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

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA PARA EL MONITOREO DEL AVANCE EN TIEMPO REAL DEL TRABAJO DE ESTUDIANTES EN LABORATORIOS

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN

SEBASTIÁN RODRIGO HERNÁNDEZ GÓMEZ

PROFESOR GUÍA:
JOSÉ PINO URTUBIA

MIEMBROS DE LA COMISIÓN:
NELSON BALOIAN TATARYAN
BENJAMÍN BUSTOS CÁRDENAS
FRANCISCO GUTIÉRREZ FIGUEROA

SANTIAGO DE CHILE
2018

RESUMEN

Se ha comprobado que la retroalimentación o feedback entregado a los estudiantes en el aula durante el proceso de aprendizaje mejora sus resultados en actividades evaluativas. Cuando se da correctamente y a tiempo, la retroalimentación guía al estudiante en su proceso de aprendizaje y le entrega la dirección que necesitan para alcanzar el objetivo o la meta de la lección. La retroalimentación o feedback es información proporcionada por un agente (por ejemplo, profesor).

Recientemente se ha visto un aumento en la adopción de programas computacionales para potenciar el aprendizaje. Tal es caso del curso CC1000 Herramientas Computacionales para Ingeniería y Ciencias dictado en la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile. En este curso, las herramientas computacionales son precisamente el objeto de estudio.

El uso de tecnologías en la sala de clases configura un escenario propicio para la implementación de sistemas informáticos que permitan al profesor monitorear de manera automática, no intrusiva y en tiempo real el progreso realizado por estudiantes en sus computadores. Este monitoreo en tiempo real permitirá al profesor entregar feedback oportuno para guiar el aprendizaje de sus estudiantes.

En este trabajo se muestra la propuesta y el desarrollo de una herramienta que permita al profesor monitorear el avance de los estudiantes durante las sesiones presenciales de laboratorio en las que se trabaja sobre documentos activos, por ejemplo Matlab y Excel. El profesor podrá medir el avance de los alumnos en completar las tareas propuestas así como la correctitud de las respuestas dadas, gracias a indicadores relevantes que le serán entregados a través de una interfaz de usuario. Los datos pertinentes serán obtenidos en tiempo real consultando cada uno de los computadores utilizados por los estudiantes para realizar el laboratorio y contrastándolos con una pauta de evaluación elaborada previamente por el profesor. La solución será probada en sesiones de laboratorio del curso CC1000 Herramientas Computacionales para Ingeniería y Ciencias.

Se pretende además que la solución sea extensible y aplicable a otros programas que trabajen con documentos activos o sesiones con las que se pueda interactuar desde aplicaciones externas, por ejemplo el entorno de desarrollo de Python.

A mis padres, Elizabeth y Francisco, los amo.

Agradecimientos

Quiero agradecer a mis profesores José y Nelson, quienes con mucha paciencia y confianza me han guiado y han sabido mantenerme motivado a lo largo de este trabajo. Además, agradecer a los profesores Patricio Poblete y Maira Marques, quienes con excelente disposición me permitieron probar el sistema desarrollado en el curso Herramientas Computacionales dictado en escuela de verano 2018.

A toda mi familia, en especial madre y padre quienes me han apoyado de manera incondicional durante toda la vida, en todo momento, ya sea bueno o malo, los amo. A mis amigos, con quienes siempre he podido contar en este extenso tiempo dentro de la universidad.

A todos los profesores de quienes he tenido el privilegio de aprender.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Introducción

En las clases presenciales de laboratorio[1] es importante monitorear el trabajo de los estudiantes para asegurarse de que reciban el feedback adecuado a tiempo. El feedback se ha considerado como parte fundamental para el proceso de aprendizaje [6] y es importante que el profesor identifique cómo y cuándo entregarlo [7]. Sin embargo, monitorear el avance de cada estudiante puede llegar a consumir mucho tiempo de la clase, sobre todo si se trata de un curso con una gran cantidad de estudiantes. Considérese una clase de computación con 60 estudiantes en la que, luego de que el profesor enseñara algunas operaciones sobre matrices, los estudiantes deban resolver una serie de problemas escribiendo código en Matlab. El profesor entonces quisiera conocer el desempeño de los alumnos durante el desarrollo de la sesión para poder intervenir en caso de ser necesario, sin que esto consuma mucho tiempo. Para esto, sería ideal una herramienta que permitiera monitorear el avance de manera automática de cada estudiante y que se lo muestre al profesor en términos de métricas relevantes que le permitan asistir a los estudiantes en aquello que les está costando en el momento preciso.

La educación puede darse de manera presencial en salas de clases, de manera remota a través de Internet o bien una combinación entre las dos. Cuando se realiza de manera remota, el alcance de alumnos puede ser tan alto como el sistema soporte, pudiendo llegar a miles de estudiantes. En contraste, la educación presencial en sala de clases está restringida por factores físicos como el espacio, acceso al profesor, etc. En este trabajo de título se considerarán laboratorios presenciales en los que participan aproximadamente 50 estudiantes de manera simultánea.

El uso de tecnologías en la sala de clases configura un escenario propicio para la implementación de sistemas informáticos que solucionen el problema del monitoreo del avance en tiempo real. De hecho existen trabajos en los que se han implementado soluciones informáticas para monitorear el trabajo de los estudiantes [1]. En el pasado, se ha desarrollado material de aprendizaje en forma de hojas de trabajo electrónicas para implementar actividades de aprendizaje en clase para los estudiantes [9], [8], [11], [10]. Estos materiales se han denominado "Documentos Activos". Estos documentos activos proporcionan a los estudiantes un buen entorno para interactuar. Además, permiten el trabajo colaborativo haciendo uso de las redes disponibles. En la mayoría de los casos, se han utilizado documentos del tipo XML como una forma de manipular estos documentos y almacenarlos en dispositivos externos. Si los estudiantes trabajan en estos documentos electrónicos, modificando su contenido, entonces es posible hacer un análisis automático y, por tanto, sistemático de su trabajo. Por ejemplo,

[1]Instancias en las que el alumno debe hacer algún trabajo o resolver problemas en forma individual o grupal, guiado por el profesor

el análisis se puede utilizar para averiguar cómo avanzan los estudiantes en la realización de las tareas descritas en el documento activo, si están o no llenando el documento con las respuestas correctas, etc.

Actualmente existe un curso dictado en la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile llamado Herramientas Computacionales para Ingeniería y Ciencias (CC1000) en el que se enseña el uso de múltiples herramientas computacionales tales como Excel, Matlab, Maple, Word, Latex, uso de Blogs, entre otras, siendo Excel y Matlab el principal foco, ocupando aproximadamente dos tercios del curso. En clases, el profesor presenta brevemente los contenidos necesarios para resolver un problema, luego los estudiantes lo resuelven con el uso de computadores y la ayuda del profesor. Como se señaló anteriormente, resulta muy importante para el profesor monitorear el avance de los estudiantes a medida que se desarrolla el laboratorio, ya que esto le permitirá identificar aquellos conceptos que debe reforzar y a qué ritmo llevar la sesión de laboratorio.

En este trabajo de título se pretende diseñar e implementar un sistema que permita a profesores monitorear el avance en tiempo real del trabajo de los estudiantes en clases presenciales de laboratorio del curso CC1000, analizando sus documentos electrónicos del tipo Excel y Matlab. El sistema será desarrollado para ser usado en los laboratorios del curso Herramientas Computacionales para Ingeniería y Ciencias.

0.1. Motivación

Actualmente, para medir en tiempo real el progreso de los estudiantes en los laboratorios, el profesor auxiliar debe pasar por cada puesto consultando uno por uno a los estudiantes. Considerando un curso con 50 personas y un tiempo de consulta de 1 minuto por persona, realizar esta tarea a cabalidad tomaría unos 50 minutos, lo que consumiría más de un 50 % del tiempo de clases, cuya duración es de 1 hora y 30 minutos cada sesión. Otra forma para monitorear el avance es haciendo una pregunta general al curso, la que es menos exacta porque es probable que estudiantes que no estén teniendo un buen desempeño omitan su respuesta, o bien no respondan de manera consistente con la realidad por factores sociales. Podrían también integrarse auxiliares que realicen el trabajo de monitoreo personalizado, pero esto significaría costos adicionales para el curso.

Debe considerarse que en los laboratorios se enseñan diferentes herramientas, las que requerirán diferentes formas de medir progreso. Por ejemplo, trabajando con Excel, una forma de medir el progreso podría ser comparando el valor en determinada celda con algún resultado esperado, mientras que al evaluar un código escrito en Matlab, por ejemplo alguna función, podrían utilizarse baterías de tests.

En trabajos previos se han desarrollado prototipos de sistemas que monitorean el avance en los laboratorios del curso Herramientas Computacionales para Ingeniería y Ciencias, pero hasta ahora no han sido implantados, principalmente por dificultades de usabilidad [3][4]. Estos trabajos han servido como pruebas de concepto para validar la factibilidad tecnológica de una potencial solución al problema de monitoreo en tiempo real. Sin embargo, la forma en

la que se implementaron requiere tiempo para instalar e inicializar el sistema y que el profesor usuario tenga conocimiento especializado. La herramienta a desarrollar debe intervenir lo menos posible en el flujo natural de la clase. El profesor no puede desviar su atención para dedicarse a operar la herramienta, que de hecho está pensada para ahorrarle tiempo; debiera ser lo más automática y transparente posible.

0.2. Objetivos de la Memoria

El objetivo general de la memoria es realizar el diseño e implementación de un sistema que permita a profesores del curso CC1000 dictado en la FCFM monitorear el avance en tiempo real de los estudiantes en los laboratorios.

Para lograr esto, se proponen los siguientes objetivos específicos a ser cumplidos en este trabajo de memoria:

- Establecer métodos para evaluar el progreso sobre los archivos estructurados en los que el estudiante desarrolla actividades de Excel y Matlab, que como se señaló anteriormente, abarcan la mayor parte de las actividades de laboratorio del curso.
- Establecer los indicadores relevantes para el docente que le permitan medir el progreso de los estudiantes.
- Desarrollar una herramienta del tipo cliente-servidor que permita al profesor monitorear el avance sobre el trabajo realizado por los estudiantes en computadores sobre archivos estructurados, entregando esta información como indicadores en un panel.
- La solución debe ser capaz de soportar el monitoreo sobre 30 computadores de manera simultánea, que serán ocupado por uno o dos estudiantes cada uno.
- El sistema debe entregar flexibilidad al profesor para adaptar el monitoreo de avance a nuevos laboratorios, ya que estos pueden cambiar cada semestre.

0.3. Idea General de la Solución

Como se mencionó anteriormente, el uso de documentos activos por parte de los estudiantes para desarrollar actividades de laboratorio permite hacer un análisis automático y, por tanto, sistemático de su trabajo. El análisis se puede utilizar para averiguar cómo avanzan los estudiantes en la realización de las tareas en el documento activo, si están o no llenando el documento con las respuestas correctas, etc.

En este trabajo se pretende desarrollar una herramienta que permita al profesor monitorear el avance de los alumnos durante las clases de laboratorio presenciales en los que se trabaja sobre documentos de formato Matlab y Excel. El profesor podrá medir el avance gracias a indicadores relevantes que serán mostrados en su pantalla, los cuales serán obtenidos en tiempo real consultando cada uno de los computadores utilizados por los estudiantes para realizar el laboratorio.

Concretamente, el sistema desarrollado permitirá responder a preguntas como ¿Están teniendo problemas los estudiantes en el desarrollo del laboratorio?, ¿En qué preguntas están teniendo problemas?, ¿Cuántos alumnos han resuelto hasta determinada pregunta del laboratorio?, ¿Cuántos de ellos respondieron correctamente a cierta pregunta?, entre otras. Por ejemplo, en el caso de Excel, si en el enunciado del laboratorio se indica que la respuesta a una determinada pregunta debe ponerse en una celda específica, bastará comparar el valor esperado con el valor de esa celda en cada uno de los computadores para responder a la pregunta de "¿Cuántas personas obtuvieron un resultado correcto en tal pregunta?". Más aún, se podrían gatillar eventos cuando se cumplan ciertas condiciones como por ejemplo, cuando hayan 30 alumnos que hayan completado una determinada celda, notificar al profesor y mostrar la distribución de las respuestas. Responder este tipo de preguntas permitirá al profesor saber cuándo deberá intervenir para aclarar ciertos conceptos. Por ejemplo, si se observa que casi nadie logró responder correctamente una determinada pregunta cuya resolución es fundamental para abordar las preguntas posteriores, entonces sería preciso intervenir y entregar la retroalimentación pertinente para que no tengan problemas al resolver las preguntas venideras. Esta información también será valiosa para mejorar el laboratorio como tal, identificar posibles cambios que podrían hacerse para que los estudiantes lo resuelvan de manera más ágil y no se queden estancados.

En general, las preguntas que uno podrá responder con el sistema a desarrollar serán:

- ¿Qué preguntas han respondido los estudiantes?
- ¿Cómo las han respondido? (Bien o mal)
- ¿Cuántos han respondido cierta pregunta?

Capítulo 1

Marco Teórico

En este capítulo se presentan los antecedentes necesarios para una completa comprensión del presente trabajo de título. Para esto, se explica el contexto en el que se desarrollan las sesiones de laboratorio y los componentes que eso involucra.

1.1. Sala Galileo

Las actividades de laboratorio realizadas en el curso CC1000 tienen lugar en la sala Galileo, un espacio de 300 m^2 equipado con 10 mesas de trabajo, las que cuentan con computadores para uso de los estudiantes. Además cuenta con un computador administrado por el profesor auxiliar a cargo del laboratorio que le permite controlar las sesiones de cada computador.

Figura 1.1: Sala Galileo.

Todos los computadores se encuentran interconectados en una red y además Internet, como se muestra en el diagrama de la figura 1.2. Esta red permite a cada computador configurar ciertas carpetas para que los demás computadores tengan permisos de lectura o escritura sobre ellas. Por ejemplo, podría establecerse que determinada carpeta de los computadores usados

por los estudiantes pueda ser accesada por el computador del profesor auxiliar. Además, podría indicarse a los alumnos que deben realizar su trabajo, es decir, crear sus archivos de Excel y Matlab en dicha carpeta. Con esto, el computador del profesor auxiliar ganaría el acceso sobre los documentos de trabajo de los estudiantes, permitiendo así el monitoreo de avance.

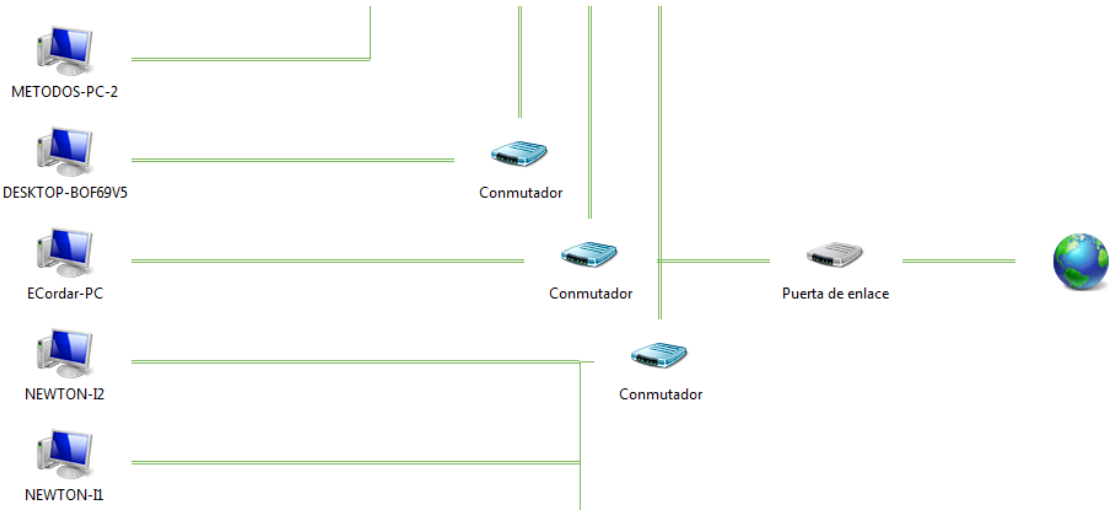

Figura 1.2: Diagrama de red en Sala Galileo.

Cada computador funciona con el sistema operativo Windows 7 Professional de 64 bits, procesador Intel Core i5 con 3.1 GHz, 8GB de RAM. Cuentan con Excel 2013 y Matlab 2015a. Cada computador es identificado en la red por su nombre, que está compuesto por el prefijo "NEWTON-" seguido de una letra, que identifica la mesa de trabajo en la que se ubica el computador y un número, que identifica al computador dentro de la mesa de trabajo. Por ejemplo, "NEWTON-E2" identifica al computador número 2 de la mesa de trabajo E.

1.2. Laboratorios

En cada laboratorio se le entrega a cada estudiante un documento con problemas que debe resolver.

1.2.1. Excel

De manera general, en los laboratorios de Excel los alumnos deben realizar análisis sobre datos, construir modelos matemáticos, representar datos de manera gráfica y resolver problemas interesantes tales como encontrar la recta de mínimos cuadrados sobre un conjunto de puntos en el plano, etc. Para llevar a cabo estas actividades, los alumnos deben realizar diferentes operaciones en Excel, tales como las que se resumen en la siguiente lista:

- Tabular datos

- Realizar cálculos con las funciones de Excel a partir de los valores de otras celdas
- Crear gráficos
- Ordenar tablas según valores de cierta columna
- Crear tablas dinámicas

Al final de cada sesión de laboratorio, el alumno debe entregar el archivo Excel en el que trabajó, a partir del cual será evaluado.

1.2.2. Matlab

Al igual que en el caso de los laboratorios de Excel, en los laboratorios de Matlab los alumnos deben realizar análisis sobre datos, construir modelos matemáticos, representar datos de manera gráfica y resolver problemas interesantes. Para llevar a cabo estas actividades, los alumnos deben realizar diferentes operaciones en Matlab, tales como las que se resumen en la siguiente lista:

- Definir variables con cierto nombre de diferentes tipos (vector, numérico, etc)
- Calcular nuevos valores a partir de los definidos previamente usando las funciones de Matlab
- Graficar vectores con determinado título, leyenda, etiqueta para los ejes
- Guardar los gráficos en un archivo de imagen
- Definir funciones de Matlab
- Crear archivos de Matlab con funciones
- Escribir scripts que interactúen con el usuario (con el comando input, para solicitar ingreso de valores por parte del usuario)
- Mostrar mensajes con la función `disp()`
- Modificar funciones dadas para obtener diferentes comportamientos solicitados
- Crear imágenes usando el comando `colormap` que cumplan ciertas características

Los entregables de estas actividades pueden ser las imágenes generadas a partir de los gráficos, documentos Word con las respuestas a los problemas, o bien documentos de extensión `.m` con las funciones y scripts de Matlab.

De manera general se describieron las tareas que debe realizar cada estudiante para resolver las actividades de laboratorio de Excel y Matlab. Con esto, es posible establecer la forma en que se medirá el avance. Por ejemplo, en el laboratorio nº 4 del curso CC1000, sección 7, primer semestre de 2017, se pedía calcular la recta de mínimos cuadrados a partir de una determinada tabla con pares ordenados `(x,y)` utilizando Excel. Para esto, se pedía calcular x^2, y^2, $x*y$ y poner los resultados en columnas adyacentes a la tabla inicial. Luego, se pedía calcular la sumatoria sobre las nuevas columnas y a partir de esto calcular los valores que definen la recta. En este caso, la solución a desarrollar para el monitoreo de avance debe permitir al profesor indicar que en cierta celda, fila o columna de los documentos en los que trabaja cada estudiante se esperan valores calculados a partir de otras celdas, filas o columnas.

De esta manera, el software compararía los valores esperados con los de los documentos de los estudiantes para medir el avance.

Otro ejemplo es la creación de gráficos. En muchos de los ejercicios se pide al estudiante crear gráficos a partir de tablas de datos. Lo que el profesor necesita es verificar que el estudiante haya ingresado de manera correcta los valores de título, leyenda y etiqueta de los ejes para los gráficos solicitados. Entonces, la solución a desarrollar debe permitir al profesor indicar el formato que se espera con respecto a los gráficos, detectarlos en el documento Excel del estudiante y comparar.

Otro problema, esta vez del laboratorio nº 7 de Matlab del curso CC1000, sección 7, primer semestre de 2017, consistía en crear en el archivo `direccion.m` una función llamada `direccion` que reciba un número entre 0 y 360 y entregue en forma de texto el punto cardinal al que corresponde dicho número. Una posible forma de monitorear el avance sobre funciones de Matlab podría ser estableciendo tests que la función debe acertar correctamente. Para la función `direccion`, los tests podrían haberse configurado como se ilustra en la tabla 1.1.

Tabla 1.1: Tabla de tests para la función `direccion`

Entrada	Valor esperado
0	NE
45	NE
90	SE
100	SE
180	SO
200	SO
270	NO
300	NO

De esta manera, la solución a desarrollar evaluaría las funciones escritas por los estudiantes con respecto a los valores de entrada y compararía el resultado con los valores esperados. Si, por ejemplo, la función responde correctamente a 4 de los 8 tests establecidos por el profesor, el porcentaje de avance sobre dicha función sería de 50 %, es decir, en dicho ejercicio el estudiante lleva completado el 50 %.

1.3. Pruebas con Matlab y Excel

Para monitorear el progreso del trabajo de estudiantes es necesario encontrar una manera de acceder a los documentos o instancias en la que desarrollan dicho trabajo.

Matlab En el caso de Matlab, el estudiante escribe comandos en la `Command Window` o ventana de comandos de Matlab y todas las variables que asigna son almacenadas en el `Workspace`, que corresponde a un diccionario en el que se asocia el nombre de la variable con su respectivo valor, como puede verse en la figura 1.3.

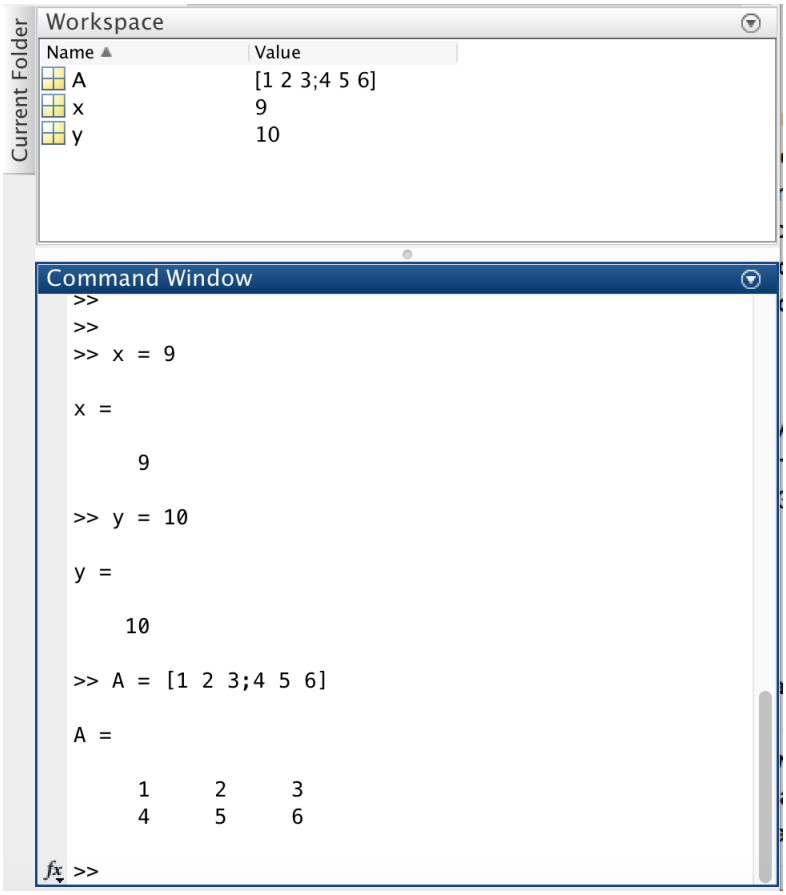

Figura 1.3: Ejemplo de asignación de variables y su respectivo Workspace.

Teniendo acceso al Workspace se puede comparar los valores esperados de cada variable contra los valores que cada estudiante obtuvo y con esto monitorear su progreso. En una instancia de Matlab, el Workspace es almacenado en memoria RAM. Una forma de acceder a éste es primero guardarlo en memoria secundaria en un determinado archivo de manera periódica en el transcurso de la sesión de laboratorio de cada estudiante y luego leer el archivo guardado. Para esto, se realizó una prueba con la función `autosave` [12] cuya definición se adjunta en el anexo. Esta función almacena las variables del workspace en un archivo, digamos `workspace.mat`, cada cierto lapso determinado de tiempo. Con esto, es posible leer desde otras aplicaciones las variables del Workspace de cada estudiante. El siguiente código escrito en Python lee las variables del archivo `workspace.mat` en el que previamente se almacenó el Workspace de una instancia de Matlab.

```
1 # Se importa libreria para leer Workspaces de Matlab y numpy
2 import scipy.io
3 import numpy
4
5 # Se importa archivo con el Workspace
6 mat = scipy.io.loadmat('workspace.mat')
7
8 # Se obtienen los valores de cada variable del Workspace
9 x = mat["x"]
```

```
y = mat["y"]
A = mat["A"]

# Se verifica que la variable 'A' tengan el valor esperado
if numpy.array_equal(A,[[1,2,3],[4,5,6]]):
    print("Respuesta correcta para variable 'A'")
else:
    print("Respuesta incorrecta para variable 'A'")
```

1.3.1. Excel

Excel cuenta con una opción para guardar información de Autorecuperación después de una cantidad determinada de minutos. Con esto, puede configurarse que el trabajo del estudiante se guarde, digamos, cada 1 minuto y de esta forma se podría acceder al documento de cada estudiante con un desfase de a lo más 1 minuto. Considérese la tabla en la figura 1.4. A continuación se muestra un código escrito en Python que verifica que para cada fila de la tabla en la figura 1.4, el valor de la tercera columna corresponde al cuadrado del valor de la primera columna.

	A	B	C	D	E
1	X	Y	X^2	Y^2	X*Y
2	-2	7,8	4	60,84	-15,6
3	-1	5,7	1	32,49	-5,7
4	-0,8	4,9	0,64	24,01	-3,92
5	-0,4	5,1	0,16	26,01	-2,04
6	-0,1	4	0,01	16	-0,4
7	0	2,2	0	4,84	0
8	0,5	3,1	0,25	9,61	1,55
9	0,8	1,7	0,64	2,89	1,36
10	1,2	2	1,44	4	2,4
11	2,1	-2	4,41	4	-4,2
12	3,5	-2,9	12,25	8,41	-10,15

Figura 1.4: Tabla de ejemplo

```
# Se importa openpyxl para leer y escribir archivos Excel
import openpyxl
# Se importa documento Excel
book = openpyxl.load_workbook('Libro1.xlsx', data_only=True)
# Se obtiene primera hoja
sheet = book.active
# Para cada fila, se verifica que la celda de la tercera columna
   corresponda al cuadrado de la celda de la primera columna
for i in [2,3,4,5,6,7,8,9,10,11,12]:
    x = sheet.cell(row=i, column=1).value
    xx = sheet.cell(row=i, column=3).value
    if xx == x*x:
        print("Respuesta correcta")
    else:
        print("Respuesta incorrecta")
```

1.4. Trabajos Anteriores

En previas implementaciones de sistemas para el monitoreo de avance en tiempo real [2] [4] se construyeron lenguajes que permiten al profesor realizar consultas a los documentos activos de cada estudiante a través de una interfaz. En uno de estos trabajos, Álvarez y Antoine [4] construyeron una gramática para realizar tales consultas. La gramática consiste en una lista de elementos terminales y no terminales que permiten realizar múltiples operaciones de consulta, en términos generales, obtener información del documento activo y contrastarla con algún valor (string, double, boolean u otro). Además desarrollaron una interfaz gráfica que le permite al profesor construir consultas concatenando bloques de la librería Blockly de Google que representan elementos de la gramática y de esta manera elaborar consultas tan complejas como se quiera. En la figura 1.5 se muestra un ejemplo de consulta en términos de la gramática construida con la interfaz gráfica.

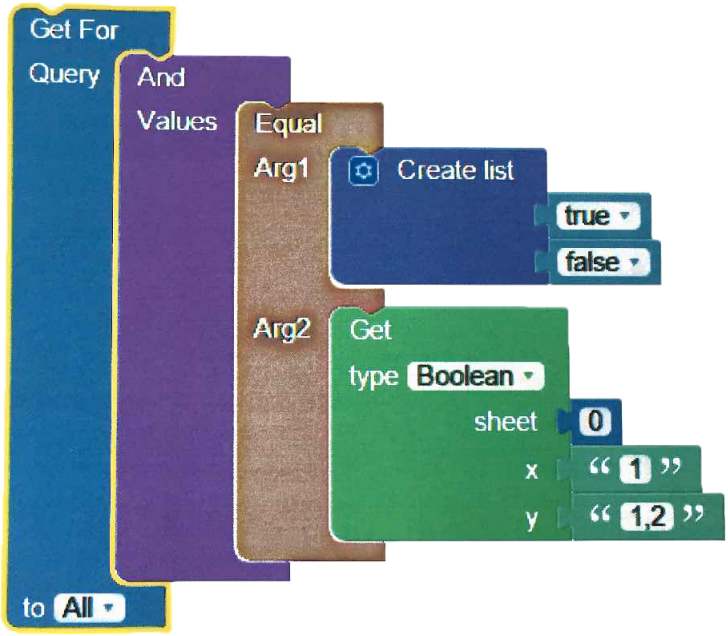

Figura 1.5: Para todos los alumnos, revisar si tienen en las celdas A1 y A2 los valores true y false, respectivamente.

El sistema consiste de dos componentes principales. El primero es el servidor, que se encarga de la resolución de consultas y además proporciona la interfaz web. A través de la interfaz web, el profesor puede crear un nuevo laboratorio y construir las consultas que necesite. Las consultas pueden estar dirigidas a un alumno, todos los alumnos o bien un grupo específico. El servidor se encarga de administrar la conexión con los clientes (estaciones de trabajo) para enviarles las consultas que éstas deben resolver. Toda esta información es almacenada en una base de datos en el servidor.

El segundo componente es el cliente, que es un programa que debe ser ejecutado por los estudiantes para establecer la conexión con el servidor. Cuando se ejecuta el cliente, se abre un formulario que solicita la IP del servidor, el nombre del o los estudiantes que trabajarán en la estación de trabajo y la ruta del archivo en el que trabajarán. Con estos datos, el cliente

envía una solicitud al servidor que como respuesta abre una conexión por socket en un puerto determinado con el cliente. A través de esta conexión por socket las consultas creadas por el profesor son enviadas a las estaciones de trabajo seleccionadas que luego devuelven los resultados.

Los resultados obtenidos de las consultas son almacenados en una base de datos para ser mostrados como se observa en la figura 1.6.

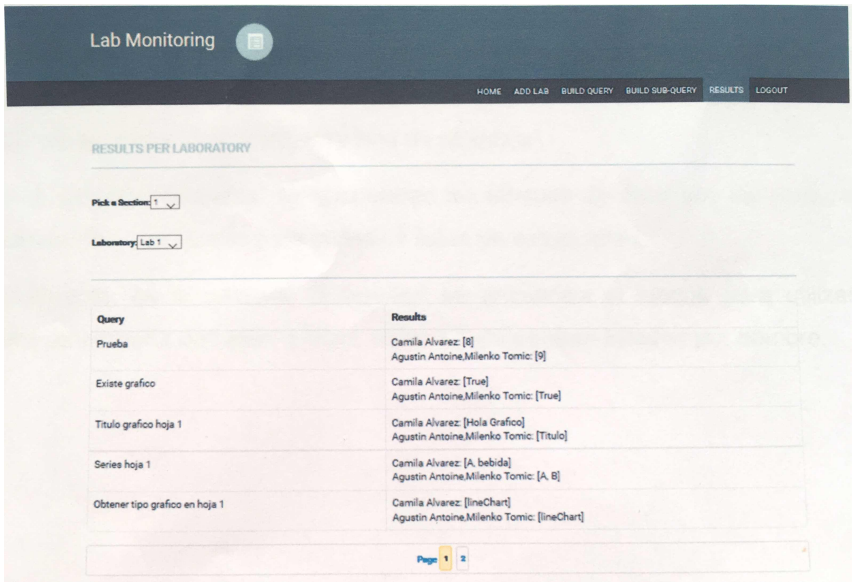

Figura 1.6: Resultados mostrados en la interfaz del profesor.

La gramática desarrollada en el trabajo recién mencionado entrega al profesor una gran flexibilidad para elaborar consultas tan complejas como lo necesite. Sin embargo, esto implica una barrera de entrada para el profesor ya que primero debe estudiar y memorizar los elementos de la gramática y tecnicismos en la interfaz para poder elaborar sus consultas. Al intentar implantar el sistema, este punto resultó ser una dificultad. Se debe determinar hasta qué punto el profesor necesita flexibilidad para realizar consultas ya que una herramienta que entregue un alto poder de consulta hará que la interfaz de usuario sea más compleja y difícil de usar. Por otro lado, uno de los requisitos propuestos es que el sistema debe ser lo menos invasivo en el flujo natural de la sesión de laboratorio posible. El profesor debe elaborar consultas con la interfaz web lo que podría tomarle tiempo valioso de clases. El cliente requiere una interacción por parte del estudiante, que si bien es pequeña, podría ser mejorada. En cuanto a la vista de resultados de la figura 1.6, podrían usarse elementos visuales que faciliten la lectura al profesor tales como gráficos, histogramas con el avance a lo largo de la sesión, etc.

El trabajo propuesto por Hardings [2] funciona de manera similar. Por un lado, una gramática para construir consultas de manera anidada y agentes que se encargan de la comunicación entre terminales del profesor y de los estudiantes. Una consulta puede conectarse con otras consultas en forma bidireccional ya que su entrada y salida son del formato XML. La forma en que se muestran los resultados en este caso se observa en la figura 1.7. Se muestra una tabla con el nivel de avance para cada estudiante sobre cada sección o página de la sesión de laboratorio. Al igual que en el trabajo anterior, se podría mejorar utilizando elementos

visuales.

Student	Page 1	Page 2	Page 3	Page 4	Page 5
pedro	createOnly	get + print	joinOnly	join + mod	read + print
jorge	create + See	get + print	none	none	none
alberto	createOnly	getOnly	get + join	joinOnly	getOnly
juan	createOnly	get + print	joinOnly	join + mod	none

Figura 1.7: Tabla de resultados en la interfaz del profesor.

Capítulo 2

Especificación del problema

Herramientas Computacionales para Ingeniería y Ciencias (CC1000) es un curso dictado en la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile. Este curso busca entregar al estudiante el conocimiento básico del uso de herramientas computacionales que le permitirá resolver problemas que enfrentará a lo largo de su carrera. Entre las competencias que se busca entregar se encuentran:

- Comunicar información utilizando el computador y las redes.
- Resolver problemas estadísticos y gráficos utilizando herramientas de software especializadas.
- Escribir programas en el lenguaje de programación Matlab que resuelvan problemas numéricos del ámbito de las ciencias físicas y matemáticas.
- Resolver analíticamente problemas de álgebra y de cálculo utilizando un software de computación simbólica

La metodología docente usada en este curso se basa en la resolución de problemas. Los estudiantes resuelven problemas tanto en las sesiones presenciales de clases (laboratorios) como en las tareas (proyectos), asistidos por un computador.

2.1. Sesión de Laboratorio

La sesión de laboratorio corresponde a la instancia de clases presenciales en la que el estudiante ejercita el uso de determinada herramienta computacional. En cada sesión de laboratorio se le presenta al estudiante una serie de problemas que debe resolver utilizando alguna de las herramientas que en dicho laboratorio corresponda practicar, ya sea Excel, Matlab o alguna de las otras que se han mencionado. Cada estudiante cuenta con un computador conectado a Internet para la resolución del laboratorio.

2.1.1. Excel

En los laboratorios de Excel, el estudiante abre una nueva instancia de este programa y en este desarrolla todo el laboratorio. Cabe destacar que Excel provee una interfaz de comunicación para que programas externos puedan consultar datos a las instancias de este programa.

2.1.2. Matlab

Similarmente, en los laboratorios de Matlab, el estudiante abre una nueva instancia de dicho programa y en este desarrolla todo el laboratorio. Al igual que Excel, Matlab provee una interfaz de comunicación para que programas externos se conecten, consulten y manipulen sus instancias.

Como se ha mencionado, diversos trabajos han mostrado el efecto positivo de entregar el feedback adecuado al estudiante en el momento oportuno [6] [7] [8]. Pero para saber qué feedback entregar y en qué momento, es necesario monitorear constantemente el progreso de todos los estudiantes. Cuando se trata de clases particulares, esto resulta una tarea sencilla, debido a la interacción directa del profesor con el estudiante. Pero cuando la sesión de laboratorio es en un aula con varios estudiantes, como es el caso del curso CC1000 con aproximadamente 50 estudiantes simultáneos, esta tarea resulta difícil porque no puede tenerse una comunicación directa con cada estudiante ya que tomaría mucho tiempo (en la versión actual del curso, dos estudiantes comparten un mismo computador, por lo que el número de estaciones de trabajo utilizadas en una sesión es de 25 a 30).

Entonces, dado que los estudiantes usan programas de computador para resolver problemas, el avance de las actividades de laboratorio se podría extraer de manera automática de la información de la sesión en la que están trabajando.

Como se ha mencionado, de manera general el problema que se quiere resolver es conocer en tiempo real el progreso de los estudiantes durante las instancias de laboratorio de tipo Excel y Matlab y en función de este conocimiento guiar la sesión de laboratorio.

Capítulo 3

Diseño y Análisis de la Solución

El problema que se quiere resolver es la falta control que tiene el profesor sobre el avance de los estudiantes en las instancias de laboratorio. Para esto, se requiere construir una herramienta que permita al profesor monitorear el avance de los estudiantes en tiempo real en las sesiones de laboratorio.

La solución a implementar debe extraer información de la instancia de Excel o Matlab que en cada computador del laboratorio se está ejecutando y entregarla de manera conveniente al profesor. En el trabajo desarrollado por Álvarez y Antoine [4] se solicitaba al estudiante especificar la dirección del archivo en el que trabajaría durante la sesión. Este paso se puede ahorrar ya que Excel y Matlab proveen mecanismos para interactuar con las instancias que se estén ejecutando en el momento. En la librería de Python llamada `xlwings` provee una función que entrega una lista de las sesiones abiertas a las que se puede tener acceso de manera externa [13]. Similarmente, la API de Matlab para Python brinda acceso a instancias del programa en las que se especificó que la sesión fuera compartida para su libre interacción [13]. Se aprovechará entonces esta capacidad para eliminar el paso de solicitar al estudiante la dirección del archivo en la que trabajará y en vez de esto se determinará de manera automática la sesión abierta.

En implementaciones anteriores, el profesor elaboraba cada consulta durante la sesión de laboratorio y estas eran enviadas a las estaciones de trabajo para su resolución. Esto es poco realista ya que el profesor no cuenta con el tiempo para elaborar consultas durante el laboratorio, se espera en cambio que estas estén definidas previamente y que sus resultados se obtengan de manera periódica y constante sobre todas las estaciones de trabajo a lo largo de la sesión. Por una parte, se deben definir métricas que permitan cuantificar el avance de los estudiantes a partir de sus documentos de trabajo y por otro, se debe proveer un lenguaje que permita al profesor definir o caracterizar la estructura que tiene un documento para que sea considerado correcto y así identificar a aquellos estudiantes que han avanzado correctamente y aquellos que no. Esta estructura, al ser aplicada sobre las sesiones de cada estudiante, deberá entregar una respuesta en términos de las métricas definidas. En adelante, la estructura mencionada que permite determinar el avance sobre un documento de trabajo se llamará pauta.

Por ejemplo, si dado un laboratorio de Excel, el estudiante debe responder `12` en la celda `A10`, sería interesante que el profesor pudiera consultar, mediante la pauta, cuántas de las estaciones de trabajo ya tienen ese valor. Si muy pocas lo tienen después de un tiempo prudente de haber iniciado la sesión del laboratorio, el profesor podría interrumpir el trabajo de los estudiantes y darles alguna sugerencia que destrabe su dificultad.

Además de esto, se debe determinar una manera eficiente en la que fluya la información entre los componentes del sistema (computador del profesor, estaciones de trabajo de los estudiantes, servidor si es que se implementa, etc.) a través de la red disponible en la sala en la que se lleva a cabo el laboratorio sin saturarla.

3.1. Posibles Soluciones al Problema

Dado que el profesor necesita información sobre las actividades de todas las estaciones de trabajo, debe haber un canal de comunicación entre el computador del profesor y aquellas. Una solución podría ser que el sistema haga circular un documento electrónico a través de todas las estaciones de trabajo y estas incorporen la información solicitada. Al terminar de circular, en ese documento estarían contenidas las respuestas de los estudiantes a los problemas que se plantean y de esta forma se podría aplicar un análisis sobre la colección de respuestas para obtener indicadores de avance. Esta solución podría ser factible, pero al no haber simultaneidad, sería ineficiente.

Otra solución sería que cada cierto periodo de tiempo, cada estación enviara el documento en que el estudiante está trabajando al computador del profesor. Esto es factible, pero los archivos podrían llegar a pesar bastante, por ejemplo si contienen tipos de dato multimedia, lo que podría resultar en una congestión de la red. Más aún, el procesamiento de estos archivos para evaluar el avance podría saturar el computador del profesor.

En implementaciones anteriores, el cliente solicita una conexión por socket y el servidor se encarga de establecerla. Mediante esta conexión, se envían las consultas realizadas por el profesor a un grupo de estudiantes.

3.2. Estructura de la solución

Dado un documento de trabajo de algún estudiante, para determinar su avance debe hacerse una comparación respecto de la pauta. Este procesamiento no debiera ser demasiado extenso, pero debe aplicarse sobre cada uno de los documentos en los que están trabajando los participantes del laboratorio. Si se decide realizar este trabajo en un solo computador, podría resultar en un cuello de botella. Esto afectaría la escalabilidad porque cada estudiante adicional incrementaría significativamente el uso de CPU y memoria de un solo computador. Además, interesa evaluar el progreso durante el transcurso del laboratorio, por lo que debe hacerse repetidas veces, por ejemplo cada 30 segundos. Se propone entonces realizar el trabajo de comparación en cada estación de trabajo para distribuir la carga. Esto es más escalable

ya que cada estudiante que se una al laboratorio se encargaría de evaluar su propio progreso y enviarlo asíncronamente, en vez de encargar este trabajo a una sola estación que podría los tiempos de entrega de resultados. Esta última opción se ilustra en la figura 3.1.

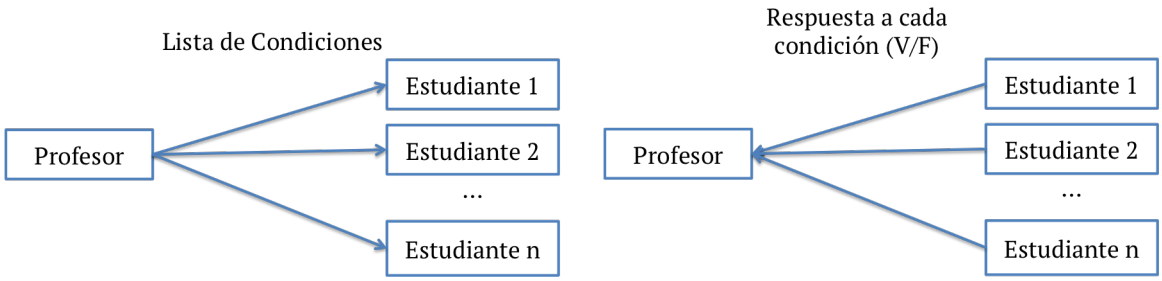

Figura 3.1: Esquema de la solución en la que la evaluación de las condiciones se realiza en el computador de cada estudiante. Se envía la pauta desde el computador del profesor al de cada estudiante y luego se envía la lista de respuestas de cada condición desde el computador de cada estudiante al del profesor.

La comparación con la pauta arrojará como resultado una estructura da datos que contenga sólo los indicadores relevantes para medir el avance. El resultado de este procedimiento aplicado sobre todas las estaciones de trabajo llegará al computador del profesor de manera periódica. Podría ocurrir que todas las estaciones de trabajo enviaran sus resultados al mismo tiempo, lo que provocaría que el computador del profesor se bloquee por un momento mientras se procesan todas las solicitudes. Es por esto que se implementará un intermediario, en adelante servidor, que se encargará de recibir los resultados de la comparación con la pauta de cada estación de trabajo, encolarlos y realizar cualquier procesamiento sobre estos en caso de ser necesario. De esta forma, el computador del profesor sólo deberá consultar al servidor para conocer el avance de todas las estaciones de trabajo, en vez de cada una.

3.3. Lenguaje de Consulta

Como se mencionó, implementaciones anteriores de sistemas de monitoreo de avance ofrecen un lenguaje que permite al profesor elaborar consultas tan complejas como lo necesite sobre los documentos de trabajo de los estudiantes [2] [4]. Sin embargo, esta flexibilidad perjudica la facilidad de uso, ya que se debe adquirir conocimiento técnico sobre el lenguaje para poder elaborar consultas. Además, el lenguaje de consulta visto anteriormente requiere la implementación de un elemento de la gramática por cada funcionalidad que se quiera utilizar en Excel y esto implica mucho trabajo.

Obligar al profesor a dominar un lenguaje nuevo implica serias dificultades para la adopción de este sistema en el laboratorio. En cambio, los lenguajes de Matlab, Excel o el programa que se enseñe ya son dominados por el profesor. Por esto, si las pautas pudieran escribirse como expresiones de Matlab o Excel, la adopción del sistema sería más rápida por parte del profesor.

La pauta puede verse como un conjunto de condiciones que establece el profesor para los

documentos de trabajo. Por ejemplo, si en una pregunta de un laboratorio de Matlab se pide calcular el factorial de 5 y guardarlo en la variable `x`, el profesor establecerá como condición que el valor de x corresponde a 120. Eso lo puede escribir con una expresión booleana de Matlab, por ejemplo `x==5!`, que al ser evaluada sobre el workspace del estudiante, determina que la respuesta a esa pregunta sea correcta.

3.4. Avance como Conjunto de Condiciones

Excel provee un lenguaje para realizar cálculos utilizando como argumento referencias a las celdas de las tablas. Excel provee nativamente una gran variedad de fórmulas, entre ellas operadores aritméticos, lógicos y de comparación, que resultarían útiles para evaluar determinadas condiciones sobre las instancias. Por ejemplo, si se quisiera verificar que el estudiante calculó correctamente el promedio de las celdas de A1 hasta A5 en la celda A6, se podría evaluar la fórmula `=A6=PROMEDIO(A1:A5)` en alguna otra celda. En esta fórmula se usa el comparador "igual que" para verificar que el valor de la celda A6 tenga el valor del promedio de las celdas de A1 a A5. Un ejemplo un poco más complejo sería verificar que los valores entre la fila 1 y 100 de la columna A estén ordenados en forma creciente. Esto se puede hacer usando la fórmula matricial `=Y(A1:A99<=A2:A100)`. El resultado de la fórmula será booleano, es decir, verdadero o falso, correspondiente al cumplimiento o no de la condición requerida.

En el caso de Matlab, se dispone de un lenguaje Turing-completo en el que también se proveen operadores aritméticos, lógicos y de comparación con los que se pueden evaluar condiciones que revelen información del progreso del estudiante. Por ejemplo, para verificar que el valor de la variable `prom` del workspace de Matlab en alguna estación de trabajo corresponde al promedio entre las variables `a`, `b` y `c` previamente definidas por el estudiante, se podría evaluar simplemente el valor de la condición `prom == (a+b+c)/3`. O bien, si una pregunta consistiera en elaborar una función, digamos `cuadrado(x)`, podría evaluarse la expresión `cuadrado(3) == 9` para verificar que esta entregue valores correctos ante distintas entradas.

El trabajo realizado en la sesión del laboratorio tendrá una determinada estructura y, de estar correcto, es decir, de contestar correctamente a los planteamientos de cada pregunta, cumplirá condiciones que pueden ser descritas con los lenguajes anteriormente señalados. De esta manera, para cada pregunta podría elaborarse una lista de condiciones que den cuenta de los requisitos del enunciado de cada pregunta y así, el progreso del laboratorio podría medirse como proporcional a la cantidad de condiciones booleanas que al ser evaluadas resulten verdaderas.

3.5. Inspector

Como se mencionó, se delegará la responsabilidad de evaluar el progreso sobre las sesiones de laboratorio al computador de cada estudiante para distribuir el trabajo de cómputo. Para

esto se construirá un programa, al que se denominará Inspector, que correrá como un proceso aparte. El inspector será responsable de realizar las siguientes tareas:

- Solicitar al servidor la pauta elaborada por el profesor del laboratorio en ejecución.
- Conectarse con la sesión, ya sea de Excel o Matlab, dependiendo del tipo de laboratorio, que se esté ejecutando en el computador del estudiante.
- Realizar la evaluación de las condiciones de la pauta sobre la sesión.
- Enviar el resultado de la evaluación en el servidor.

Este procedimiento debe hacerse de manera periódica, según el intervalo que se haya establecido en la configuración del laboratorio.

El inspector debe funcionar en el sistema operativo Windows y debe ser compatible con las versiones de Matlab y Excel que se use en los computadores de la sala Galileo.

3.6. Estructura de datos

A continuación se definirá la estructura de datos de la solución que se implementará.

- `Lab`: representa la sesión de clases presenciales en la que se resuelven problemas con el objetivo de ejercitar el uso de determinada herramienta (Excel o Matlab). Está compuesto por un conjunto de preguntas, tiene una duración determinada, es de determinado tipo (Excel o Matlab), tiene un progreso asociado que depende del progreso de cada pregunta y tiene asistentes asociados que representan a los estudiantes que participaron en la instancia del laboratorio.
  - `name`: nombre del laboratorio
  - `type`: tipo del laboratorio, puede ser Excel o Matlab.
  - `description`: objetivo general del laboratorio
  - `assistants`: lista de asistentes al laboratorio. Cada asistente corresponde a un computador sobre el que se evaluarán las condiciones.
  - `timestamp`: fecha de creación del laboratorio.
  - `start_timestamp`: fecha en la que se da inicio a la sesión de laboratorio
  - `finish_timestamp`: fecha en la que se da término a la sesión de laboratorio
  - `progress`: número del 0 al 100 que corresponde al porcentaje de progreso neto que haya sido calculado por última vez de manera automática sobre todos los estudiantes.
  - `duration`: duración de la sesión de laboratorio.
  - `interval`: ventana de tiempo en segundos que se espera antes de realizar la próxima evaluación de condiciones sobre cada computador.
  - `status`: estado del laboratorio. Puede ser pendiente, en curso o completado.
- `Question`: Corresponde a cada pregunta que debe resolverse en el laboratorio. Tiene asociada una lista de condiciones que definirán su avance. En el caso de laboratorios de tipo Excel, para facilitar la especificación de las condiciones por parte del profesor, se

proveerá una tabla en la que se podrán ingresar condiciones haciendo referencia a cada celda con la letra `c`, como se muestra en la tabla 3.1.

	A	B	C
1	c = 10		
2		c <10	
3			c = PROMEDIO(A1:C1)

Tabla 3.1: Tabla de condiciones

  - `name`: nombre de la pregunta. En el caso de Excel, a través de este atributo se relacionará cada hoja del documento con una pregunta.
  - `description`: descripción de la pregunta.
  - `guideline_table`: corresponde a la tabla de condiciones, como se ejemplifica en la tabla 3.1.
  - `guideline_list`: además de la tabla de condiciones, podrá establecerse una lista de condiciones que refieran a cualquier celda o rango de ellas. Este atributo representa esa lista de condiciones.
  - `guideline_scripts`: en el caso de laboratorios de Matlab, una de las cosas que se quiere evaluar es la elaboración de scripts, que corresponden a archivos con código de Matlab. Estos archivos se deben ejecutar en ambientes separados y pueden tener inputs asociados. Este atributo almacena una lista de documentos embebidos del tipo `ScriptGuideline`, que corresponde a la pauta sobre un script de Matlab.
  - `lab`: referencia al laboratorio al que pertenece la pregunta.
  - `timestamp`: fecha de creación de la pregunta.
  - `areas`: lista de áreas o materias que se abordan el laboratorio.
  - `nrows`: número de filas que tiene la tabla de condiciones.
  - `ncols`: número de columnas que tiene la tabla de condiciones.
  - `progress`: promedio de los porcentajes de progreso en esta pregunta de cada asistente.
- `ScriptGuideline`: Representa a la pauta de un script de Matlab.
  - `name`: este campo contiene el nombre del archivo de script de Matlab que debe haber creado el estudiante.
  - `inputs`: en los scripts es posible utilizar la función `input` de Matlab, que pausa la ejecución para solicitar al usuario que ingrese datos a través de la entrada estándar del proceso. Por esto, este campo almacena la lista de strings que deben ser ingresados por el usuario que ejecute el script, acompañado del valor que el script debería imprimir en la salida estándar del proceso. En buenas cuentas, este campo almacena una lista de tuplas (valor de entrada, valor de salida).
  - `conditions`: contiene una lista de condiciones (expresiones de Matlab) que serán evaluadas una vez finalizada la ejecución del script en un proceso de Matlab separado.
- `Student`: Representa a un participante del laboratorio, sobre cuyo computador se evaluará la lista de condiciones para determinar el progreso.

  - `name`: en este campo se almacena la dirección MAC del computador usado por el participante.
  - `timestamp`: fecha de creación.
- `Answer`: corresponde al resultado de la evaluación de la lista de condiciones de las preguntas del laboratorio sobre la sesión del computador de un asistente.
  - `timestamp`: corresponde al instante en el que se realizó la evaluación.
  - `answers`: contiene el valor original de las celdas, que en `guideline_table` contengan alguna condición, de cada hoja de cálculo de la sesión de Excel en la que trabaja el estudiante.
  - `results`: representa la correspondencia entre condición y valor de su evaluación en la instancia de Excel en la que trabaja determinado estudiante, para cada elemento de la lista de condiciones.
  - `scripts`: la hoja de respuestas debe permitir almacenar los resultados de las ejecuciones de los scripts. En este campo se deben almacenar esos resultados como una lista o bien como un diccionario cuya llave sea el nombre del script y cuyo valor contenga el resultado de la ejecución.
  - `student`: referencia al estudiante que envía el objeto Answer.
  - `lab`: referencia al laboratorio al que pertenece el Answer.
  - `type`: corresponde al tipo de laboratorio de `lab`.

El modelo de datos se implementará en una base de datos no relacional llamada MongoDB. Se eligió una base de datos no relacional ya que el problema que se aborda requiere flexibilidad en la estructura de los datos. Por ejemplo, como se vio en el caso de laboratorios de Excel, la pauta escrita por el profesor incluye una tabla en cuyas celdas se escriben condiciones para cada una de ellas. El profesor puede querer una tabla bastante grande pero escribir condiciones en unas pocas celdas. Si esto se quisiera almacenar en una base de datos relacional, podría por ejemplo almacenarse en una tabla. Como son pocas las celdas que realmente almacenan condiciones, se estarían guardando muchos datos con el valor nulo, lo que aumentaría innecesariamente el tamaño de la base de datos como se puede ver en la tabla 3.2.

	A	B	C	...
1	c = 10	null	null	null
2	null	c = 10	null	null
3	null	null	c = 10	null
...	null	null	null	c = 10

Tabla 3.2: Tabla de condiciones en base de datos relacional

Otra forma de abordar este problema con base de datos relacional sería crear la estructura que se muestra en la figura 3.2.

De esta manera, una tabla guardaría una referencia a sus filas en la tabla `rows`, y las columnas de cada fila sería guardarían en la tabla `columns`. De esa forma, se guardarían sólo las celdas que tuvieran condiciones y se evitaría guardar valores nulos. El problema de esta estructura es que es compleja de manejar y de extraer a memoria principal. Por otro

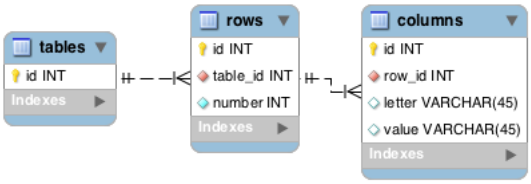

Figura 3.2: Modelo de datos para almacenar tablas.

lado, se podría aprovechar la flexibilidad de bases de datos no relacionales como MongoDB para almacenar esta estructura como documentos. Entonces, una tabla sería simplemente un diccionario cuyas llave es el número de fila y cuyo valor es otro diccionario cuya llave es la letra de la columna y cuyo valor es el valor de la celda, en este caso, una expresión de Excel, como se muestra en la figura 3.3.

```
{
    "1": {
        "A": "c=1",
        "B": "c=2",
        "C": "c=0"
    },
    "2": {
        "A": "c=3",
        "B": "c=4"
    }
}
```

Figura 3.3: Diccionario que representa una tabla.

La característica de MongoDB de manejar las entidades como documentos es la que se consideró para decidir usarla como base de datos, ya que el proyecto requiere esta flexibilidad para almacenar pautas, hojas de respuesta, progreso, etc.

3.7. Métricas

Como se ha mencionado, en cada sesión de laboratorio interesa verificar el cumplimiento de una lista de condiciones en tiempo real, previamente definidas, sobre la sesión de laboratorio de cada estudiante ya que esto permitirá medir la cantidad de avance de cada uno. En adelante, a la cantidad de condiciones que evaluadas sobre una sesión resultan correctas se denominará "progreso". Interesa medir el progreso sobre diversas dimensiones, por ejemplo, el progreso a lo largo de la sesión del laboratorio, progreso según cada materia que se aborda en las preguntas del laboratorio, progreso sobre cada una de las condiciones que fueron enlistadas, progreso sobre cada estudiante, etc.

El progreso según cada una de las dimensiones mencionadas cambia a lo largo del laboratorio ya que este se debe actualizar con cada nuevo conjunto de respuestas de cada estudiante (evaluaciones de la lista de condiciones sobre su sesión), por lo tanto es necesario que se actualicen los valores al menos periódicamente, cada cierto intervalo de tiempo previamente definido. De esta manera el profesor que monitoreará el laboratorio podrá conocer en tiempo real (o desfasado en la ventana de tiempo predefinida), el avance de los estudiantes.

Los lenguajes de consulta creados en implementaciones anteriores de sistemas para el monitoreo de avance entregan al profesor una gran flexibilidad par elaborar consultas. Estos lenguajes permitían dirigir consultas a un grupo de estudiantes, agrupar los resultados según distintas dimensiones, entre otras opciones. En la práctica, se observó que el profesor no necesita tal flexibilidad y que podrían definirse consultas de agrupamiento estándar que abarquen la mayoría de casos que el profesor necesitará. Esto con el objetivo de mejorar la usabilidad y que el profesor no tenga que gastar tiempo de la sesión de laboratorio en diseñar las consultas.

Al cabo de cierto tiempo de una sesión de laboratorio, se tendrán múltiples objetos `Answer` que fueron enviados al servidor por las diferentes estaciones de trabajo que contendrán una fotografía del avance de cada estudiante en distintos momentos. Con estos datos, se podrían realizar consultas a la base de datos para obtener el progreso según distintas dimensiones. Por ejemplo, para obtener la última hoja de respuestas enviada por cada estación de trabajo podría hacerse la consulta de la figura 3.4.

```
db.getCollection('answer').aggregate(
    [{
            $match: {
                'lab': ObjectId("5a5f54a1a85ac505e7bfbf95")
            }
        },
        {
            $group: {
                _id: "$student",
                results: {
                    $last: "$results"
                },
                scripts: {
                    $last: "$scripts"
                },
                timestamp: {
                    $last: "$timestamp"
                }
            }
        }
    ]
)
```

Figura 3.4: Consulta a la base de datos MongoDB para obtener la última hoja de respuestas enviada por cada estación de trabajo.

De esta forma, se podría construir un endpoint de la API que retorne las últimas hojas de respuestas enviadas por los estudiantes. Esta consulta puede resultar simple y rápida, pero casos más complejos podrían tomar más tiempo, como por ejemplo, obtener la última hoja de respuestas enviada por cada estudiante agrupados por intervalos de tiempo, agrupados por pregunta, con el objetivo de construir un histograma que muestre el avance a lo largo de la sesión de laboratorio. A medida que avanza el laboratorio se vuelve más costoso para el motor de base de datos resolver estas consultas porque se van acumulando bastantes hojas de respuesta por cada estudiante. La frecuencia con que el profesor necesita consultar estos datos requiere una solución que tome menos tiempo. Este problema podría resolverse almacenando el progreso de los estudiantes según intervalos de tiempo. Esto es, cada cierta cantidad de tiempo, guardar la suma de respuestas correctas, incorrectas y en blanco de cada pregunta, de manera que extraer el histograma no requiera a cada rato la ejecución de una consulta compleja, sino que solo se requiera extraer un documento directo y ya calculado desde la base de datos. Siguiendo este ejemplo, se definirán documentos que almacenen el cálculo de progreso según las distintas dimensiones que se necesite.

3.7.1. Avance actual de cada estudiante

Se creará una colección de MongoDB que almacene, para cada laboratorio, la contabilidad de condiciones correctas, incorrectas y totales de la última hoja de respuesta de cada estudiante.

3.7.2. Avance por área

Se creará una colección de MongoDB que almacene, para cada laboratorio, el progreso según cada área.

3.7.3. Avance por condición

El avance por condición almacena la el número de estudiantes que han respondido de manera correcta e incorrecta a cada condición de cada pregunta. Se creará una colección de MongoDB que almacene, para cada laboratorio, un diccionario cuya llave sea el identificador de cada pregunta y cuyo valor sea otro diccionario cuya llave sea una condición y cuyo valor sea un diccionario que contenga el total de preguntas, la cantidad de preguntas que fueron respondidas correctamente y las que fueron respondidas incorrectamente para cada condición de cada pregunta.

El siguiente ejemplo muestra un documento que contiene el avance por condición de un laboratorio. Esto es, un diccionario cuyas llaves son el id de cada pregunta, apuntando a otro diccionario con la información de la cantidad de estudiantes que han respondido de manera correcta o incorrecta por cada condición de cada pregunta.


```
{
  "5a5f54c6a85ac505f21211fb": {
    "hipotenusa(3,4)==5": {
      "condition": "hipotenusa(3,4)==5",
      "incorrect": 0,
      "total": 18,
      "correct": 14
    }
  },
  "5a5f55fda85ac505f212123a": {
    "length(arrayfun(@Harmonic, nn))==1000": {
      "condition": "length(arrayfun(@Harmonic, nn))==1000",
      "incorrect": 0,
      "total": 18,
      "correct": 11
    },
    "length(nn) == 1000": {
      "condition": "length(nn) == 1000",
      "incorrect": 1,
```

```
        "total": 18,
        "correct": 13
      },
      "length(x)==6": {
        "condition": "length(x)==6",
        "incorrect": 16,
        "total": 18,
        "correct": 1
      }
    }
}
```

3.7.4. Cronograma

Se creará una colección de MongoDB que almacene la contabilidad del avance cada cierto intervalo de tiempo.

Cada vez que llegue una nueva hoja de respuestas de alguna estación de trabajo, se debe actualizar cada uno de los documentos recién mencionados. Esto debe estar protegido contra concurrencia porque puede que dos estudiantes envíen su hoja de respuesta al mismo tiempo.

3.8. Concurrencia

Como se describió, el problema que se quiere resolver supone la interacción de múltiples estudiantes resolviendo sus respectivos laboratorios al mismo tiempo. La interacción supone el envío de muchas evaluaciones de lista de condiciones. En consecuencia, la solución que se implementará debe soportar múltiples conexiones simultáneas. Estas conexiones son el resultado de múltiples instancias del proceso `Inspector` que reporten sus resultados de manera periódica y asíncrona.

Supongamos que en un instante determinado, dos estudiantes envían su progreso de manera simultánea y los dos tienen correcta una condición adicional a las que tenían previamente en cierta pregunta `P1`. Esto debería gatillar el recálculo de las métricas. Si, por ejemplo, el total de condiciones acertadas de todos los estudiantes para la pregunta `P1` es `x`, el nuevo total correspondería a `x+2`, pero si no se asegura la consistencia de los datos, podría ocurrir una condición de carrera, lo que terminaría por entregar un nuevo total de `x+1`, como se muestra en la figura 3.5. Por esto, el motor de base de datos que se elija deberá garantizar ACID[1] o de lo contrario se deberá utilizar un mecanismo externo para garantizar que no

[1]ACID es el acrónimo para Atomicity, Consistency, Isolation y Durability. En español, Atomicidad, Consistencia, Aislamiento y Durabilidad. Son un conjunto de propiedades necesarias para que un conjunto de instrucciones sean consideradas como una transacción en un sistema de bases de datos. Una transacción es un conjunto de órdenes que se ejecutan formando una unidad de trabajo de forma indivisible o atómica. Un ejemplo de una transacción compleja es la transferencia de fondos de una cuenta a otra, la cual implica múltiples operaciones individuales.

ocurran condiciones de carrera.

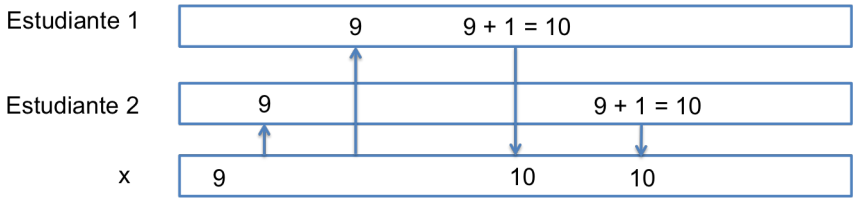

Figura 3.5: Condición de carrera sobre el progreso de cierta pregunta `P1`.

Capítulo 4

Implementación

Se necesita establecer un puente de comunicación bidireccional entre el computador del profesor y el de los estudiantes que participan en la sesión de laboratorio. Para evitar sobrecargar el computador del profesor, se utilizará un servidor externo que se encargue de manejar la comunicación con los computadores de los estudiantes, de manera que el profesor sólo deba realizar una petición al servidor para obtener el progreso de todos los estudiantes y sus respectivas métricas, en vez de una petición por estudiante.

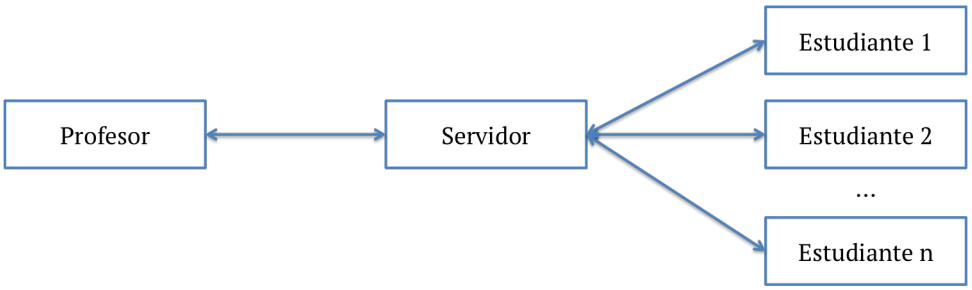

Figura 4.1: Arquitectura servidor - terminales de estudiante - terminales de profesor.

Se construyó una API REST utilizando Django en el servidor que actuará de intermediario entre los computadores de los estudiantes y el del profesor. Tanto el Inspector como la interfaz para el profesor interactuarán con la API.

4.1. Interfaz Web

Para la interacción del profesor con el sistema se construyó una interfaz web con el framework Angular 4. Se eligió Angular, en parte, porque facilita mucho la tarea de actualizar componentes de la página web en el navegador, funcionalidad necesaria para mostrar en tiempo real el avance de los estudiantes, actualizando los valores de progreso y componentes gráficos.

Angular es un framework completo para construir aplicaciones en cliente con HTML y

Javascript, es decir, con el objetivo de que el peso de la lógica y el renderizado lo lleve el navegador, en lugar del servidor.

En términos generales, para crear una aplicación con Angular se componen plantillas HTML (templates) con el markup de Angular. Se escriben componentes para gestionar esas plantillas. Un `Component` controla una zona de espacio de la pantalla que se podría denominar vista. Tomando como ejemplo una aplicación tipo to-do list: La carcasa que engloba toda la aplicación, la barra de navegación, un listado de tareas, cada una de las tareas, o un editor de tareas, etc. son todas vistas controladas por componentes. Se encapsula la lógica de la aplicación en Servicios.

4.1.1. Componente `LoginComponent`

Este componente ofrece un formulario para que el profesor se autentique con el servidor.

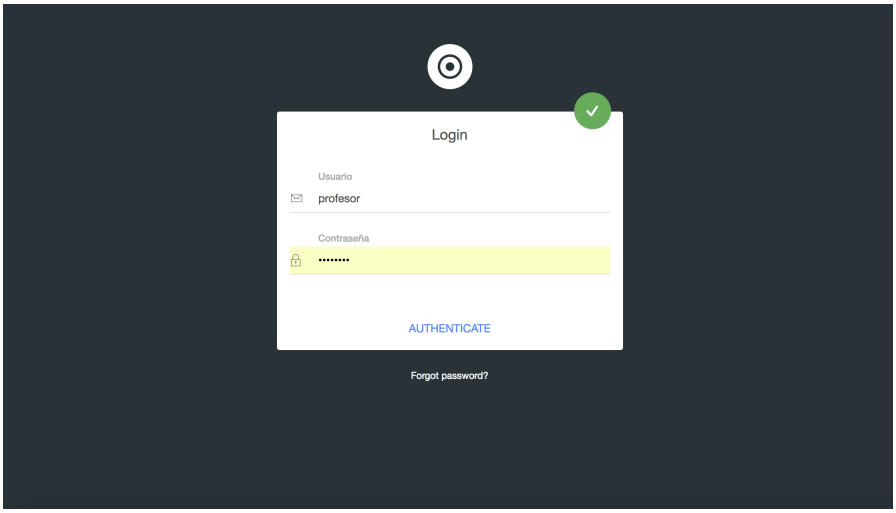

Figura 4.2: Formulario de inicio de sesión

4.1.2. Módulo `Laboratories`

Este módulo encapsula la creación, modificación y consulta de los laboratorios.

Componente `LaboratoriesComponent` Muestra la lista de laboratorios que se han creado. Para cada laboratorio, muestra su estado, descripción y una barra de progreso.

Componente `ShowComponent` Es un panel que muestra las distintas métricas asociadas a un laboratorio, tales como el cronograma, gráfico de avance por pregunta, gráfico de torta para medir avance por área, barras de progreso por cada condición y por cada estudiante.

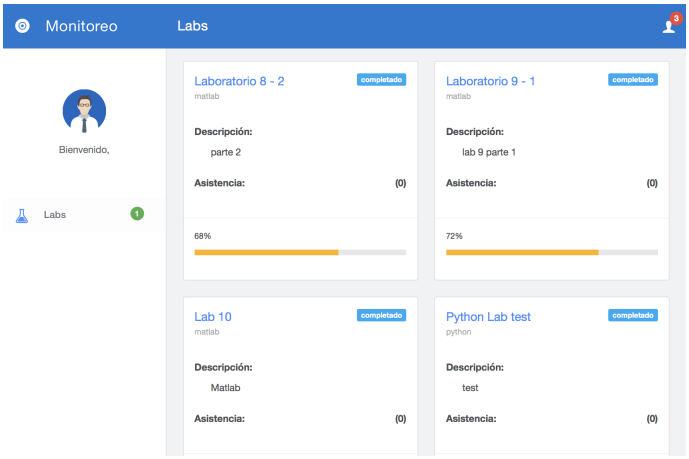

Figura 4.3: Componente `LaboratoriesComponent` del módulo Laboratories

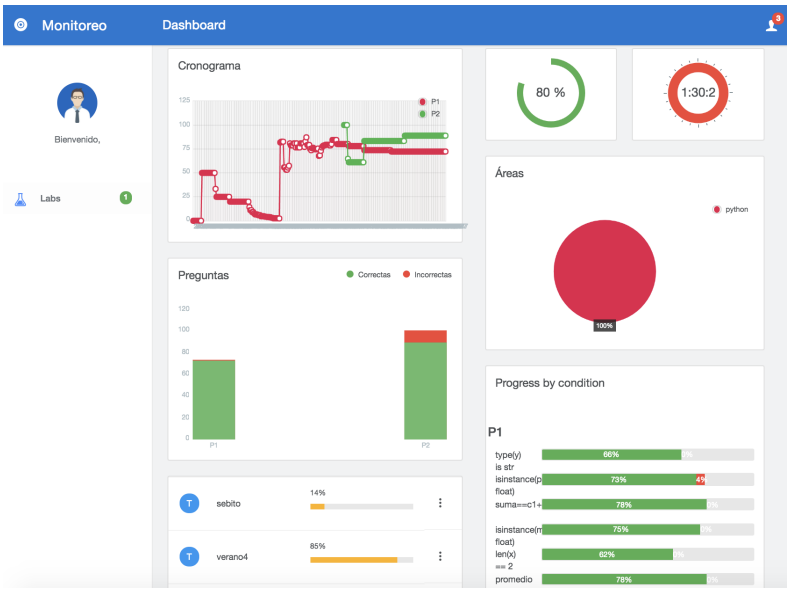

Figura 4.4: Componente `ShowComponent` del módulo Laboratories

Componente `NewComponent` Ofrece un formulario que permite crear un nuevo laboratorio. Este puede ser de tipo Excel o Matlab.

Componente `EditComponent` La misma vista que `NewComponent` pero con los datos prellenados del laboratorio que se quiere editar. Además se muestra la lista de preguntas que se han creado para este laboratorio, con la opción de crear nuevas preguntas. Para cada pregunta, muestra su descripción y una barra de progreso.

4.1.3. Módulo `Question`

Este módulo encapsula la creación y modificación de preguntas de un laboratorio.

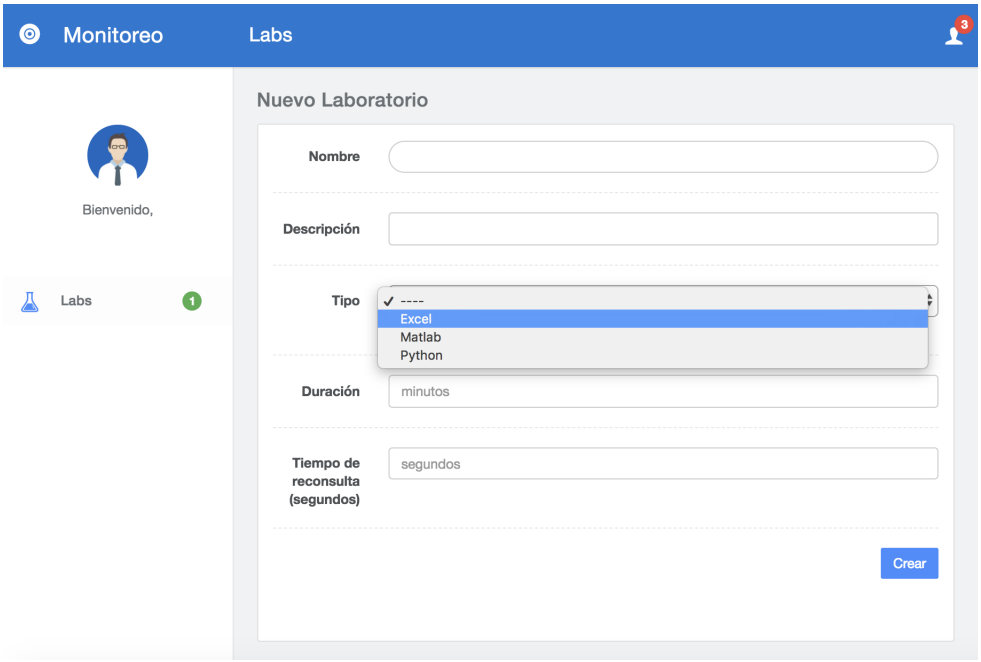

Figura 4.5: Componente `NewComponent` del módulo Laboratories

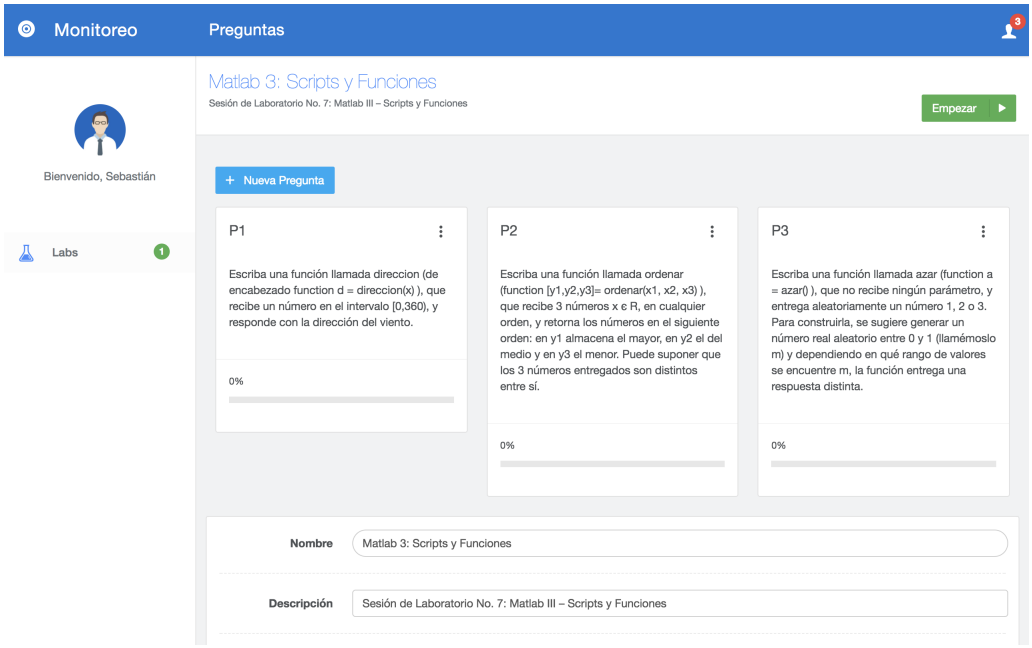

Figura 4.6: Componente `EditComponent` del módulo Laboratories

Componente `NewComponent` Ofrece un formulario para crear una nueva pregunta. Dependiendo del tipo de laboratorio se mostrarán diferentes campos en el formulario que permitan configurar la pregunta según el tipo que corresponda. Para preguntas de laboratorios de tipo Excel, se ofrecen campos para ingresar las dimensiones de la tabla que será utilizada como pauta, una tabla con campos para ingresar las condiciones para cada celda y una lista de campos para ingresar una lista de condiciones generales tan larga como se necesite.

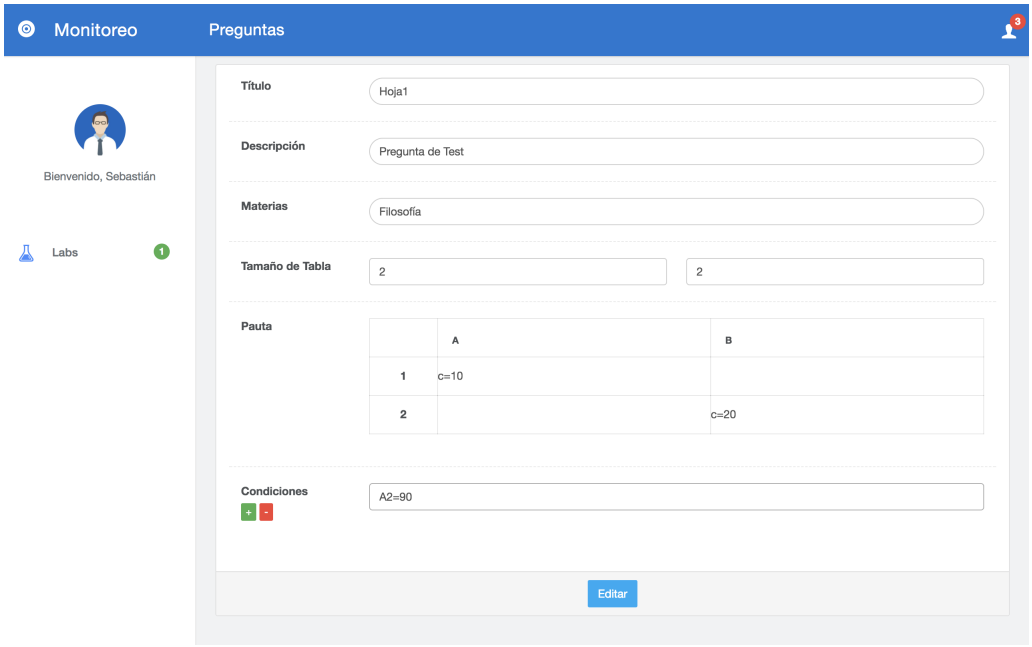

Figura 4.7: Formulario para crear la pauta para preguntas tipo Excel

En el caso de preguntas de laboratorios tipo Matlab, se ofrecen simplemente una lista de campos para introducir la lista de condiciones.

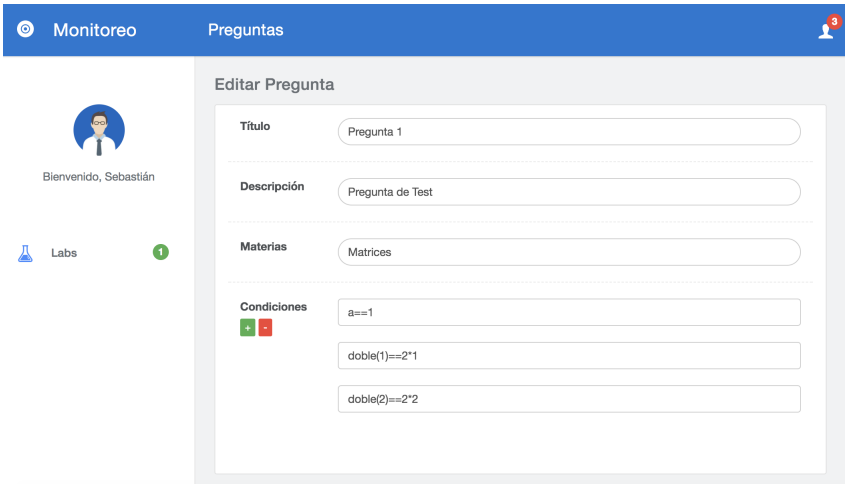

Figura 4.8: Formulario para introducir lista de condiciones para preguntas tipo Matlab

Pero además, una pregunta de Matlab puede requerir la elaboración de scripts. Como se mencionó anteriormente, los scripts son ejecutados en instancias nuevas de Matlab, diferentes a la que utiliza el estudiante. Por esto, se ofrece también un formulario para crear la pauta de estos scripts con un campo para el nombre del script, los inputs que debe recibir y sus outputs asociados y una lista de condiciones que se deben cumplir una vez que el script retorna.

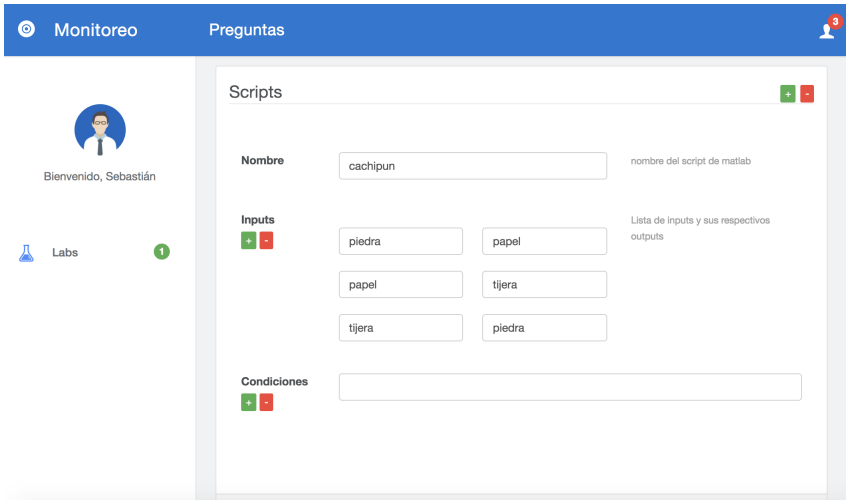

Figura 4.9: Formulario para elaborar pautas de scripts para preguntas tipo Matlab del componente NewComponent del módulo Questions

4.1.4. Servicios

Para realizar las solicitudes a la API se construyó `MonitoreoApiService`. Este servicio ofrece los métodos necesarios para hacer solicitudes de escritura y lectura, esto es, métodos get, post, patch, delete. Estos métodos del servicio se llaman cada vez que se quiera obtener, crear, editar o eliminar laboratorios o preguntas.

El servicio `MetricsService` realiza el preprocesamiento necesario para poblar los gráficos con los datos que se obtienen del servidor. Por ejemplo, colorear cada una de las áreas en el gráfico de torta, llenar las barras de progreso según el porcentaje de avance, etc.

Se provee también `AuthService`, que se encarga de autenticar al usuario con el servidor cada vez que este ingresa sus datos en el formulario de login.

4.2. Servidor

El backend está escrito en Python utilizando el framework Django.

4.2.1. Modelos

El modelo de datos se implementó en una base de datos no relacional llamada MongoDB. Como se mencionó anteriormente, se eligió una base de datos no relacional ya que el problema que se aborda requiere flexibilidad en la estructura de los datos.

4.2.2. Concurrencia

MongoDB no tiene protección contra concurrencia, por lo que se deberá implementar aparte. Para esto, se utilizará Redis Queue (RQ), una librería de Python para encolar trabajos y procesarlos en segundo plano utilizando workers. Utiliza la base de datos en memoria Redis.

Cada vez que se envíe una hoja de respuesta desde una estación de trabajo, es decir, un objeto `Answer`, se encolará la tarea de actualizar cada una de las métricas, de manera que no ocurran problemas de consistencia de datos. En el caso del cálculo del histograma, se cuenta con un "scheduler", que almacena la contabilidad de preguntas correctas, incorrectas y nulas periódicamente.

4.3. Inspector

Como se definió previamente, la responsabilidad de evaluar la lista de condiciones sobre la sesión de trabajo del estudiante recae en su computador. Para esto, debe implementarse un proceso que se ejecute en las estaciones de trabajo de los estudiantes, que se conecte con la sesión, ya sea de Excel o Matlab, haga la evaluación y envíe al servidor los resultados a través de la API en forma de un objeto `Answer`. Al agente encargado de este trabajo se denomina `Inspector`.

El Inspector cuenta con la dirección del servidor y credenciales que le permitirán realizar operaciones de Inspector. Es importante que el servidor prohíba las llamadas a la API de clientes que no se hayan autenticado previamente, para evitar que agentes externos corrompan los datos. En primer lugar, el Inspector se autentica como tal mediante el estándar JWT (JSON Web Token) con el servidor. A partir de entonces, cada petición HTTP que haga el inspector va acompañada de un Token en la cabecera. Este Token no es más que una firma cifrada que permite a la API identificar al usuario. Este Token no se almacena en el servidor, si no en el lado del cliente y la API es la encargada de descifrar ese Token y redirigir el flujo de la aplicación en un sentido u otro. Luego, el inspector ingresa en un ciclo en el que realiza las siguientes tareas:

- Se obtiene el nombre de usuario de la sesión activa del computador en el que se ejecuta el inspector y se verifica que exista un objeto `Student` en el servidor con este nombre de usuario. Si no existe, se crea.
- Mediante una consulta a la API se verifica que haya laboratorios en curso. Si no los hay, se espera un tiempo predefinido y se reinicia el ciclo.
- Si hay un laboratorio en curso, se verifica que exista una sesión activa de Excel o Matlab, dependiendo de cuál sea el tipo del laboratorio.
- Si existe tal sesión, se delega a `MatlabInspector` o `ExcelInspector` la tarea de evaluar la sesión utilizando la pauta.

`BaseInspector` es una clase abstracta que representa al agente que, dado una sesión de Excel o Matlab, evalúa la pauta sobre dicha sesión. Las clases que extiendan a `BaseInspector` deben implementar el método `inspect(session)`, que recibe un objeto sesión, ya sea de Excel

o Matlab, y retorna un objeto Answer con el resultado de la evaluación de la pauta sobre la sesión.

4.3.1. MatlabInspector

`MatlabInspector` extiende a la clase `BaseInspector`. Implementa el método `inspect`, que recibe como parámetro una sesión de Matlab. Este objeto es una instancia de la clase `MatlabEngine`, provista por la API para Python de Matlab.

API de Matlab para Python La API Matlab Engine para Python provee un paquete de Python para realizar llamadas a Matlab como un motor computacional. Dependiendo de la versión de Matlab con la que se cuente, la API puede soportar las versiones de Python 2.7, 3.4, 3.5 y 3.6. Dado que en los computadores de la sala Galileo se utiliza Matlab R2016b, será esta la versión soportada por Inspector.

Entre las funciones que provee la API para Python de Matlab R2016B se encuentran:

- Matlab.engine.start_Matlab: Empezar una nueva sesión de Matlab.
- Matlab.engine.find_Matlab: Enlistar el identificador de las sesiones de Matlab de tipo compartidas que se estén ejecutando actualmente.
- Matlab.engine.connect_Matlab: Conectarse a una sesión de Matlab de tipo compartida, a través de su identificador.

Una sesión de Matlab puede o no ser de tipo compartida, dependiendo de la configuración que haya establecido el usuario. Para compartir una sesión actualmente en curso, Matlab dispone las siguientes funciones, que pueden ser ejecutadas desde la sesión de Matlab que se esté usando:

- Matlab.engine.shareEngine: Convierte una sesión de Matlab de normal a compartida.
- Matlab.engine.engineName: Retorna el identificador e la sesión actual de Matlab.
- Matlab.engine.isEngineShared: Determina si la sesión de Matlab actual es o no de tipo compartida.

La función `inspect` recorre la lista de condiciones de la pauta y las evalúa en la sesión de Matlab. El objeto de sesión de Matlab provee la función `eval`, que recibe como argumento una expresión de Matlab y retorna el resultado de evaluar dicha expresión sobre la sesión, el cual será booleano. Una vez evaluada la condición, se anota el resultado de esta en el objeto `Answer`.

Además de evaluar la lista de condiciones se deben evaluar los scripts según la pauta. Un script puede modificar el workspace de la sesión en la que sea ejecutado, por lo que es importante crear un nuevo proceso de Matlab, distinto del que está en ese momento utilizando el estudiante. Para evaluar un script, primero se le busca en la carpeta de trabajo de Matlab del estudiante a través de su nombre. Si es que existe, se procede a abrir un nuevo proceso de Matlab invisible para el estudiante y se ejecuta el script. El script puede requerir, en caso

de que la pauta así lo indique, valores de entrada. Una vez ejecutado el script, se ingresan los valores de input a través de la entrada estándar del proceso. Justo después de esto, se verifica que el valor arrojado en la salida estándar del proceso sea el que estaba asociado en la pauta a dicho input. Luego de que se recorrieran todos los inputs, se procede a evaluar las condiciones sobre la sesión de Matlab del nuevo proceso, de igual manera que se hizo en la sesión abierta del estudiante. Una vez finalizado el procedimiento, se incorporan los resultados al objeto `Answer`, y se retorna.

4.3.2. ExcelInspector

ExcelInspector extiende a la clase `BaseInspector`. Implementa el método `inspect`, que recibe como parámetro una referencia a la sesión de Excel que se encuentre en ejecución. Esta referencia permite la comunicación con la sesión, entregando total libertad para consultar y modificar el documento asociado a la sesión de Excel. Lo que se requiere es evaluar las condiciones de la pauta sobre el documento de Excel. Esto puede hacerse escribiendo la condición en alguna celda del documento y posteriormente consultando su valor final. Cada condición es una fórmula de Excel que al ser evaluada entregará un resultado. Por ejemplo, si se escribe la fórmula `=1+1` en alguna celda del documento de Excel, al consultar el valor de esta celda debería arrojar como resultado 2. Otro ejemplo, al escribir la fórmula `=A10=PROMEDIO(A1:A9)` en determinada celda, el valor de esa celda al ser consultado respondería con un valor booleano a la pregunta de si la celda `A10` contiene el promedio de las celdas entre `A1` hasta `A9`.

Como se mencionó, cada pregunta del laboratorio tiene asociada una lista de condiciones que al ser evaluadas determinarán el avance. Cada pregunta hace referencia a una hoja específica del documento, por lo que las condiciones asociadas deben ser evaluada exclusivamente en dicha hoja.

Para realizar esta tarea, se podría abrir una nueva sesión de Excel con el documento del estudiante, escribir cada condición en celdas vacías y evaluar el resultado. El problema de esto es que una sesión de Excel toma bastante tiempo en iniciarse, lo que haría demorar el envío de respuestas al servidor. Por otro lado, se podría aprovechar la sesión que ya tiene abierta el estudiante para realizar la evaluación.

El método `inspect` de `ExcelInspector` entonces obtiene una referencia a la sesión actualmente abierta de Excel. Luego escribe las condiciones en la última fila de cada hoja del documento y obtiene el resultado de su evaluación. Esto se hace para evitar abrir una nueva instancia de Excel. Una vez evaluadas las condiciones, los resultados son añadidos al objeto `Answer` y lo retorna.

Capítulo 5

Pruebas

Inicialmente se pretendía probar la solución en la instancia del curso CC1000 de plan común, 2º semestre de 2017. Sin embargo, esto no fue posible ya que al momento de realizarse los laboratorios, no se tenía contemplada una versión disponible para pruebas. Por esto, se probó el sistema en la realización del curso Herramientas Computacionales de la escuela de verano 2018. El curso es dictado por Patricio Poblete, académico del Departamento de Ciencias de la Computación en conjunto con Maira Marques, alumna de doctorado del Departamento de Ciencias de la Computación. Se realizó desde el 8 hasta el 26 de enero de 2018 en el tercer piso del Departamento de Ciencias de la Computación.

Figura 5.1: Curso Herramientas Computacionales semestre de verano 2018.

En el curso participaron aproximadamente 30 estudiantes de enseñanza media y los contenidos que se abordaron son similares a la versión dictada en Plan Común de Ingeniería. El objetivo general del curso es acercar a los alumnos al mundo de la computación, mostrar todo el potencial y utilidad que distintas herramientas computacionales pueden a tener en el día a día. Sus objetivos específicos son enseñar a los estudiantes a crear, editar y manejar un documento Word, manejar una planilla Excel, crear su propio blog, escribir programas sencillos que resuelvan problemas numéricos del ámbito de las ciencias físicas y matemáticas en los lenguajes Python y Matlab. Todas las sesiones de clases tienen un enfoque teórico/práctico.

Los estudiantes resuelven problemas tanto en las sesiones presenciales de clases como en las tareas (proyectos). Las clases tendrán la siguiente estructura general:

- Profesor presenta contenidos/herramientas.
- Estudiantes hacen una actividad en clases utilizando un computador con la tutoría de los profesores y de un profesor auxiliar.

Se realizaron dos pruebas, una en un laboratorio de Python y la otra en uno de Matlab. Se desplegó el sistema de monitoreo en una máquina virtual en Digital Ocean[1]. Con esto, el profesor podía acceder a través de un link a la interfaz de monitoreo. Se contaba además con un link para que los alumnos descargaran el programa `Inspector` específico para Python y Matlab. Al ser ejecutado, `inspector` abre una nueva instancia de Python o Matlab en la que los estudiantes deben trabajar exclusivamente y sobre la que se realizará el monitoreo. Los pasos para realizar la prueba se enlistan a continuación:

- Se crea un nuevo laboratorio en el sistema de monitoreo.
- Se crea una versión de Inspector compatible con Matlab R2015a que se encarga de abrir la sesión de Matlab que los estudiantes utilizarán para realizar su trabajo, además de realizar el monitoreo.
- Se anota en la pizarra una URL a la que los estudiantes deben acceder para descargar el inspector.
- Una vez que los estudiantes descargan y abren el ejecutable, se inicia el Inspector y con ello la sesión de Matlab. En este momento el inspector empieza a enviar objetos de tipo `Answer` al servidor.
- A medida que avanza la sesión del laboratorio, un asistente crea condiciones a través de la interfaz web que permitan mostrar que los estudiantes siguen las instrucciones que da el profesor.
- El profesor observa los distintos indicadores de avance en la pantalla y esto le permite determinar y entregar verbalmente el feedback oportuno para los estudiantes.

Al momento de realizar la prueba, el Inspector tuvo problemas para ejecutarse ya que la versión de Matlab que estaba instalada en los computadores era la R2015a y el sistema se implementó para funcionar con Matlab R2016b. El problema es que la API de Matlab para Python cambia bastante entre la versión R2015a y R2016b. La versión R2015a no cuenta con la opción para conectarse a una sesión existente de Matlab y sólo puede tener acceso a instancias que hayan sido creadas utilizando la API. Por esto, se tuvo que crear un ejecutable que crea un proceso de Matlab al que el inspector tuviera acceso.

En los contenidos del curso se incluye Python, por lo que se implementó un inspector para monitorear el avance sobre sesiones de este lenguaje. En este caso, `Inspector` para Python abre una instancia del entorno de desarrollo de Python en la que los estudiantes desarrollarán sus actividades y en ella ejecuta el proceso de monitoreo. Para esto, se implementó `PythonInspector`, que extiende a `Inspector`. De esta manera, la lista de condiciones es evaluada sobre el ambiente en el que los estudiantes desarrollan su trabajo, teniendo así

[1]Digital Ocean es un proveedor estadounidense de servidores virtuales privados, basado en la ciudad de Nueva York.

acceso a todas las variables y funciones que hayan sido definidas por ellos. En el laboratorio de Matlab, entre las instrucciones que recibieron los estudiantes se encuentran:

- Crear una función llamada "hipotenusa(a,b)" que recibe como argumento dos catetos de un triángulo rectángulo y retorna la hipotenusa.
- Crear una función llamada "demora(n)" que entregue el número de pasos que demora el granizo de La Conjetura de Collatz en caer desde una altura n.
- Crear la función "Harmonic(n)" que calcula la suma de los elementos de la serie armónica hasta n.

Para evaluar esto, se escribieron las condiciones que se muestran en la figura 5.2.

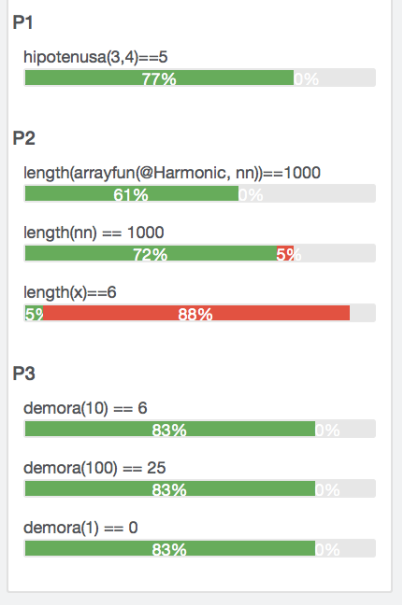

Figura 5.2: Avance por condición.

En la figura 5.3 se muestra el progreso general de cada estudiante. Se enlistan los nombres de usuario de cada estación de trabajo y su progreso. El profesor cuenta con una lista que asocia los nombres de cada estudiante con el usuario que utiliza, lo que le permite identificar a aquellos que estén teniendo dificultades.

Similarmente, durante el laboratorio de Python el profesor solicitaba a los estudiantes realizar cálculos y asignarlos a determinadas variables. Con esto, se elaboraron las condiciones que se muestran en la figura 5.4.

Luego de proponer una tarea, el profesor tenía acceso al panel que se muestra en la figura 5.5. En este gráfico se observa que la pregunta 1 y 2 tienen comportamiento decreciente. Esto puede explicarse por varios motivos. Primero, el avance de cada pregunta es el promedio del avance de cada estación de trabajo. Si en determinado momento sólo una estación ha enviado su hoja de respuestas y este tiene todas las respuestas correctas, el avance en esta pregunta será 100 %. A medida que se vayan sumando hojas de respuesta, se obtendrán números que representen mejor la realidad. En segundo lugar, a medida que los estudiantes desarrollan las otras preguntas, puede que reasignen variables que eran usadas para evaluar

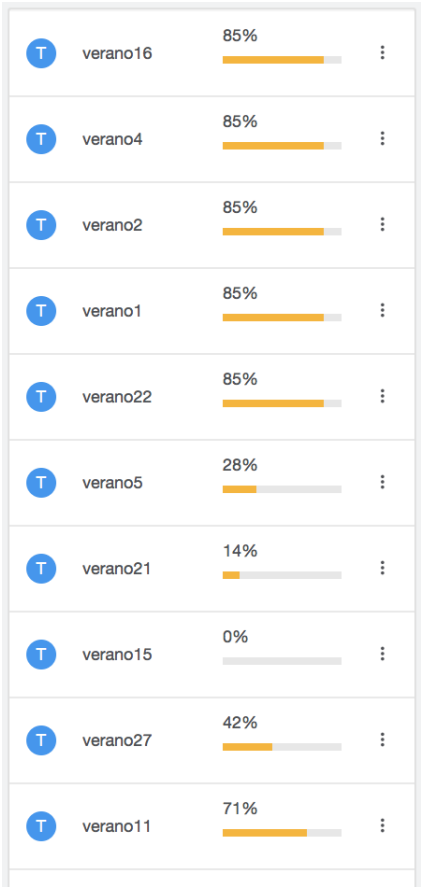

Figura 5.3: Avance por estudiante.

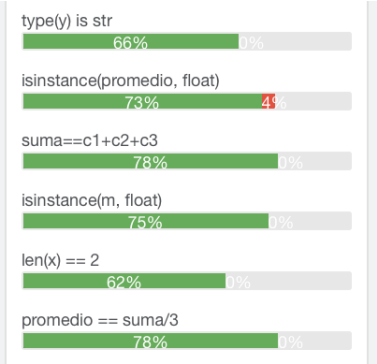

Figura 5.4: Avance por pregunta.

el progreso de otras preguntas, por lo que se mostrará que algunas preguntas tienen algún grado de retroceso.

Las condiciones fueron ingresadas a medida que se desarrollaba el laboratorio. Por esto, el avance de la pregunta 3 aparece desde la segunda mitad del laboratorio. Cuando este sistema de monitoreo sea adoptado, es preciso que las condiciones se ingresen antes de iniciar el laboratorio.

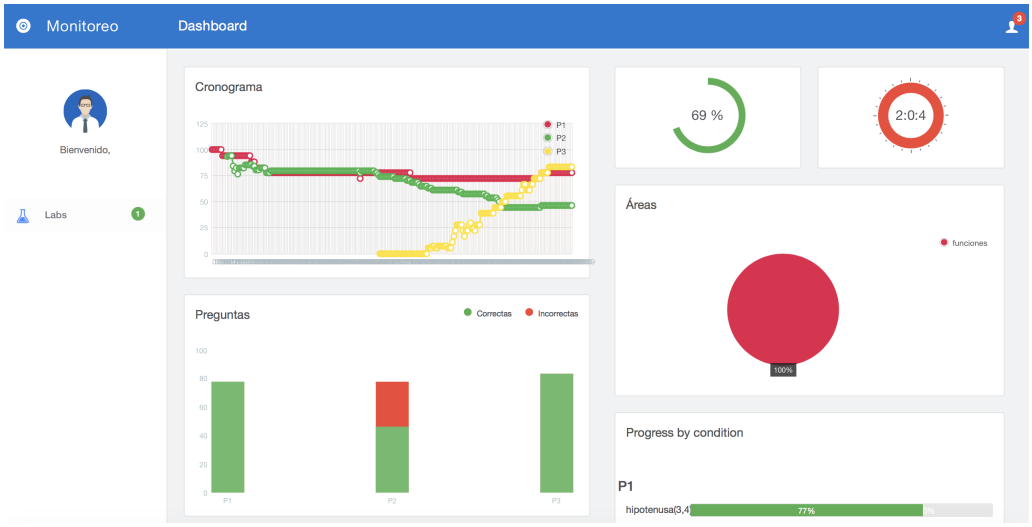

Figura 5.5: Panel de monitoreo

Conclusión

El principal objetivo de este trabajo consistió en crear una herramienta que permitiera al profesor tener monitorear el avance de los estudiantes en las tareas asignadas durante un laboratorio de Excel y Matlab. Esta información permitiría al profesor entregar feedback adecuado y a tiempo. Esto se logró creando un sistema que permite al profesor crear una lista de condiciones que se deben cumplir en las sesiones de los estudiantes, lo que permite determinar en qué medida han avanzado.

Se logró crear un sistema que requiere una mínima intervención al flujo natural del laboratorio, comparado con otras implementaciones. Los estudiantes solo deben descargar un archivo y ejecutarlo. El profesor por su parte puede elaborar una lista de condiciones de antemano en términos de lenguajes que ya domina, de forma que durante el laboratorio sólo debe ingresar al sistema para ver los resultados de los estudiantes.

Para realizar las pruebas al sistema, se escribieron condiciones durante la sesión del laboratorio. Para no quitar tiempo de clases al profesor, es ideal que la pauta sea escrita antes de iniciar el laboratorio, de manera que el profesor sólo tenga que mirar los resultados del monitoreo. En las pruebas se comprobó que es necesario familiarizar previamente al equipo docente con el sistema. La breve introducción realizada momentos antes de las pruebas no fue suficiente para el uso autónomo sin asistencia por parte de los profesores. Las pruebas realizadas mostraron que el sistema es capaz de funcionar en una sesión de 30 computadores.

El sistema es extensible a otros lenguajes de programación. Para soportar otro lenguaje o aplicación, se debe extender la clase `Inspector` e implementar el método `inspect` que debe realizar la evaluación de las condiciones recibiendo como argumento un puntero a la sesión del programa usado por cada estudiante para realizar su trabajo. Esto se hizo de manera rápida con Python, implementando un Inspector que inicia una instancia del ambiente de desarrollo y lo consulta.

El sistema podría usarse para monitorear sesiones de laboratorio no presenciales gracias a que funciona con Internet en vez de alguna red local. El sistema soportó la sesión con 30 estudiantes enviando sus respuestas periódicamente, pero es necesario realizar pruebas para determinar el número máximo de estudiantes que el sistema soporta.

Actualmente, el Inspector consulta cada cierto tiempo las sesiones de trabajo de los estudiantes para enviar los resultados al servidor. Podría implementarse que envíe las respuestas sólo si el estudiante ha realizado cambios. La interfaz del profesor también realiza consultas periódicas al servidor para obtener el progreso. Podría implementarse WebSockets para ob-

tener el progreso inmediatamente después que el estudiante publica su hoja de respuestas. Esto disminuiría el desfase temporal de la información que recibe el profesor con respecto de la realidad.

Bibliografía

[1] N. Baloian, J. A. Pino, J. Hardings, H. U. Hoppe. *Monitoring Student Activities with a Querying System over Electronic Worksheets*. In Proc. 20th Collaboration Researchers' International Workshop on Groupware (CRIWG), pp. 38-52, Sep 2014. Santiago, Chile. Springer-Verlag, Germany. Lecture Notes in Computer Science vol. 8658.

[2] Jens Hardings Perl. *Basis for a Methodology to Define, Validate and Apply Best Practices in a Computer-integrated Classroom*. Marzo 2006. PhD Thesis. Universidad de Chile. Santiago, Chile.

[3] Sergio Peñafiel. *Propuesta de Trabajo Dirigido: Monitoreo de Alumnos*. 7 de Septiembre de 2016.

[4] Camila Álvarez y Agustín Antoine. *Trabajo Dirigido: Sistema de Monitoreo en Laboratorio*. 2015. DCC, Universidad de Chile.

[5] *Programa del Curso: CC1000 Herramientas Computacionales para Ingeniería y Ciencias*. https://ucampus.uchile.cl/m/fcfm_programas_cursos/bajar?id=5995. UCampus.

[6] Bangert-Drowns, R., Kulik, C., Kulik, J., Morgan, M. *The Instructional Effect of Feedback in Test-Like Events*. Review of Educational Research 61(2), 213–238 (1991)

[7] Hattie, J., Timperley, H. *The power of feedback*. Review of Educational Research 77, 81–112 (2007).

[8] Ciravegna, F., Dingli, A., D. Petrelli, and Y. Wilks, (2002) "Timely and non-intrusive active document annotation via adaptive information extraction," Workshop Semantic Authoring Annotation and Knowledge Management, 2002.

[9] Hasebrook, J.P. and Maurer, H.A. (2004) Learning Support Systems for Organizational Learning, World Scientific Publishing Company

[10] Pinkwart, N. (2005) "Collaborative Modeling in Graph Based Environments", Ph.D. thesis, Universität Duisburg-Essen, Germany

[11] Verdejo, M.F., Barros, B. , Read, T. and Rodriguez-Artacho, M. (2002) "A system for the specification and development of an environment for distributed cscl scenarios,". LectureNotes in Computer Science 2363, pp. 139–148, 2002.

[12] https://www.mathworks.com/Matlabcentral/fileexchange/4688-autosave

[13] Librería de Python *xlwings*. http://docs.xlwings.org/en/stable/index.html

[14] *Matlab API for Python*. https://la.mathworks.com/help/matlab/matlab-engine-for-python.html

Apéndice A

Apéndice

A.1. autosave.m

```
function tout = autosave(varargin)
% AUTOSAVE automatically saves data in the base workspace
% AUTOSAVE saves all of the variables in the base workspace every 10
% minutes, to the matlab.mat file.
% AUTOSAVE(PERIOD) will save data every PERIOD minutes to the matlab.
   mat file.
% AUTOSAVE(PERIOD,SAVEFILE) will save variables to the file whose
   name is
%  specified by SAVEFILE.
% AUTOSAVE(PERIOD,SAVEFILE,SAVEARGS) will pass the options in the
   cell
%  array as inputs to the SAVE function.
% AUTOSAVE STOP will disable autosaving.
% AUTOSAVE START will restart autosaving.
% AUTOSAVE DELETE will disable autosaving, delete the autosave file,
   and
%  remove the timer used to perform the saving.
%
% Examples:
%
%    % autosave with defaults
%
%    autosave
%
%
%    % autosave every 3 minutes
%
%    autosave(3)
%
%
```

```
%     % autosave every 7 minutes, to data.mat
%
%     autosave(7,'data.mat')
%
%
%     % save variables whose names start with 'x' into 'test.mat'
%     % every 5 minutes.
%
%     autosave(5,'test.mat','x*')
%

% defaults
period = 10;
filename = 'matlab.mat';
saveargs = {};

% check inputs
error(nargchk(0,3,nargin));

% handle STOP/START/DELETE
if nargin == 1 && ischar(varargin{1})
    switch  lower(varargin{1})
        case 'start'
            t = getappdata(0,'AutoSaveTimerHandle');
            if isempty(t)
                t = timerfind('Tag','AutoSaveTimer');
            end
            if isempty(t) || ~isvalid(t)
                t = autosave;
            end
            if strcmp(get(t,'Running'),'off')
                start(t);
            end
        case 'stop'
            t = getappdata(0,'AutoSaveTimerHandle');
            if isempty(t)
                t = timerfind('Tag','AutoSaveTimer');
            end
            if ~isempty(t) && isvalid(t)
                stop(t);
            end
        case 'delete'
            t = autosave('stop');
            if isvalid(t)
                delete(t);
            end
            filename = getappdata(0,'AutoSaveFile');
            if ~isempty(filename) && exist(which(filename))
                delete(filename)
            end
```

```
77      end
78
79      % return early
80      if nargout
81          tout = t;
82      end
83      return;
84  end
85
86
87  % assign inputs
88  switch nargin
89      case 1
90          period = varargin{1};
91      case 2
92          period = varargin{1};
93          filename = varargin{2};
94      case 3
95          period = varargin{1};
96          filename = varargin{2};
97          saveargs = varargin{3};
98  end
99
100 % check inputs.
101 msg = checkPeriod(period); error(msg);
102 msg = checkFilename(filename); error(msg);
103 msg = checkSaveArgs(saveargs); error(msg);
104
105 % convert to seconds
106 periodSeconds = period * 60;
107
108 t = getappdata(0,'AutoSaveTimerHandle');
109
110 if isempty(t) || ~isvalid(t)
111     % create timer object
112     t = timer('TimerFcn',{@savedata,filename,saveargs},...
113         'Period',periodSeconds,...
114         'ExecutionMode','fixedRate',...
115         'Tag','AutoSaveTimer');
116     % save timer handle
117     setappdata(0,'AutoSaveTimerHandle',t);
118     setappdata(0,'AutoSaveFile',filename);
119 end
120
121
122 % start the timer
123 if strcmp(get(t,'Running'),'off')
124     start(t);
125 end
126
```

```
127 if nargout
128     tout = t;
129 end
130
131 %%%%%%%%%% save data  %%%%%%%%%%
132
133 function savedata(h,ev,filename,saveargs)
134 %SAVEDATA saves data from the base workspace
135
136 % convert cell array to a single string
137 if iscellstr(saveargs)
138     tmpsaveargs = '';
139     for n = 1:length(saveargs)
140         tmpsaveargs = [tmpsaveargs ' ' saveargs{n}];
141     end
142     saveargs = tmpsaveargs;
143 end
144
145 % create string to execute that will save the data
146 savestr = ['save ' filename ' ' saveargs];
147 evalin('base',savestr);
148
149
150
151 %%%%%%%%%% check inputs  %%%%%%%%%%
152
153 function msg = checkPeriod(period)
154 msg = '';
155 if ~isnumeric(period) || numel(period) ~= 1
156     msg = 'First input must be a numeric scalar.';
157 end
158
159 function msg = checkFilename(filename)
160 msg = '';
161 if ~ischar(filename) || size(filename,1) ~= 1
162     msg = 'Second input must be a string array.';
163 end
164
165 function msg = checkSaveArgs(saveargs)
166 msg = '';
167 if isempty(saveargs)
168     return
169 end
170 if ~ischar(saveargs) && ~iscellstr(saveargs)
171     msg = 'Third input must be a string array or cell array of
        strings.';
172 end
```