 8.1: Inicio de sesión.



8.3. Registro

En caso de no contar con un usuario registrado, se puede crear uno.

Figura 8.2: Registro.



8.4. Mi Perfil

Si una persona ha iniciado sesión, puede editar su perfil ingresado a la sección "Mi Perfil", haciendo click en su nombre de usuario, en la parte superior derecha.

En esta sección el usuario puede actualizar sus datos personales, cambiar su contraseña y subir una foto de perfil.

Figura 8.3: Mi Perfil.

8.5. Cerrar Sesión

Si una persona ha iniciado sesión, puede cerrarla haciendo click en su nombre de usuario y luego en "Cerrar Sesión", en la parte superior derecha.

8.6. Como Administrador

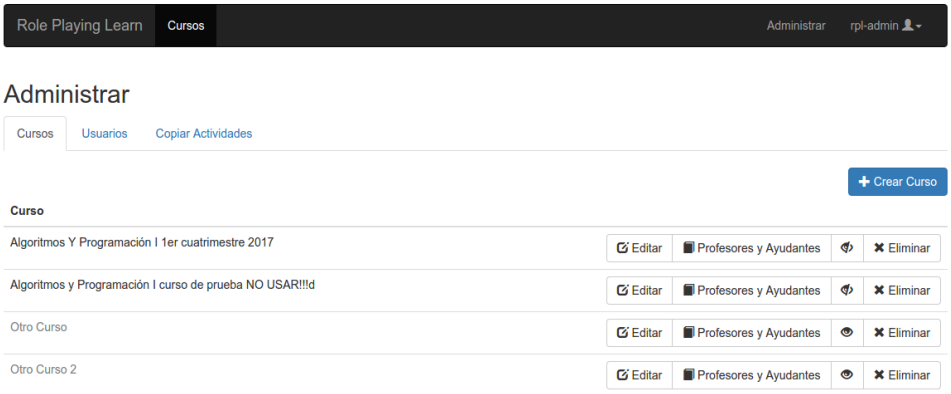
Si el usuario que inicia sesión tiene el rol de Administrador asignado, se puede acceder a un panel de administrador que le permite gestionar los cursos, los usuarios y otras acciones.

El panel se puede acceder a través de un botón llamado "Administrar" ubicado en la parte superior derecha.

8.6.1. Administrar cursos

En el tab de cursos el administrador puede crear, editar y eliminar cursos. También puede asignar profesores o ayudantes dentro de un curso.

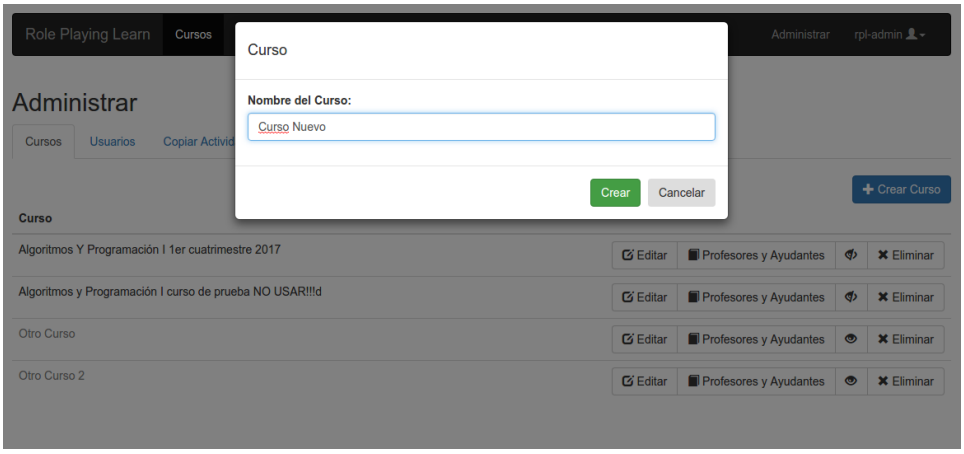
Figura 8.4: Administrar cursos.



Crear curso

Para crear un curso nuevo sólo se debe ingresar el nombre del mismo.

Figura 8.5: Crear curso.



Editar curso

Para editar un curso se debe hacer click en el botón Editar. Se puede editar sólo el nombre del curso.

Eliminar curso

Para eliminar un curso se debe hacer click en el botón Eliminar. Se pide una confirmación antes de eliminar un curso.

Habilitar/Deshabilitar curso

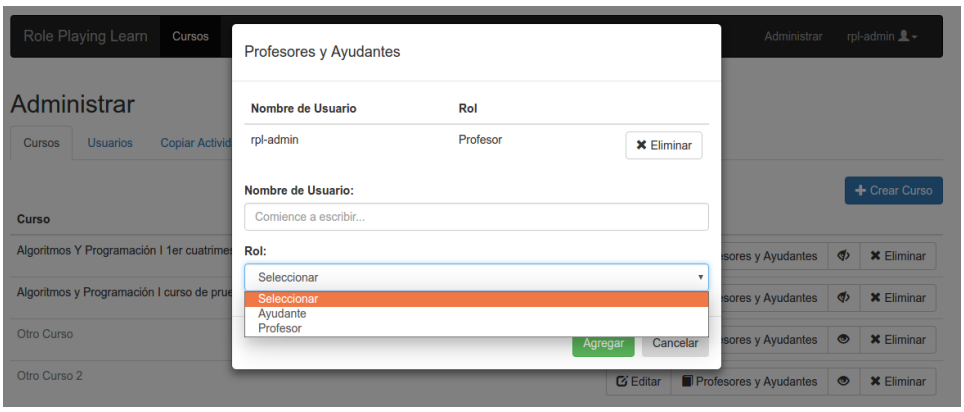
Existe la opción de habilitar/deshabilitar un curso haciendo click en el icono en forma de Ojo. El nombre del curso aparece grisado si se encuntra deshabilitado.

Profesores y Ayudantes

El usuario con permisos de administrador es el único que puede asignar usuarios como profesor o ayudante dentro de un curso. Un usuario no puede tener dos roles en el mismo curso.

Los roles pueden ser desasignados con el botón Eliminar.

Figura 8.6: Profesores y ayudantes de un curso.

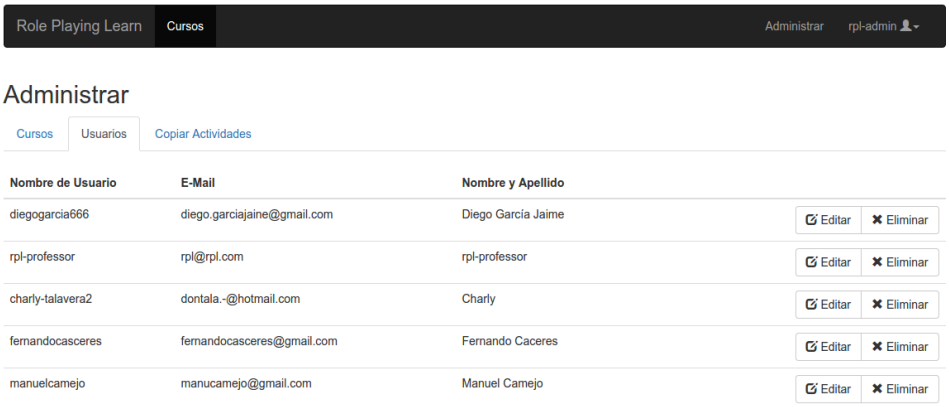


8.6.2. Usuarios

En el tab de usuarios el administrador puede editar y eliminar usuarios.

Listado de usuarios

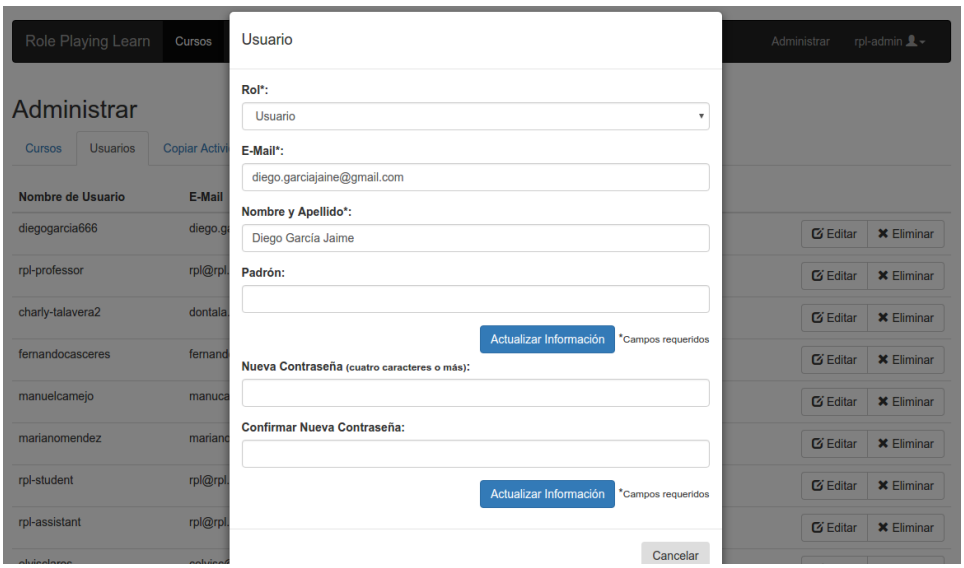
Figura 8.7: Listado de usuarios.



Editar usuario

Para editar un usuario se debe hacer click en el botón Editar. Se pueden editar varios campos e inclusive cambiar la contraseña.

Figura 8.8: Editar usuario.



Eliminar usuario

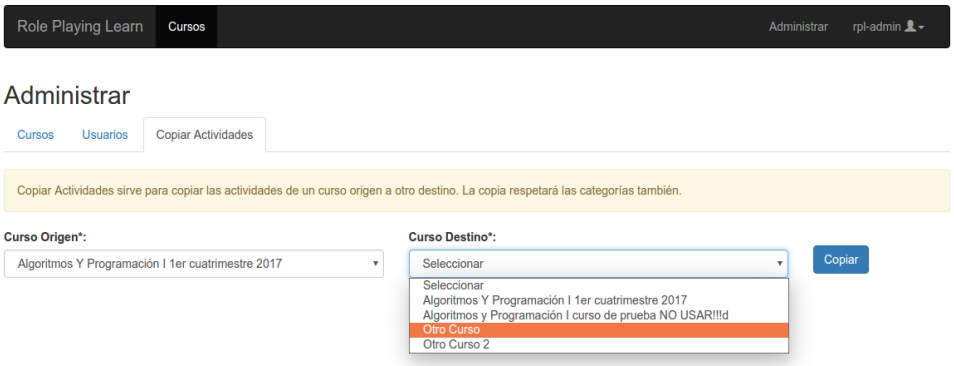
Para eliminar un usuario se debe hacer click en el botón Eliminar. Se pide una confirmación antes de eliminar un usuario.

8.6.3. Copiar actividades

La plataforma permite copiar las actividades de un curso a otro curso diferente. La copia realizada contendrá las mismas categorías y actividades que el curso origen.

Se debe seleccionar el curso origen y el curso destino.

Figura 8.9: Copiar actividades.



8.7. Como Profesor o Ayudante de un Curso

El profesor de un curso puede realizar ciertas operaciones relacionadas con la administración de un curso. Estas operaciones pueden estar vinculadas con las actividades, las categorías, la personalización o los reportes de un curso.

Notar que los ayudantes pueden realizar las mismas operaciones excepto la aceptación de alumnos y la personalización del curso. Además tienen ciertas limitaciones a la hora de ver los reportes (en general, los ayudantes sólo pueden ver las información de los alumnos que tienen asignados).

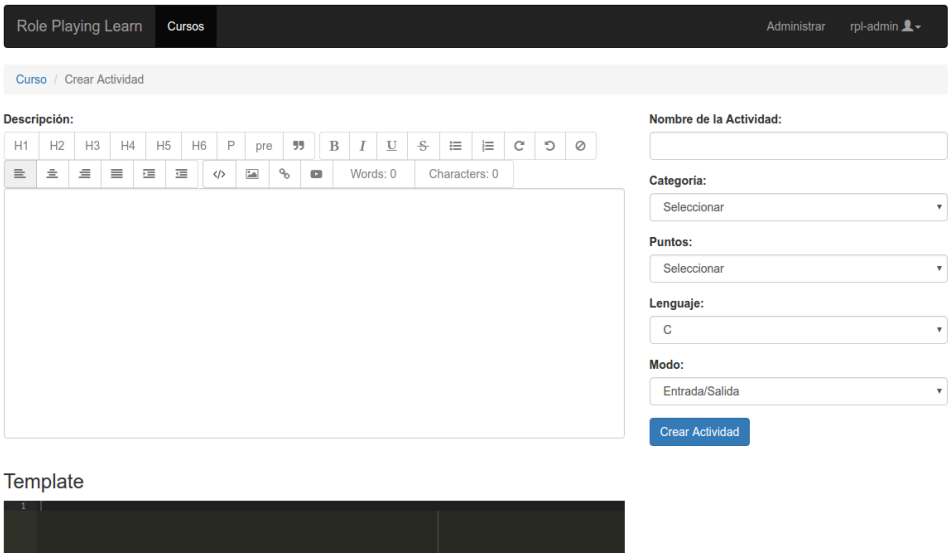
8.7.1. Actividades

Listado de Actividades

Figura 8.10: Listado de Actividades.

Crear Actividad

Figura 8.11: Crear actividad.



Descripción

La misma se muestra cuando el alumno ingresa a la actividad. Permite la inclusión de imágenes y de texto con formato.

Nombre de la Actividad

Nombre que corresponde a la actividad.

Puntos
Puntos que suma el alumno si resuelve correctamente la actividad.
Lenguaje
Lenguaje de programación de la actividad.
Modo

- Entrada/Salida: la salida del envío del alumno se va a comparar con una especificada. Si se selecciona esta opción se debe especificar la entrada que se le va a proporcionar al programa (*standard input*) y la salida esperada (por *standard output*).

- Pruebas: el profesor debe escribir las pruebas unitarias que se deben correr en cada caso en el apartado "Pruebas".

Template
Código por defecto que le va a aparecer al alumno. Es recomendable que incluya comentarios y, en lo posible que ya compile.

Se recomienda ver la sección llamada "Ejemplos de actividades" con ejemplos disponibles para cada lenguaje.

Editar actividad

Para editar una actividad se debe hacer click en el botón Editar.

Eliminar actividad

Para eliminar una actividad se debe hacer click en el botón Eliminar. Se pide una confirmación antes de eliminar una actividad.

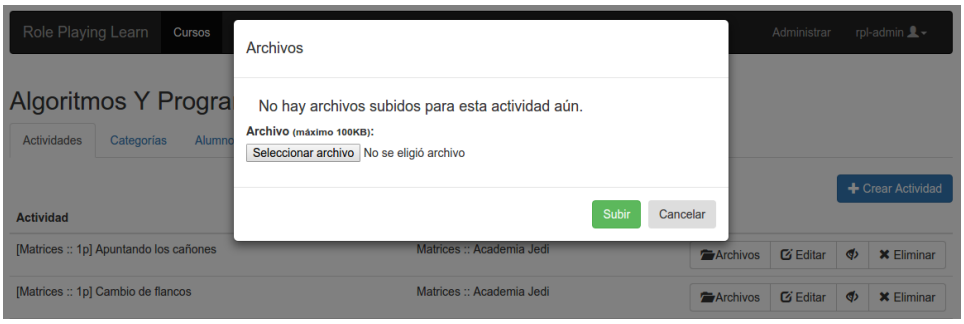
Habilitar/Deshabilitar actividad

Existe la opción de habilitar/deshabilitar una actividad haciendo click en el icono en forma de Ojo. El nombre de la actividad aparece grisado si se encuentra deshabilitada.

Archivos de una actividad

Existe la opción subir archivos a una actividad. Estos archivos estarán disponibles en el entorno donde se corre la actividad con el mismo nombre con el que se subieron.

Figura 8.12: Archivos de una actividad.

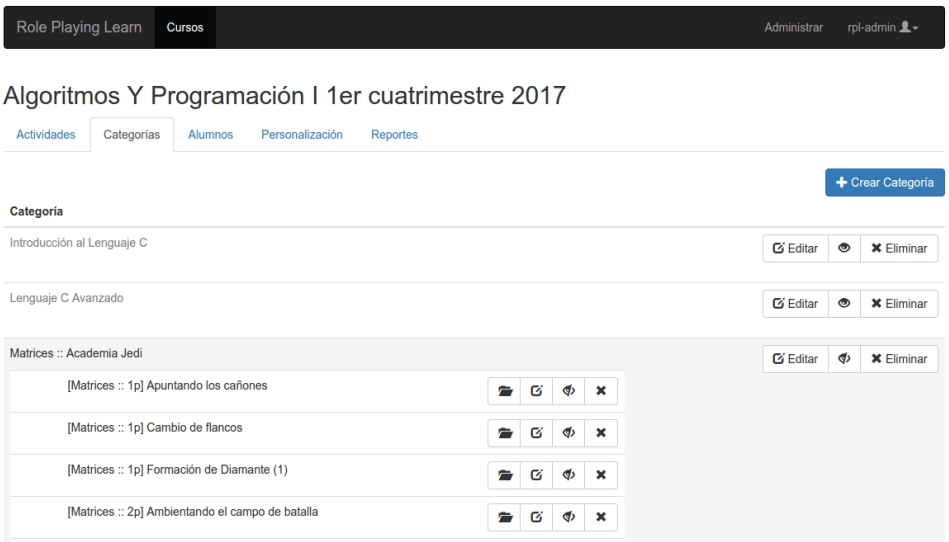


8.7.2. Categorías

Listado de Categorías

Muestra un listado de las categorías con sus respectivas actividades.

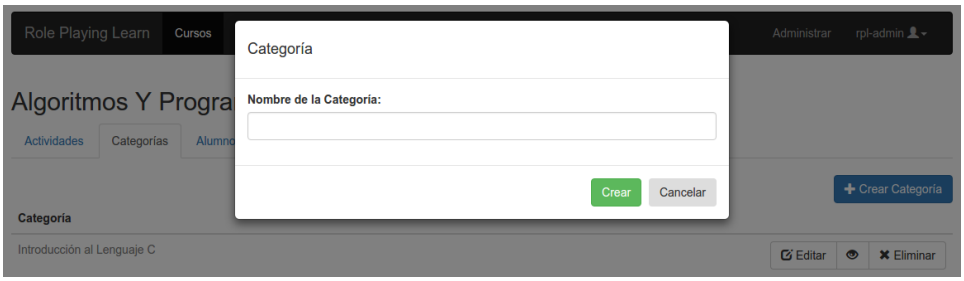
Figura 8.13: Listado de Categorías.



Crear Categoría

Sólo se debe escribir el nombre de la nueva categoría a la hora de crear una.

Figura 8.14: Crear categoría.



Editar categoría

Para editar una categoría se debe hacer click en el botón Editar.

Eliminar categoría

Para eliminar una categoría se debe hacer click en el botón Eliminar. Se pide una confirmación antes de eliminar una categoría.

Habilitar/Deshabilitar categoría

Existe la opción de habilitar/deshabilitar una categoría haciendo click en el icono en forma de Ojo. El nombre de la categoría aparece grisado si se encuentra deshabilitada.

8.7.3. Alumnos

Lista los alumnos del curso. Se puede aceptar o rechazar las inscripción de un alumno. También se puede asignarle un ayudante para que pueda hacerle un seguimiento más fino a través de los reportes.

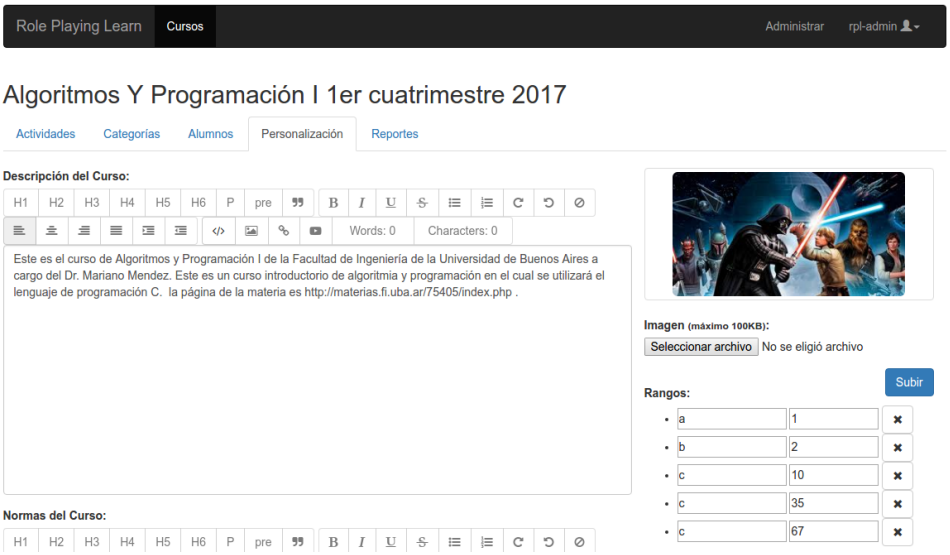
Si se rechaza el alumno por error, el alumno puede volver a solicitar la inscripción al curso. No se pierde la información del mismo.

Figura 8.15: Alumnos.

8.7.4. Personalización

El curso permite personalizar varios aspectos de la estética del curso.

Figura 8.16: Personalización.



Descripción del curso

La misma se muestra cuando el alumno ingresa al curso (una vez ya aceptado). Permite la inclusión de imágenes y de texto con formato.

Normas del curso

El alumno antes de inscribirse a un curso debe aceptar las normas del curso. Permite la inclusión de imágenes y de texto con formato.

Imagen del curso

Se permite subir una imagen para representar el curso. La imagen se puede ver en donde se muestran todos los cursos.

Rangos

Los rangos representan la leyenda que se va a mostrar al lado de cada estudiante, según su puntaje.

A la izquierda debe ir la leyenda y a la derecha el puntaje mínimo para cumplir con ese rango.

Por ejemplo:

- Aprendiz - 0
- Novato - 30
- Experto - 100

CSS personalizado

Es posible definir un CSS personalizado para el curso. Se dispone de las siguientes clases de CSS para modificar, entre otras: *.customization*, *.customization-container*, *.customization-course-name*, *.customization-topic-name*, *.customization-list-group*, *.customization-list-group-item*, *.customization-badge*, *.customization-activity-name* y *.customization-ranking*.

Por ejemplo, el siguiente CSS muestra una imagen de fondo:

```
.customization {
  background: #000000 url("https://example.com/imagen.jpg") no-repeat center;
  color: white;
  padding: 20px;
}
```

8.7.5. Reportes

A través del tab de Reportes se puede ingresar a los mismos.

Por Alumno y Categoría

Se debe seleccionar un alumno, la categoría es opcional.

El reporte permite visualizar las actividades del alumno, si fueron resueltas o no y el historial de envíos.

Asimismo, se dispone de un gráfico que dada una categoría muestra las actividades ordenadas por dificultad (puntaje) en el eje X y la cantidad de envíos en el eje Y. Se representa con un punto de color verde si la actividad fue resuelta y con uno de color rojo si no fue así. Este gráfico sólo es visible si se especifica una categoría.

Figura 8.17: Por Alumno y Categoría.

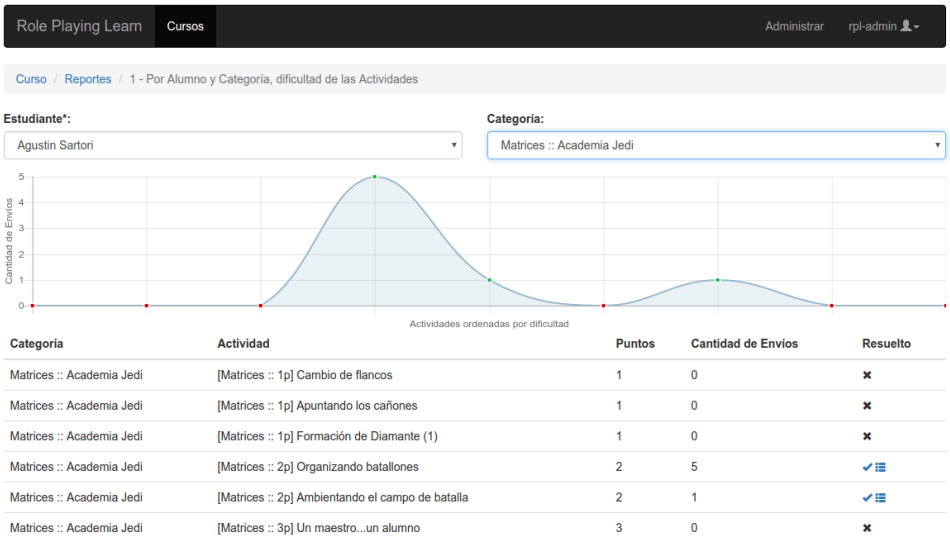
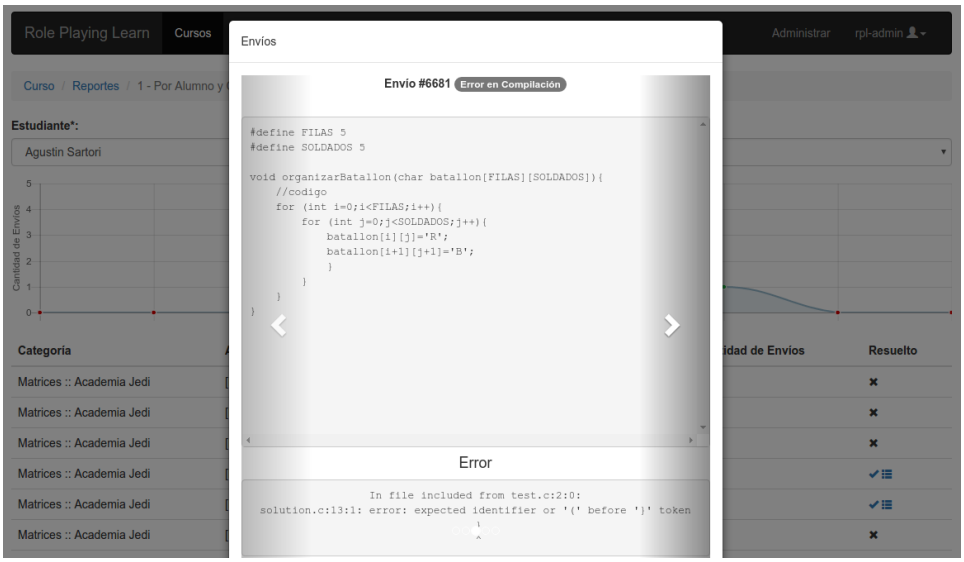


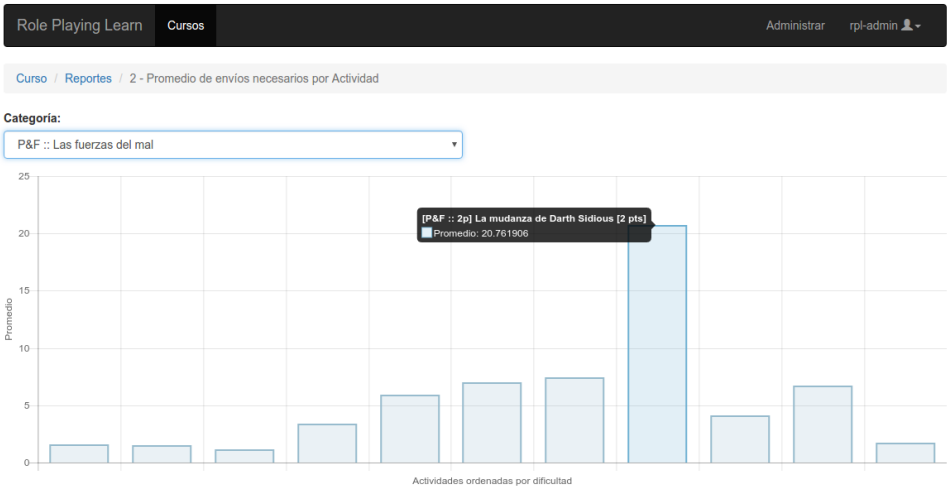
Figura 8.18: Historial de envíos.



Promedio de envíos necesarios por Actividad

Dada una categoría, el gráfico muestra en el eje X las actividades ordenadas por dificultad (puntaje) y en el eje Y la cantidad promedio de envíos necesarios para resolver la actividad (se toman sólo las soluciones definitivas).

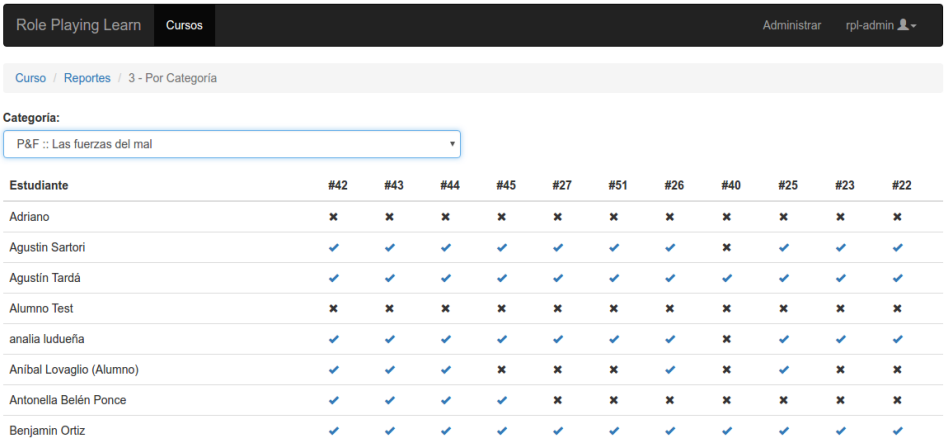
Figura 8.19: Promedio de envíos necesarios por Actividad.



Por Categoría

Dada una categoría, se dispone de una tabla con todos los alumnos y las actividades de esa categoría. Permite visualizar quién realizó cada actividad y ver la solución enviada.

Figura 8.20: Por Categoría.

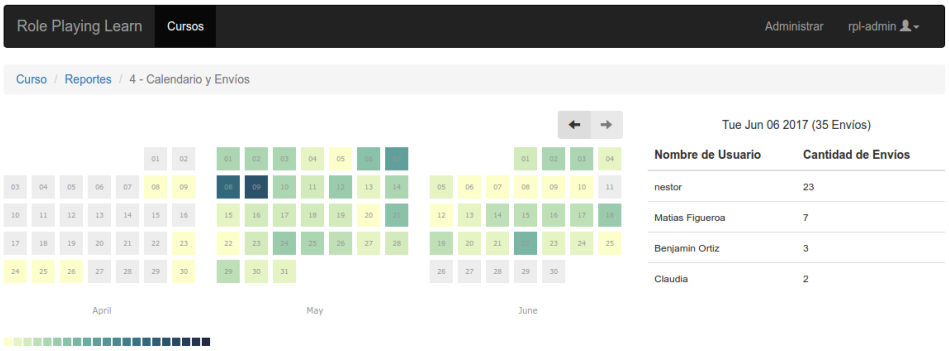


Calendario y envíos

El calendario permite visualizar la actividad desde el primer envío de una actividad hasta el último.

Se representa la cantidad de envíos diarios con una escala de colores lineal. Haciendo click en cierto día se puede ver el detalle de quiénes participaron en los envíos.

Figura 8.21: Calendario y envíos.



Alumnos y porcentaje de completado

Dada una categoría, se dispone de una tabla que muestra los alumnos y qué porcentaje de las actividades llevan hechas. Se puede especificar un porcentaje de corte.

Figura 8.22: Alumnos y porcentaje de completado.

Role Playing Learn | Cursos | Administrar | rpl-admin

Curso / Reportes / 5 - Alumnos y porcentaje de completado

Categoría: P&F :: Las fuerzas del mal

Porcentaje Base (%): 45

Nombre y Apellido	Porcentaje
Schischlo Franco Daniel	100.00%
Juan Cruz Bossio	100.00%
Delfina García Villamor	100.00%
Francisco Nasif	100.00%
Rodrigo Souto	100.00%
daphne marcela hernandez	100.00%
Gonzalo Menese	100.00%
Benjamin Ortiz	100.00%

Ranking por fecha

Este reporte permite ver el ranking del curso hasta determinada fecha.

Figura 8.23: Ranking por fecha.

Role Playing Learn | Cursos | Administrar | rpl-admin

Curso / Reportes / 6 - Ranking por fecha

Ranking hasta la fecha*: 2017-07-11

	Nombre de Usuario	Puntos
1.	Nicolas Bertillod	83 pts
2.	Juan Pablo Vallejo	83 pts
3.	Rodrigo Souto	83 pts
4.	Facundo de la Plata	83 pts
5.	Delfina García Villamor	74 pts
6.	German Godoy	72 pts
7.	Matias Figueroa	71 pts
8.	Sebastian B. Inneo V.	70 pts

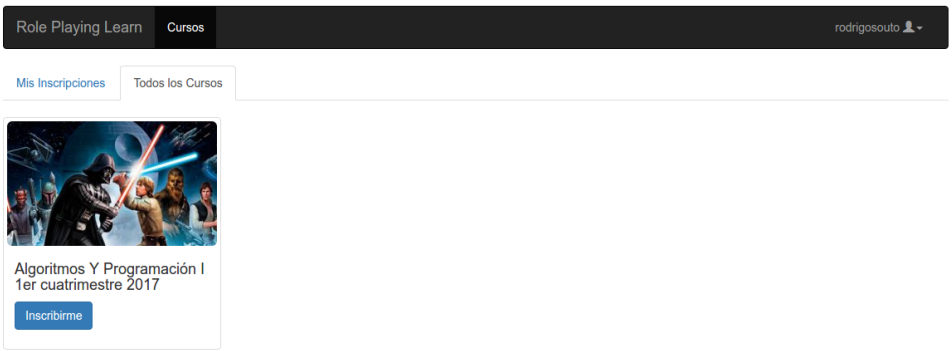
8.8. Como Alumno

8.8.1. Inscripción en un curso

Si el alumno no está inscripto en ningún curso va a visualizar una leyenda indicándolo.

En el tab de "Todos los cursos" puede ver los cursos que aceptan inscripción. Al hacer click en "Inscribirme", la misma queda pendiente hasta que un profesor del curso la apruebe. Es necesario aceptar las normas del curso para poder inscribirse en un curso.

Figura 8.24: Inscripción en un curso.



8.8.2. Dentro de un curso

Una vez dentro del curso, el alumno puede ver el ranking de los demás estudiantes y todas las actividades disponibles, organizadas en categorías.

El ranking muestra a los alumnos del curso ordenados por puntaje. Junto al nombre del estudiante se muestra su rango según el puntaje.

Figura 8.25: Dentro de un curso.

8.8.3. Resolución de una actividad

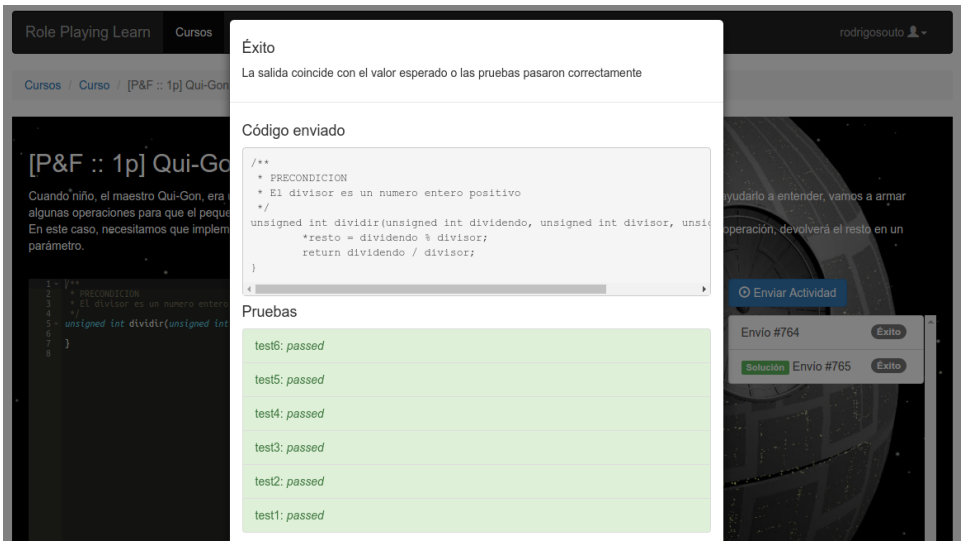
La resolución de la actividad consiste en qué la actividad pase las pruebas unitarias o que la salida del programa coincida con la esperada.

El objetivo de la actividad está especificado en la descripción. Una vez que el alumno considere que ha finalizado la actividad puede enviarle mediante el botón "Enviar Actividad". Hay que esperar unos minutos y ya se obtiene un resultado. En caso de que haya algún error o falle en alguna etapa, se le indica al estudiante.

Figura 8.26: Resolución de una actividad.

También es posible ver el historial de envíos haciendo click en alguno.

Figura 8.27: Historial de una actividad.



Si el resultado del envío es exitoso, se podrá marcar como solución. Más aún, es posible ver las resoluciones de otros estudiantes si una solución se marca como definitiva. Hay que tener en cuenta que una vez marcada como definitiva no se puede cambiar.

Capítulo 9

Ejemplos de Actividades

9.1. Introducción

El objetivo de esta sección es brindar un ejemplo simple para cada lenguaje soportado.

En todos los casos el objetivo de la actividad brindada es resolver la suma de dos números.

9.2. C

9.2.1. Pruebas

Para soportar las pruebas se utiliza la versión v2.2.2 de la librería *Criterion*. Se puede consultar la documentación oficial en su web[1].

Template

```c
#include <stdio.h>

int suma(int numero1, int numero2) {
        // Codigo del alumno
        return 0;
}
```

En el template basta con dejar definidas las funciones que van a ser utilizadas en las pruebas.

Pruebas

```c
#include <criterion/criterion.h>
#include "solution.c"
```

[1] https://github.com/Snaipe/Criterion

```
Test(misc, test1) {
    cr_assert(suma(2,3) == 5);
}

Test(misc, test2) {
    cr_assert(suma(10,15) == 25);
}
```

En las definición de las pruebas la inclusión de la librería *Criterion* y del archivo *solution.c* es obligatoria. La definición de cada *Test()* consta de dos argumentos. El primero, en general, no debe variar, mientras que el segundo es el nombre que describe la prueba unitaria y figura cuando el alumno resuelve las actividades.

Solución posible

```
#include <stdio.h>

int suma(int numero1, int numero2) {
    return numero1 + numero2;
}
```

9.2.2. Entrada/Salida

Template

```
#include<stdio.h>

int main() {
    // Codigo del alumno
    return 0;
}
```

En el template basta con definir un main principal.

Entrada proporcionada

```
2
3
```

Salida esperada

```
5
```

Solución posible

```c
#include <stdio.h>

int main() {
    int numero1;
    int numero2;

    scanf("%d",&numero1);
    scanf("%d",&numero2);

    printf("%d", numero1 + numero2);
}
```

9.3. Java

9.3.1. Pruebas

Para soportar las pruebas se utiliza JUNIT[2].

Template

```java
public class Solution {

    public int sumar(int numero1, int numero2) {
        // Codigo del alumno
        return 0;
    }

}
```

Notar que la clase se debe llamar *Solution* y los métodos deben ser *públicos*.

Pruebas

```java
import org.junit.Test;

import static org.junit.Assert.assertTrue;

public class TestSolution {

    @Test
    public void test1() {
        Solution solution = new Solution();
```

[2] http://junit.org/

```
        assertTrue(solution.sumar(2, 3) == 5);
    }

    @Test
    public void test2() {
        Solution solution = new Solution();
        assertTrue(solution.sumar(10, 15) == 25);
    }

}
```

En las definición de las pruebas la inclusión de la librería *org.junit.Test* es obligatoria. Asimismo es importante notar la presencia de la *annotation* @Test y respetar las convenciones propuestas por JUNIT.

Se debe instanciar la clase *Solution* e invocar su método.

Solución posible

```
public class Solution {

    public int sumar(int numero1, int numero2) {
        return numero1 + numero2;
    }

}
```

9.3.2. Entrada/Salida

Template

```
import java.util.Scanner;

public class Solution {

    public static void main(String[] args) {
            // Codigo del alumno
        Scanner scan = new Scanner(System.in);
        int numero1 = scan.nextInt();
    }
}
```

En el template basta con definir un main principal.

Entrada proporcionada

```
2
3
```

Salida esperada

```
5
```

Solución posible

```java
import java.util.Scanner;

public class Solution {

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int numero1 = scan.nextInt();
        int numero2 = scan.nextInt();

        System.out.println(numero1 + numero2);
    }
}
```

9.4. Python

9.4.1. Pruebas

Para soportar las pruebas se utiliza unittest[3].

Template

```python
def sumar(numero1, numero2):
        # Codigo del alumno
        return 0
```

Notar que la clase se debe llamar *Solution* y los métodos deben ser *públicos*.

Pruebas

```python
import unittest
import solution

class TestMethods(unittest.TestCase):
```

[3] http://junit.org/

```
    def test1(self):
        self.assertTrue(solution.sumar(2, 3) == 5)

    def test2(self):
        self.assertTrue(solution.sumar(10, 15) == 25)
```

En las definición de las pruebas la inclusión de la librería *unittest* es obligatoria. Asimismo es importante incluir *solution* y respetar las convenciones propuestas por *unittest*.

Solución posible

```
def sumar(numero1, numero2):
        return numero1 + numero2
```

9.4.2. Entrada/Salida

Template

```
import fileinput

# Codigo del alumno
for line in fileinput.input():
    print line
```

En el template basta con definir un main principal.

Entrada proporcionada

```
2
3
```

Salida esperada

```
5
```

Solución posible

```
import fileinput

total = 0
for line in fileinput.input():
    total = total + int(line)

print total
```