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

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA PARA RECOPILAR APUNTES DE MANERA ASÍNCRONA EN U-CURSOS

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN

MARTÍN ESTEBAN ARAYA ZAVALA

PROFESORA GUÍA:
JOCELYN SIMMONDS WAGEMANN

PROFESOR CO-GUÍA:
WILLY MAIKOWSKI CORREA

MIEMBROS DE LA COMISIÓN:
ÉRIC TANTER
DANIEL PEROVICH GEROSA

SANTIAGO DE CHILE
2023

Resumen

Una de las actividades más típicas de un estudiante universitario durante su paso por una carrera corresponde a la toma de apuntes. Típicamente, esta toma de apuntes es realizada mediante el uso de papel y lápiz, aunque últimamente una cantidad importante de alumnos opta por una plataforma digital. Independientemente de la plataforma utilizada, un grupo de alumnos se destaca por re-escribir los apuntes tomados en clases realizando resúmenes, los cuales son compartidos entre diferentes grupos de amigos o de clase.

Esta memoria presenta una herramienta para aquellos alumnos que deseen realizar este proceso de compartir, entregando una plataforma que acelere dicho proceso, especialmente para aquellos que requerirían pasar de un medio físico a uno digital, mediante el uso de sistemas de reconocimiento óptico de caracteres u OCR por su acrónimo en inglés.

Teniendo esta oportunidad de mejora en mente, se realizó una encuesta a estudiantes y una comparativa de alternativas OCR existentes, con el fin de obtener información preliminar sobre ambos temas. Una vez terminados, se realizó el análisis de la plataforma escogida —la plataforma Ucampus— y los casos de uso necesarios para el funcionamiento de la plataforma.

Posteriormente, se realizó el diseño de las interfaces necesarias y el desarrollo de la implementación de las funcionalidades elegidas, ajustándose a la arquitectura de la plataforma Ucampus, definiendo la organización de la información en un sistema de documentos y cuadros.

Durante la implementación del módulo, se realizaron dos validaciones con el equipo de desarrollo del Centro Tecnológico Ucampus, la primera durante la implementación y la segunda al final de esta. Adicionalmente, se realizó una validación adicional con 5 estudiantes de diferentes universidades, la que consistió en la realización de una serie de acciones, la aplicación de la escala SUS, y una conversación abierta con ellos.

Finalmente, se concluye que la creación de una herramienta con el propósito de ayudar a la creación de apuntes de forma colaborativa se cumple. Sin embargo, no se logró el objetivo de realizar pruebas con en un ambiente real de un curso. Adicionalmente, se discute la posibilidad de trabajo futuro derivado de las falencias del trabajo realizado en esta memoria, en específico la forma de notificar a un usuario sobre cambios realizados mediante otros usuarios.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Contexto y problemática

Cuando un alumno finaliza el colegio e ingresa a la universidad de su elección experimenta muchos cambios en su vida —probablemente, exista un cambio de región o ciudad, vivir de forma independiente por primera vez, interactuar con personas nuevas— incluyendo un aumento en la envergadura y complejidad de clases, requiriendo cantidad de apuntes más numerosa que en el periodo de colegio. Por lo anterior, se dificulta la retención del contenido visto en clases, y por consiguiente se utiliza la toma de apuntes como un pilar fundamental para el repaso y estudio del contenido académico.

Especialmente cuando se acerca el periodo de exámenes, el tener una fuente de información concisa para el estudio produce un aumento considerable en el rendimiento, al contrario de un alumno que no haga registro de apuntes de la clase. Esta es la razón por la que estudiantes necesitan una correcta recopilación de apuntes e información en un corto periodo de tiempo, pues tiene un alto valor complementario al estudio [3].

Ahora, continuando con los exámenes, es en el estudio grupal previo a este periodo donde surgen nuevas problemáticas, ya que, debido a las diferencias entre todos los integrantes del grupo, se observa que la calidad de cada apunte puede ser diferente, complicando la compilación del conocimiento en estos. Para evitar esto, se pueden complementar los apuntes con fuentes externas, cuyo propósito será perfeccionar el contenido de la clase, enfocándose en la evaluación venidera.

Entre estos apuntes se esperaría encontrar diferentes técnicas utilizadas, como por ejemplo: escritos a mano, mapas mentales, o transcripciones literales de lo dicho en clases. Cada uno de estos diferentes enfoques a la toma de apuntes presenta diferentes ventajas y desventajas, pero no existen estudios concluyentes respecto a la superioridad absoluta de uno de ellos [9].

Sin embargo, al momento de realizar esta consolidación, cada miembro del equipo puede haber tenido un enfoque diferente en el contenido de la clase. O, por otro lado, el material complementario —en caso de ser un material entregado por el profesor— puede tener un

énfasis diferente al que tuvo la clase particular, donde se enfatizó en una sección particular no presente en el libro o apunte entregado.

Una forma de intentar sobrellevar esta problemática puede ser efectuar esta consolidación de forma online. Lo cual permite una escritura entre los miembros del grupo mucho más rápida para poder discutir diferencias entre los apuntes de cada miembro, permitiendo disminuir la carga cognitiva individual de cada uno de los miembros con respecto al estudio [3]. Para esto existen herramientas como Google Docs [6], Notion [15], Wikipedia [25], entre otras. que permiten la realización de un único documento compartido. Además, como estas herramientas existen de forma online, la recopilación de material externo se vuelve más fácil [22].

En particular, dentro de las herramientas mencionadas, destaca Wikipedia por la vasta cantidad de conocimiento que contiene, la cual, si bien existen discusiones sobre la calidad de su contenido [4] [23] [27], mantiene la postura que, "mientras mayor sea el conocimiento que pueda proveer dentro de sus limitaciones, mejor será este" (traducción propia desde [26]), por lo que invita a la mayor cantidad de personas posibles a aportar en el contenido del sitio.

Además de las herramientas externas mencionadas, la Facultad de Ciencias Físicas y Matemáticas presenta el material docente —donde el profesor puede subir un equivalente de un apunte de clase— o los foros de curso, donde los alumnos pueden realizar preguntas y recibir respuestas de sus compañeros o de parte del equipo docente.

A pesar de esto, la presencia de las herramientas antes mencionadas no resuelve una dificultad que proviene de las diferencias entre los apuntes de los miembros del equipo o entre el apunte entregado por el profesor y el contenido real visto en la clase, ya que se puede haber hecho un énfasis particular en un contenido específico. Adicionalmente, algo que no se toma en cuenta al momento de realizar el traspaso de apuntes a mano a un apunte digital, es que existen personas que preferirán mantener sus apuntes de forma manual, y no querer volver a escribirlos en la nueva plataforma.

Considerando lo anterior se propone un sistema que permita la edición de forma asíncrona, para poder abordar el caos que la edición en tiempo real puede producir. Adicionalmente, se presenta la oportunidad de aprovechar los apuntes de quienes los escriben a mano normalmente, mediante un sistema que permita el traslado de estos apuntes desde lo físico a lo digital.

Para esto, se plantea la creación de un módulo dentro de la plataforma de *U-cursos*, que permita a estudiantes, compartir el contenido de sus apuntes en documentos, donde todos los integrantes de un curso, tanto profesores como alumnos, puedan editar el documento libremente. Adicionalmente, se propone la adición de un sistema OCR para ayudar a estudiantes que tomen apuntes en papel y hacer que puedan agregar el contenido de estos a un documento de forma más rápida.

1.2. Objetivos

Teniendo en cuenta la problemática mencionada anteriormente y la forma de abordarla, se definen los siguientes objetivos sobre el trabajo realizado.

1.2.1. Objetivo General

El objetivo general de esta memoria es diseñar e implementar un sistema de apuntes para curso en la plataforma *U-Cursos*, que permita la edición en tiempo diferido de sus contenidos, sin dejar de lado a quienes prefieran mantener sus apuntes en papel ofreciendo una metodología de transcripción automática a digital. Para, de esta forma, permitir la construcción de un apunte que permita reflejar la realidad del curso correspondiente.

1.2.2. Objetivos Específicos

1. Analizar la cantidad de información que se puede extraer de un apunte escrito a mano mediante una implementación de OCR.
2. Diseñar una interfaz que permita la edición del contenido de forma sencilla por los usuarios de la plataforma *U-cursos*.
3. Diseñar la arquitectura necesaria para la ejecución y escalabilidad del sistema propuesto.
4. Diseñar un prototipo funcional que permita extraer texto de una página escrita a mano.
5. Implementar los puntos anteriores dentro de la plataforma *U-Cursos*.
6. Validar el sistema creado con miembros de un curso real, para evaluar la usabilidad de lo realizado, durante un periodo de 3 semanas, y recopilar las opiniones de estos.

1.3. Solución

Teniendo en mente los objetivos planteados, la solución propuesta consiste en un módulo para la plataforma *U-cursos*, consistente en un documento dividido en cuadros que pueda ser editado por todos los miembros del curso donde se active este módulo.

El módulo permitirá la edicion de los documentos dentro de este de forma asíncrona y utilizará una herramienta OCR para facilitar el traspaso de un apunte escrito de forma manual a la plataforma digital.

1.4. Metodología

Con el fin de abordar los objetivos planteados, primero se realizó una etapa de exploración de diferentes alternativas de OCR para comparar sus diferentes fortalezas y debilidades, entre las cuales fue considerada la mejor alternativa, la plataforma *Azure Cloud Vision*. Una vez se tuvo una alternativa preferida, se hizo una encuesta a diferentes estudiantes con el objetivo de tener información sobre la forma de toma de apuntes de estos, donde se hizo notar que una gran mayoría toma apuntes en papel y no todos realizan un proceso de "pasar en limpio" sus apuntes, por lo que el módulo a crear debe tener un enfoque en estas personas que si le sacarían ventaja a una herramienta como esta.

Una consideración importante antes de continuar con la metodología utilizada es que *U-Cursos* y Ucampus son herramientas desarrolladas por el Centro Ucampus. *U-Cursos* se preocupa del apoyo de la docencia, Ucampus para la gestión curricular. Si bien esta diferencia es evidente en la Universidad de Chile, en otras universidades estas dos herramientas son conocidas como una sola Ucampus, por lo que la utilización y nombrar ambas puede darse de manera indiferente dentro de la memoria.

Luego se hizo un análisis a la estructura de la plataforma *U-Cursos* dentro de Ucampus para la organización y el flujo de los datos a utilizar, para luego definir los casos de uso del módulo a crear y entender correctamente el funcionamiento de la plataforma OCR escogida para su implementación.

Posteriormente, se realizó el diseño de las interfaces y bases de datos necesarias para el funcionamiento de la herramienta. Para poder almacenar la información necesaria fueron necesarias 3 tablas, `DOCUMENTOS`, `CUADROS` y `CONFLICTOS`. Para las interfaces, fue necesaria la definición de las diferentes formas en que el usuario puede interactuar con los documentos y cuadros mediante el módulo.

Después, se empieza el proceso de implementación de la herramienta, donde se pudo observar dificultades en la implementación con el sistema OCR que determinaron la necesidad de una nueva tabla `IMAGENES` para almacenar un caché de imágenes que permita un acceso mucho más expedito a la información extraída de esta, alivianando la carga a la *API*.

Por último, durante la implementación del módulo se realizaron dos validaciones con el equipo de desarrollo del Centro Tecnológico de Ucampus, que permitieron recibir una retroalimentación sobre el estado actual del módulo en ese momento, además, una vez finalizado el módulo, se realizó una validación con 5 estudiantes de diferentes universidades del mundo, donde se les pidió realizar una serie de actividades, se aplicó la escala SUS para evaluar el uso de la herramienta y se tuvo una conversación abierta con ellos.

1.5. Estructura del documento

Ahora, es pertinente hablar de la estructura del presente documento, los capítulos que lo componen y el orden de estos.

En el capítulo Estado del arte se hablará de las diferentes tecnologías de toma de apuntes y las diferentes plataformas de OCR existentes.

En el capítulo Análisis se expandirá en la encuesta realizada a los diferentes estudiantes y los resultados de esta, además de la estructura de la plataforma de Ucampus en general, para terminar con los casos de uso definidos para el módulo a crear.

Luego, en el capítulo de Diseño, se aborda la sincronicidad de la plataforma, las interfaces creadas y el modelo de datos a utilizar.

Después, en el capítulo de Implementación, se muestran los esquemas de las bases de datos creadas, la implementación de cada una de las interfaces, la implementación del sistema OCR y el sistema de resolución de conflictos implementado.

Luego, en el capítulo de validación se expande en las dos validaciones realizadas, tanto con el equipo de desarrollo de Ucampus como con los estudiantes universitarios, entrando en profundidad a los detalles de los resultados obtenidos.

Finalmente, en el capítulo Conclusión se extraen las conclusiones del trabajo realizado y se comparan los resultados obtenidos con los objetivos planteados en esta introducción.

Capítulo 2

Estado del Arte

En este capítulo se hablará sobre las diferentes herramientas y tecnologías relevantes para el desarrollo de esta memoria. En la sección 2.1 se habla sobre las herramientas existentes para la toma de apuntes y que son utilizadas para la realización de apuntes compartidos, luego, en la sección 2.2 se hablara sobre las diferentes alternativas OCR y la evaluación que se realizó sobre ellas para decidir una a implementar.

2.1. Herramientas toma de apuntes

Actualmente, si un estudiante quiere valerse de herramientas digitales para, realizar apuntes, compartir estos apuntes, o consultar información externa, se tienen diferentes alternativas con diferentes fortalezas y debilidades. A continuación, se realiza una breve recopilación de algunas de estas herramientas, sin un orden en particular.

- **Notion:** Herramienta de recolección de información cuya funcionalidad más importante corresponde a la profundidad con la que se puede organizar la información en cada una de sus páginas, la cual puede tener enlaces circulares y permitir al usuario una organización personalizada, debido a su sistema de tags para cada una de las páginas que el usuario cree, y permitiendo organizar estas páginas según filtros altamente personalizables.

  Esta plataforma presenta formas de escritura colaborativa, sin embargo, la curva de aprendizaje es bastante elevada y no presenta formas de colaboración en tiempo real.

- **Google Docs:** Herramienta que presenta una forma sencilla de hacer trabajo colaborativo. Se necesita una cuenta de Google, y simplifica la acción de compartir trabajo mediante el uso de edición en tiempo real.

  Esta plataforma presenta una implementación básica de un sistema de OCR [5], no obstante, esta solamente funciona cuando se abre un documento en formato PDF o una imagen mediante la plataforma, no permitiendo agregar nuevas imágenes después de la primera, por lo que no se ajusta a la necesidad de un documento que permita texto escrito a mano de varios usuarios a la vez.

- **U-cursos:** Actualmente, la plataforma *U-cursos*, presenta funcionalidad para subir materiales de parte del equipo docente o del alumnado, pero estos materiales existen de forma independiente y no existe forma actual de consolidar los contenidos de cada uno de estos.

  Otra funcionalidad a destacar de *U-cursos*, es el foro existente para cada uno de los cursos, el cual permite la discusión rápida de temáticas relevantes para el curso en cuestión.

  Dentro este foro también se pueden enviar enlaces a otras plataformas y mantener de esa forma un documento actualizado, pero depende considerablemente de la participación de los miembros del curso correspondiente, lo cual presenta un desafío adicional para el sistema propuesto.

- **Wikipedia:** Esta herramienta presenta valor a la recolección de información, especialmente con las funcionalidades de versionamiento y edición libre en cada párrafo, aun así, la política de Wikipedia corresponde a mantener artículos de conocimiento general y de información específica con respecto al punto del artículo, por lo que un apunte de clase no corresponde. Además, el contenido de Wikipedia corresponde a contenido público, y algunos profesores prefieren mantener el contenido de sus clases dentro del contexto de estas, por lo cual, Wikipedia no serviría para este propósito.

2.2. OCR

Debido a la idea de incorporar un sistema de reconocimiento de caracteres a la solución a desarrollar, es necesario realizar una comparación entre algunas de las alternativas disponibles, con el fin de encontrar una que permita un desarrollo sencillo y de resultados utilizables. Estos sistemas se pueden categorizar de dos formas, aquellos que son código que se ejecuta localmente en la máquina de desarrollo y aquellos que corresponden a una API donde se pueden obtener resultados.

2.2.1. Alternativas comparadas

Al realizar la investigación sobre plataformas de OCR, se encontraron variadas alternativas, las cuales se concentraron en 5 para realizar pruebas más profundas, estas alternativas son las siguientes:

- *Microsoft Azure Vision*
- *Google Cloud Vision API*
- *Keras-OCR*
- *EasyOCR*
- *Keras + IAM*

Las dos primeras alternativas surgen a partir de las soluciones disponibles de dos de las empresas más grandes en el área de la tecnología, correspondiendo a *Google* y *Microsoft*.

Luego, se encontró el framework Keras, que permite el desarrollo de sistemas de *Machine Learning* de forma local, en lo cual se basa la solución *Keras + IAM*. Adicionalmente, existe una implementación de Keras en un repositorio [12], con el objetivo de presentar un uso sencillo, esta implementación corresponde a *Keras-OCR*.

Finalmente, se encontró la alternativa *EasyOCR*, la cual tiene un sitio web [1] para realizar las pruebas de rendimiento.

2.2.2. Criterios de comparación

Para realizar la comparación de estos resultados, es importante primero definir los criterios que se utilizaran para realizar esta comparativa.

Basados en los criterios de software planteados en los criterios de calidad planteados en [2] fueron decididos buscando maximizar la eficiencia del usuario final, y también maximizar la portabilidad y usabilidad para el desarrollador de la aplicación.

Estos criterios son los siguientes:

- **Facilidad de Uso:** Este criterio corresponde a la dificultad de escritura del código necesario para utilizar la API o programa correspondiente. Se considera simple cuando se necesita un procesamiento pequeño de los datos, o una plantilla de código simple. Se considerará complejo cuando se requiere un procesamiento de los datos a enviar o de los datos recibidos.
- **Requisitos de hardware:** Este criterio corresponde a la capacidad necesaria del hardware donde se requiere utilizar el sistema. Esta evaluación se hace en términos relativos, ya que especificar una cantidad específica de software necesaria se escapa de los alcances de esta memoria. Se considerará bajo cuando se necesita hardware de poca potencia para la utilización del software, en general debido a que el proceso de detección se realiza de forma remota. Mientras que se considerara Medio cuando el proceso de detección se realice de forma local.
- **Precio:** Este criterio corresponde a los requisitos monetarios necesarios para ser capaces de utilizar la aplicación. En general, estos costos existen mayoritariamente para las soluciones mediante API, ya que el procesamiento se realiza en el servidor de la compañía correspondiente.
- **Detección:** Este último criterio corresponde al más importante, consiste en la calidad de los resultados obtenidos mediante el uso de las soluciones, donde un resultado más cercano a la realidad es lo ideal.

2.2.3. Código utilizado

Para poder realizar la comparación con respecto a los criterios definidos en el capítulo 2.2.2 se ocupó código para poder realizar pruebas a cada una de las 5 soluciones evaluadas.

```
1  # Call API with URL and raw response (allows you to get the operation location)
2  with open('images/test.jpg', 'rb') as image_stream:
3      read_response = computervision_client.read_in_stream(
4          image_stream, raw=True,language='es'
5      )
6
7  # Get the operation location (URL with an ID at the end) from the response
8  read_operation_location = read_response.headers["Operation-Location"]
9
10 # Grab the ID from the URL
11 operation_id = read_operation_location.split("/")[-1]
12
13 # Call the "GET" API and wait for it to retrieve the results
14 while True:
15     read_result = computervision_client.get_read_result(operation_id)
16     if read_result.status not in ['notStarted', 'running']:
17         break
18     time.sleep(1)
```

Código 2.1: Código ejemplo de uso de *Microsoft Azure Vision*

```
1  def detect_document(path):
2      """Detects document features in an image."""
3      from google.cloud import vision
4      import io
5      client = vision.ImageAnnotatorClient()
6
7      with io.open(path, 'rb') as image_file:
8          content = image_file.read()
9
10     image = vision.Image(content=content)
11
12     response = client.document_text_detection(image=image)
13
14 detect_document("images/test.jpg")
```

Código 2.2: Código ejemplo de uso de *Google Cloud Vision API*

Los códigos 2.1 y 2.2 corresponden a los códigos utilizados para realizar las pruebas con las plataformas de Microsoft y Google, respectivamente. Se puede observar que ambos códigos presentan similitudes en su funcionamiento donde, teniendo una ruta particular hacia una imagen en el disco del usuario, se realiza una lectura de los datos para realizar la petición correspondiente, en las líneas 2 a 5 para el código de *Microsoft* y 7 a 8 para el código de *Google*.

Posteriormente, se realiza la solicitud al cliente correspondiente y se recibe la respuesta, en las líneas 8 a 18 para el código de *Microsoft* y 10 a 12 para el código de *Google*.

```
1  # keras-ocr will automatically download pretrained
2  # weights for the detector and recognizer.
3  pipeline = keras_ocr.pipeline.Pipeline()
4
5  images_dir="images"
6  custom_images = []
7
8  for filename in os.listdir(images_dir):
9      print(os.path.join(images_dir, filename))
10     custom_images.append(os.path.join(images_dir, filename))
11
```

```
12 # Get a set of three example images
13 images = [
14     keras_ocr.tools.read(path) for path in custom_images
15 ]
16
17 # Each list of predictions in prediction_groups is a list of
18 # (word, box) tuples.
19 prediction_groups = pipeline.recognize(images)
```

Código 2.3: Código ejemplo de uso de *Keras-OCR*

Luego, para el código relacionado a la solución *Keras OCR*, presente en el código 2.3, se vuelve a realizar una lectura de los datos presentes en el directorio, en las líneas 5 a 15. Sin embargo, al momento de realizar el reconocimiento, se tiene que la misma función, en la línea 19, da los resultados sin requerir un procesamiento adicional, ya que el análisis de la imagen se hace de forma local en la máquina del usuario.

Adicionalmente, para la solución *EasyOCR* no se tiene un código explícito, debido a que se utilizó la página web de la solución [1], donde se permite realizar una prueba del código con una imagen desde el disco del usuario, con lo que se realizaron las pruebas.

Finalmente, para el uso de la solución *Keras + IAM*, se utilizó el código presente en el siguiente Júpiter Notebook [14], el cual tiene una complejidad bastante alta, y presento bastantes dificultades para su uso, debido a que el proceso de entrenamiento superaba los 30 minutos, y finalmente no se logró realizar una detección con imágenes distintas a las que provenían de la base de datos utilizada.

2.2.4. Evaluación

Alternativa	**Facilidad de Uso**	**Requisitos de hardware**	**Precio**	**Deteccion**
Microsoft Azure Vision	Simple	Bajo	$1 cada 1000 request	Utilizable
Google Cloud Vision API	Simple	Bajo	$1.5 cada 1000 requests	Utilizable
Keras-OCR	Simple	Medio	Gratis	Poco utilizable
EasyOCR	-	-	Gratis	Inutilizable
Keras + IAM	Complejo	Medio	Gratis	

Tabla 2.1: Resumen Sistemas ML

Los resultados encontrados se resumen en la tabla 2.1, a continuación, se realizará una explicación de cada columna y la forma en que se realizaron las comparaciones.

- **Facilidad de Uso**: Los resultados encontrados fueron que la mayoría de las soluciones tienen un uso simple, donde solamente se requiere un procesamiento menor a las imágenes para luego utilizar las API o librerías correspondientes. Debido a que para la solución *EasyOCR* no fue necesario el uso de código, se omite su evaluación para este criterio. Como excepción se tiene la última solución evaluada, *Keras + IAM*, la cual, debido a la complejidad del código requerido para su uso, presento considerables dificultades al momento de realizar pruebas.

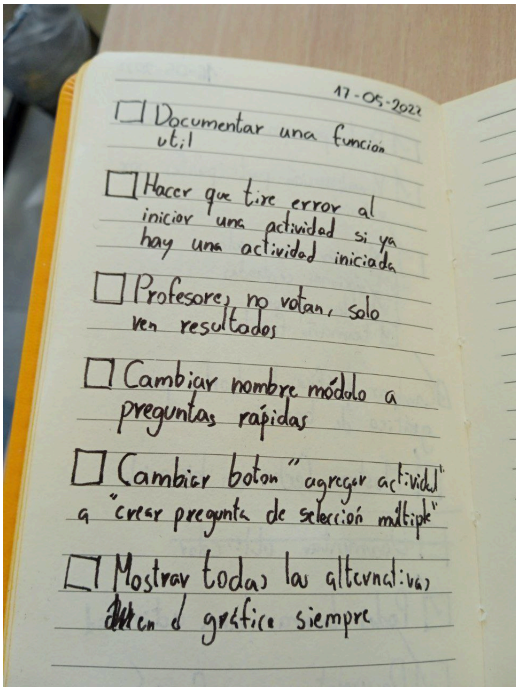

Figura 2.1: Texto de prueba

- **Requisitos de hardware**: Para este criterio, se encontró que las soluciones basadas en API presentan bajos o cercanos a nulos requisitos de hardware, puesto que solo es necesaria una conexión a Internet.

  Para la solución *EasyOCR*, nuevamente, debido a que las pruebas se realizaron en un entorno web, no fue posible realizar una comparación con respecto a requisitos de hardware de esta solución, por lo que se omite el valor en la tabla 2.1.

  En el caso de *Keras-OCR* es necesario un hardware de mediana potencia, debido a que el uso de los modelos para extraer el texto de la imagen se hace de forma local, sin embargo, para la solución *Keras + IAM* es necesario realizar el proceso de entrenamiento del modelo, lo cual es costoso en términos de tiempo, pero no necesariamente aumenta la complejidad del hardware necesario.

- **Precio**: En esta categoría solamente dos de las soluciones evaluadas presentan costo, las que corresponden a las soluciones creadas por *Microsoft* y *Google*, donde la aplicación de Microsoft tiene un precio inicial menor, pero, a medida que el uso de cada una de las API aumenta, el precio disminuye, generalmente sobre el millón de transacciones.

- **Detección**: Para esta categoría, se ocupará un ejemplo práctico comparando los resultados de las diferentes plataformas basándose en la misma imagen, la cual puede verse en la figura 2.1.

  Este texto fue elegido, puesto que no se compone únicamente de un párrafo de texto escrito a mano, sino que tiene un listado de texto. Además, la foto del texto no está completamente centrada, lo que produce que el texto de la parte superior aparezca de un tamaño diferente al de la parte inferior. Por ende, la imagen que representa un caso más cercano a la realidad.

  Los resultados obtenidos de cada aplicación se pueden ver a continuación:

– *Microsoft Azure Vision* :

```
17-05-2022
Documentar una función
util
Hacer que tire error a
iniciar una actividad si ya
hay una actividad iniciada
Profesores no votan, solo
ven resultados
Cambiar nombre módulo a
preguntas rápidas
] Cambier boton" agreger actividad
a crear pregunta de selección múltiple"
Mostrar todas las alternativas
den o gráfico siempre
```

Se puede observar que la estructura general se mantiene, y es posible, con dificultad, identificar el texto original dentro del resultado.

– *Google Cloud Vision*

```
âą Documentar una util
17-05-2022
funciÃşn
tire error al Hacer inicior que una activided si hay una actividad iniciada
Profesore , no votan , solo resultados Ven res
âą Cambiar nombre mÃşdolo a preguntas rÃąpidas
âą Cambier boton " agreger activad a " crear pregunta de selecciÃşn multipke "
âą Mostrar todas las alternativas anken & grafice siempre
```

Debido a que la plataforma de google intenta consolidar los resultados en diferentes párrafos, donde no determina un orden de estos, el resultado, si bien es similar al de microsoft, tiene la información organizada de una forma poco conveniente.

– *Easy-OCR*

```
47 05 2o2' Documentar fsncion una vl;l Haccr 9*_lsx CYYOY iniciv Adivdad un hy S;
     oclivido) Ys un inicìada ]Profesore votan sescltoos p Solo Ven
    Cambiar_hombrg_rodlo Pregunlas rafidas (ambcr_balon achvu' 2ant06 mdlipk" d
    Cresr Pregynks Hostvav toda)_l_gllwnd:vs) Auen 5iemprt Jvfi0
```

Esta solución presenta una detección bastante pobre, donde solamente algunas palabras son legibles, esto debido a que se permite la entrega de un hint sobre el lenguaje. Además, debido a que la detección se realiza de forma separada por cada palabra, no existe concepto de espacio, teniendo el resultado en una única línea, haciendo que sea muy difícil la diferenciación entre frases o párrafos.

– *Keras-OCR*

```
ate os 2oz doc umentar funcion unc tal v haccr tsc qu cryov l q inic ioy
acdtvicad unc hoy si yc uns actividod inicka dc pyofesove votana no j solo
resoltodos ven cambiay nombye modolo pygunias rapidas cambiby bolon ocrvda
agrcger sleco prgunt dc mlgipt cufsy c todas mosgtrgy la gltvncrin gratiae
wien siempre
```

Esta solución presenta un resultado aún peor que la anterior, posiblemente debido a que no permite enviar un hint sobre el idioma del texto a detectar. Además, prevalece el problema de que el resultado está únicamente en una línea.

– *Keras + IAM* Para esta solución no fue posible la realización de pruebas de detección, debido a la complejidad del código, por lo que se muestra vacío el resultado para esta categoría.

Debido a estos resultados, se utilizará el sistema propuesto por *Microsoft*, debido a que los resultados de detección son los mejores entre los sistemas evaluados. Adicionalmente, su uso es relativamente sencillo sin requerir hardware especializado. Y finalmente, si bien presenta un costo, este no es demasiado elevado.

Con esto se concluye el capítulo de Estado del Arte, con la revisión de diferentes herramientas existentes y elección de una herramienta OCR a utilizar, se puede avanzar a un análisis de cómo los diferentes estudiantes toman apuntes y las consideraciones que se determinan a partir de esto en el capítulo 3.

Capítulo 3

Análisis

En este capítulo se abordará la encuesta realizada a estudiantes en la sección 3.1 y el conocimiento extraído de esta, luego, en la sección 3.2 se discutirán las diferentes etapas en el proceso de toma de apuntes. A continuación, se realizará un análisis de la estructura de la plataforma donde se realizara el módulo en la sección 3.3, después, se realizara un análisis sobre la forma de uso de la plataforma OCR elegida en la sección 3.4. Y se finalizará definiendo los casos de uso del módulo a crear en la sección 3.5.

3.1. Exploración toma de apuntes de estudiantes

Para entender mejor la situación actual con respecto a la toma de apuntes de estudiantes de la Universidad de Chile, se realizó una encuesta abierta a estos, buscando entender mejor la experiencia de toma de apuntes de ellos.

3.1.1. Contexto

Debido a que la inspiración acerca de esta memoria surge como una anécdota personal del autor, es necesario ampliar el contexto de la misma recopilando información mediante una encuesta realizada en Google Forms. Las preguntas fueron formuladas por el autor de esta memoria y tratan sobre los hábitos que tienen los encuestados frente a la toma de apuntes regularmente.

La encuesta estuvo conformada por 8 preguntas, fue difundida mediante las plataformas *U-Cursos* y *Discord*, y se recibieron 97 respuestas de, mayoritariamente, estudiantes de la Universidad de Chile, la encuesta fue realizada del 23 al 30 de mayo del año 2022.

3.1.2. Preguntas

Las preguntas realizadas fueron las siguientes, de forma secuencial. Las preguntas 1, 2, 4, 6 y 7 fueron obligatorias, y el resto opcionales. La pregunta número 7 tiene una respuesta abierta, mientras que el resto presenta una serie de alternativas, y entrega la alternativa al encuestado de agregar otra alternativa si así lo desea.

1. ¿Qué nivel de educación estás cursando?
2. ¿Tomas apuntes regularmente?
3. Si tomas apuntes,¿de qué forma?
4. ¿Revisas tus apuntes después de la clase?
5. Si revisas tus apuntes, ¿Consultas fuentes externas?
6. Suponiendo que encuentras una inconsistencia entre la información online y lo que el profesor dijo en clases, ¿qué haces al respecto?
7. ¿Has realizado apuntes de forma colaborativa?
8. Si la respuesta anterior es si, ¿de qué forma?

3.1.3. Respuestas

A continuación, en las figuras 3.1- 3.8 se detallan las respuestas obtenidas luego de haberse realizado la encuesta, cabe destacar que la figura 3.7 no representa una correcta visualización de la respuesta, y será explicada en la sección 3.1.4.

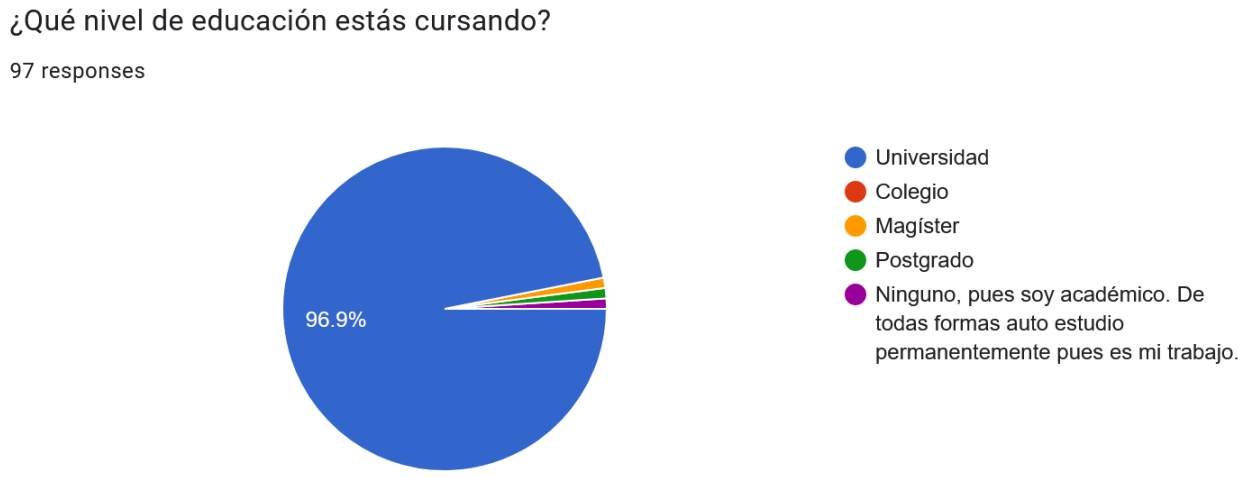

Figura 3.1: Nivel de educación de los encuestados

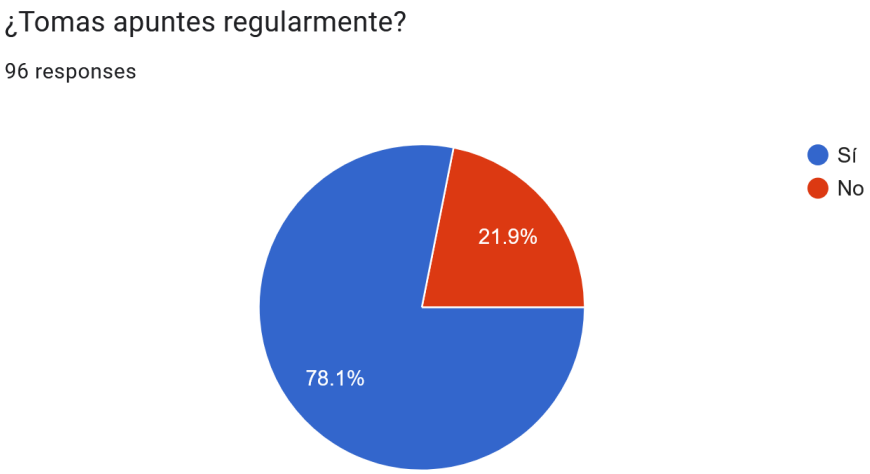

Figura 3.2: Frecuencia de toma de apuntes

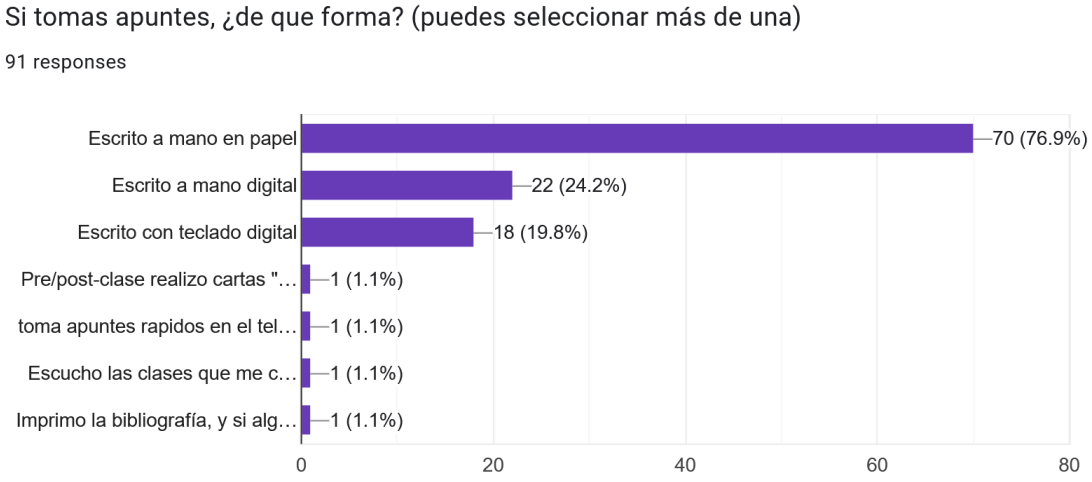

Figura 3.3: Formas de toma de apuntes

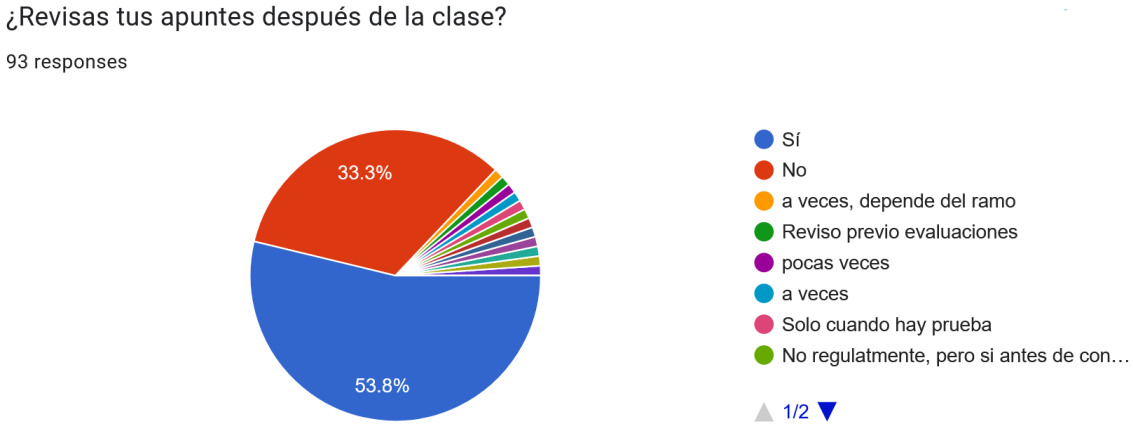

Figura 3.4: Frecuencia de revisión de apuntes tomados

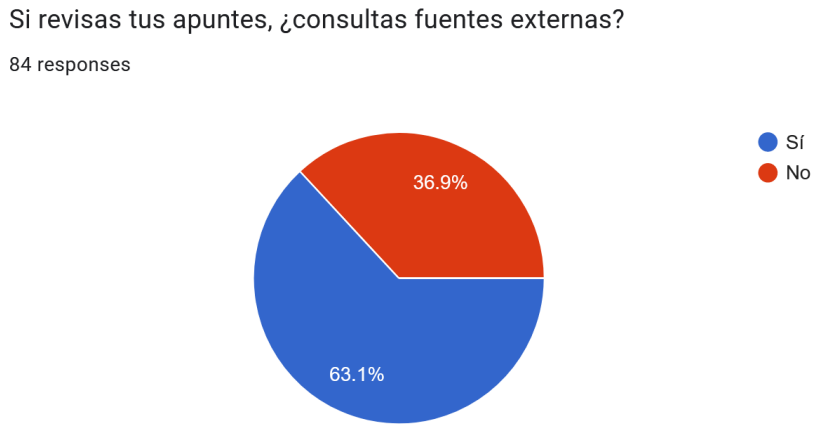

Figura 3.5: Consulta de fuentes externas al momento de revisión de apuntes

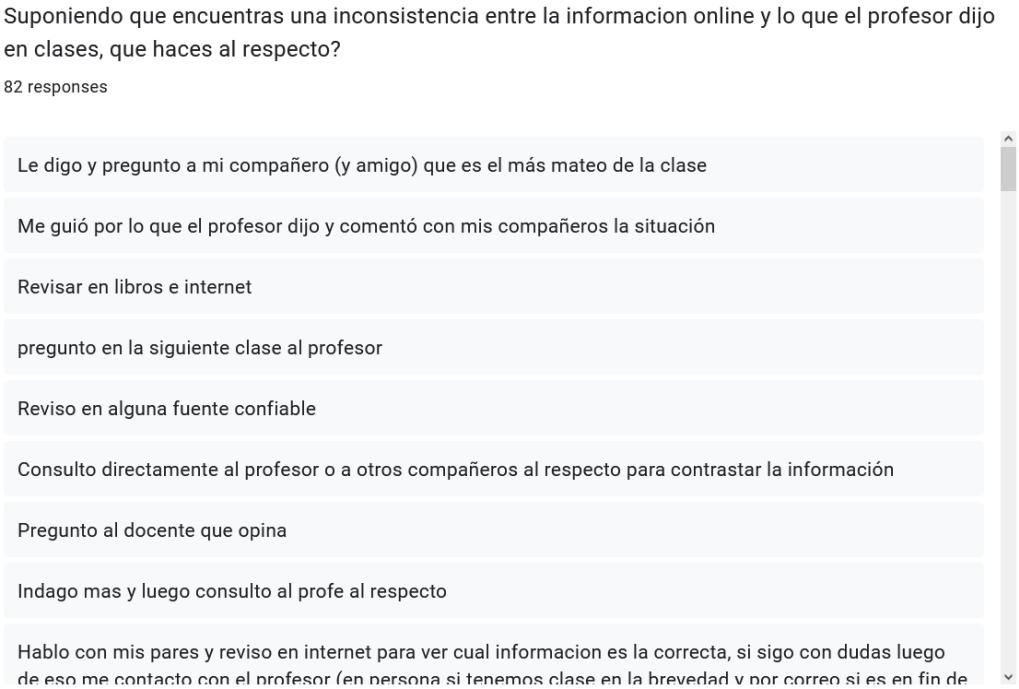

Figura 3.6: Formas de reacción ante inconsistencias

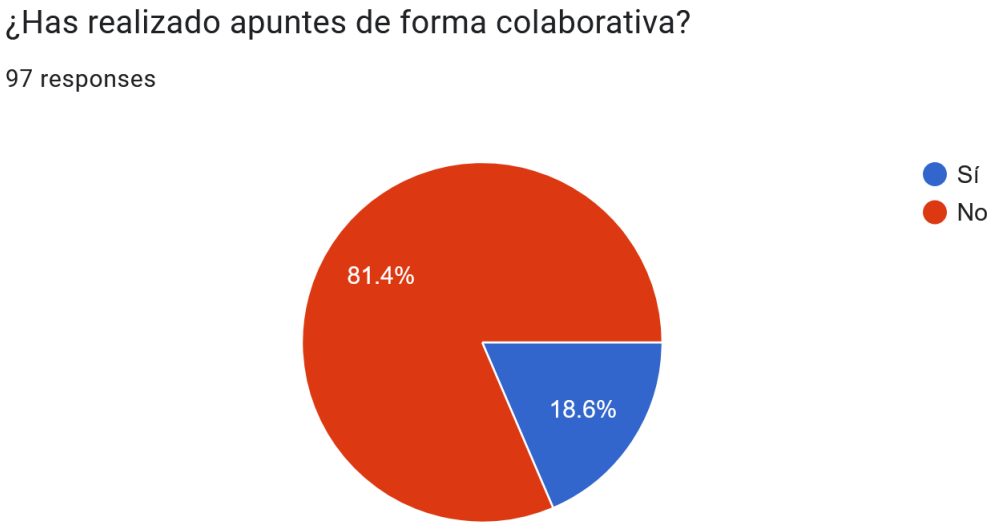

Figura 3.7: Conocimiento sobre la existencia de apuntes colaborativos

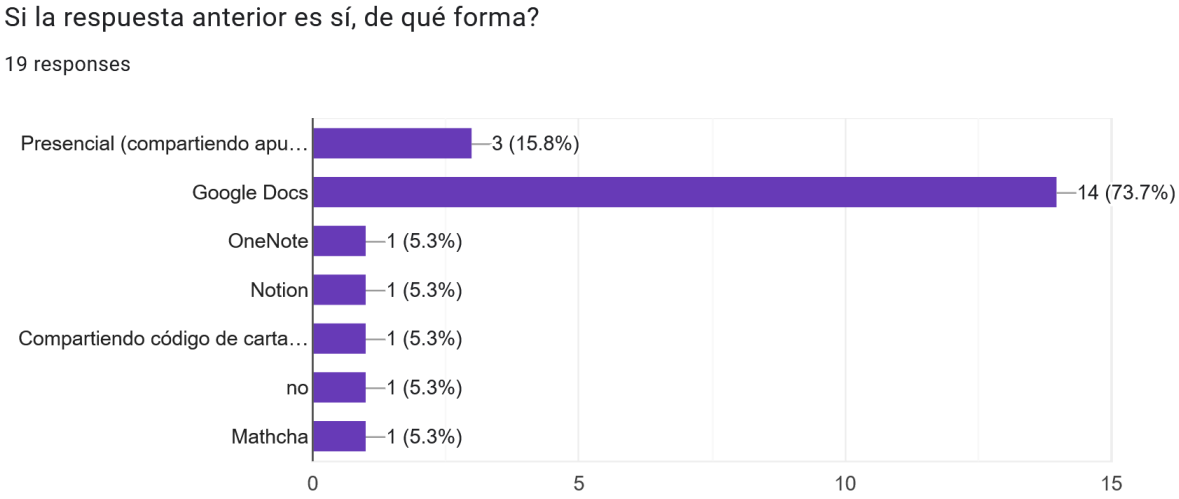

Figura 3.8: Plataformas utilizadas para la realización de apuntes colaborativos

3.1.4. Análisis

Para contextualizar las respuestas recibidas, se puede en la figura 3.1 que la mayoría de las respuestas fueron de estudiantes universitarios, en particular, de la Universidad de Chile.

Luego, las respuestas presentadas en las figuras 3.2 y 3.3, nos indican que la mayoría de los encuestados toma apuntes de forma regular y, mayoritariamente, de forma manual y en papel. Por ende, para el propósito de aportar en el apunte compartido que esta memoria plantea, es necesaria una forma sencilla para que estas personas no tengan que escribir sus apuntes múltiples veces, por lo que el uso del sistema OCR es una característica deseable del software a desarrollar.

La cuarta pregunta, presente en la figura 3.4, debido a que tenía la opción de tener una respuesta abierta, presento una variedad de respuestas además de las indicadas. Sin embargo, en general, las respuestas adicionales indican que la revisión se hace en los momentos anteriores de una evaluación, con el fin de prepararse para esta.

Adicionalmente, con respecto a la quinta pregunta en la figura 3.5, es importante saber si se utilizan fuentes externas al momento de realizar la revisión de estos apuntes, debido a que el sistema a realizar corresponde a la adición de una fuente adicional, y las respuestas entregadas nos demuestran que sería utilizada en caso de que se pueda conformar como una fuente externa confiable.

Con respecto a la sexta pregunta, que se puede ver en la figura 3.6 está enfocada en poder verificar cuál es la respuesta ante inconsistencias en el apunte, donde, si bien no se pueden apreciar la totalidad de las respuestas en la imagen, al realizar el análisis de las respuestas se llega a la conclusión de que se prefiere la consulta entre pares, o con el cuerpo docente antes de consultar a una fuente externa.

Este resultado se contradice ligeramente con la intención de esta memoria, ya que se desearía que el alumno realice las correcciones al apunte colaborativo de forma directa. Sin embargo, si se le realiza la consulta a otro miembro del alumnado o cuerpo docente, estos serían capaces de realizar las correcciones necesarias.

Finalmente, las últimas preguntas realizadas, representadas en la figuras 3.7 y 3.8 fueron enfocadas en medir si los encuestados tenían familiaridad con el concepto de apuntes colaborativos, por lo que se puede observar, menos de un quinto de los encuestados tienen experiencia con la realización de apuntes colaborativos, donde entre ellos la mayoría tiene experiencia con Google Docs, el que se podrá utilizar como inspiración para el diseño del software final.

3.2. Formas de toma de apuntes

Luego de la realización de esta encuesta y discusiones con los profesores guías, se llegó a la conclusión de dividir el periodo de toma de apuntes en 3 secciones diferentes, con el propósito de tener un enfoque más acotado y permitir definir los objetivos de esta memoria

de mejor forma, estas secciones se discutirán a continuación.

- **Durante una clase:** Cuando un estudiante toma apuntes durante la clase lo hace de forma poco ordenada, intentando seguir el ritmo del profesor. Esto produce que los apuntes de cada persona puedan ser muy diferentes, algunas personas enfocándose más en gráficos o diagramas, mientras que otras utilizan texto.

  En vista de esto, intentar consolidar los apuntes generados durante esta etapa sería bastante caótico y no corresponde al enfoque que se le quiso dar al trabajo planteado en esta memoria, sin embargo, este se puede plantear como un posible espacio de trabajo para otra memoria en el futuro.

- **Pasar en limpio:** Esta etapa de la toma de apuntes tiene la particularidad de que no todos los estudiantes la realizan, pero aquellos que la hacen se preocupan de mejorar los apuntes tomados en la etapa anterior a una forma más legible, e incluso, los comparten entre sus compañeros bajo su propia voluntad.

  Es en este punto que esta memoria tiene su enfoque, para permitir que los estudiantes que deciden realizar la labor de pasar en limpio tengan una plataforma más sencilla para elaborar un apunte más completo de forma colaborativa.

  Adicionalmente, se entrega la opción de transcribir su apunte en papel a uno digital de una forma mucho más rápida, mediante el uso de tecnologías de reconocimiento de caracteres.

- **Antes de una evaluación:** Finalmente, entre estas etapas del proceso de toma de apuntes, tenemos una etapa que se da antes de evaluaciones en los ramos correspondientes, donde un alumno decide tomar los apuntes que tiene disponibles y consolidarlos en un resumen pertinente a la evaluación.

  Estos apuntes recopilados pueden ser tanto personales como recopilados de los apuntes que se compartieron de la fase anterior. Una herramienta para esta etapa podría ser beneficiosa, pero se aleja de los alcances de esta memoria en particular.

Altruismo al compartir apuntes

Un punto importante a considerar sobre la fase de la toma de apuntes que el trabajo de esta memoria complementa es que los alumnos que comparten sus apuntes de forma voluntaria corresponden a una minoría.

Esta minoría dentro de los cursos será el grupo que ocupara la herramienta generada por esta memoria en particular, y, como se puede observar en la figura 3.2 la mayoría de los estudiantes toma apuntes en forma manual y en papel, por lo que trasladar los apuntes generados de esta forma a la plataforma digital es algo de suma importancia.

3.3. Estructura de *U-Cursos*

Ahora, teniendo el contexto sobre la toma de apuntes, también es necesario, para el entendimiento de esta memoria, comprender la estructura de la herramienta donde se realizó

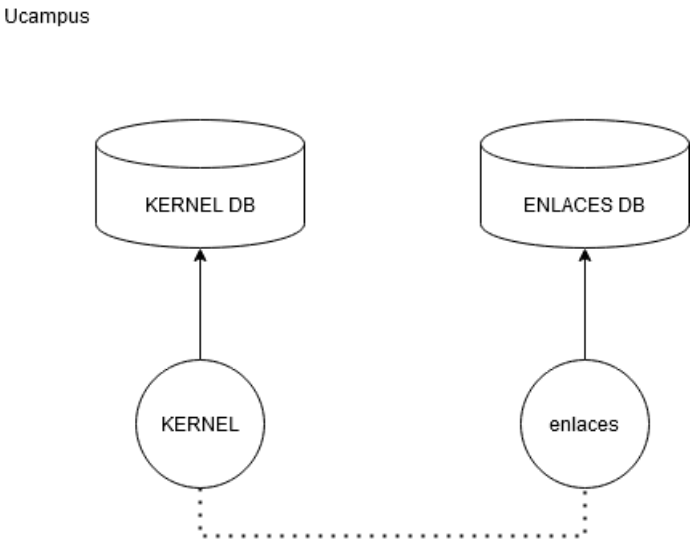

Figura 3.9: Estructura general de *U-Cursos*

la implementación de esta.

Esta plataforma corresponde a *U-Cursos*, desarrollada por el Centro Tecnológico Ucampus, es usada ampliamente dentro de la Universidad de Chile como plataforma de apoyo a la docencia.

3.3.1. Arquitectura general

La plataforma de *U-Cursos* contiene una gran variedad de funcionalidades, como el almacenamiento de notas, los foros institucionales y la toma de asistencia. Por ende, es necesario tener estas funcionalidades de forma separada en distintos módulos que pueden ser activados o desactivados en cada uno de los cursos de la plataforma.

Como base de estos módulos, está el módulo `KERNEL` que se encarga de almacenar los distintos grupos de personas y los permisos que estas tienen, siendo estos almacenados en su propio base de datos.

Luego, a partir de la información almacenada en el módulo `KERNEL`, se crean los diferentes módulos de la plataforma, donde cada uno se encarga de una funcionalidad específica. De esta forma, se aísla esa funcionalidad dentro de sus propias carpetas y bases de datos, permitiendo la separación de la información y protegiendo los datos del módulo central `KERNEL`. Algunos ejemplos de módulos con funcionalidades específicas son el módulo `notas`, `material docente` y `enlaces`.

Un diagrama de cómo un módulo almacena su propia información se puede ver en la figura 3.9, donde los módulos `KERNEL` y `enlaces` tienen sus propias bases de datos `KERNEL DB` y `ENLACES DB` sobre las que pueden tanto leer como escribir datos. Luego, el módulo enlaces puede solicitar información al módulo `KERNEL` en caso de ser necesario, representado por la línea punteada.

3.3.2. Arquitectura de un módulo

Un módulo en la plataforma *U-Cursos* está contenido dentro de una carpeta en el código de la plataforma, donde, dentro de esta, se tienen otras carpetas separando la funcionalidad siguiendo la arquitectura Modelo Vista Controlador. Para esta memoria en particular es necesario considerar 3 carpetas.

- `include`: Contiene el código correspondiente al modelo de la arquitectura, encargado principalmente de realizar consultas a la base de datos del módulo y organizar la información recopilada para ser usada en el resto del código.
- `template`: Corresponde a las vistas, respecto a la arquitectura MVC, contiene el código HTML correspondiente a cada una de las páginas correspondientes del módulo y el código *Javascript* y *CSS* necesario para su funcionamiento y apariencia correctos.
- `web`: Corresponde al controlador del módulo, se encarga de recibir la consulta de la URL desde el navegador, recopilar la información necesaria para el correcto funcionamiento mediante las funciones en la carpeta, *include* y hacer la renderización del código HTML de la carpeta *template*.

A continuación, para entender de mejor forma el flujo de información dentro de un módulo de la plataforma, en la figura 3.10 se muestra un diagrama de swimlane desde el momento que un usuario consulta una URL correspondiente al módulo hasta que se muestra la página al usuario.

3.4. Forma de uso de plataforma OCR

Otro punto importante a considerar, es como la plataforma seleccionada, *Azure Cloud Vision* realiza el flujo de información para la extracción del texto de una imagen.

Para comenzar a utilizar esta plataforma es necesario crear una cuenta en la página de *Azure Portal* [11], es necesaria la creación de un elemento de *Cloud Vision*. Una vez creado, se puede acceder a la url y contraseña necesarias para realizar las requests al sistema.

El funcionamiento para la extracción del texto es el siguiente:

1. Se envía una POST *request* a la url del elemento creado con la llave en los *headers* de esta y el contenido de la imagen en el *body*
2. La respuesta contendrá el *header Operation-Location* el cual contiene una segunda url
3. Se realiza una GET *request*a esta segunda url, la cual entrega una respuesta con 4 estados posibles
    - `notStarted`: No se ha iniciado el procesamiento de la imagen
    - `running`: La imagen se está procesando

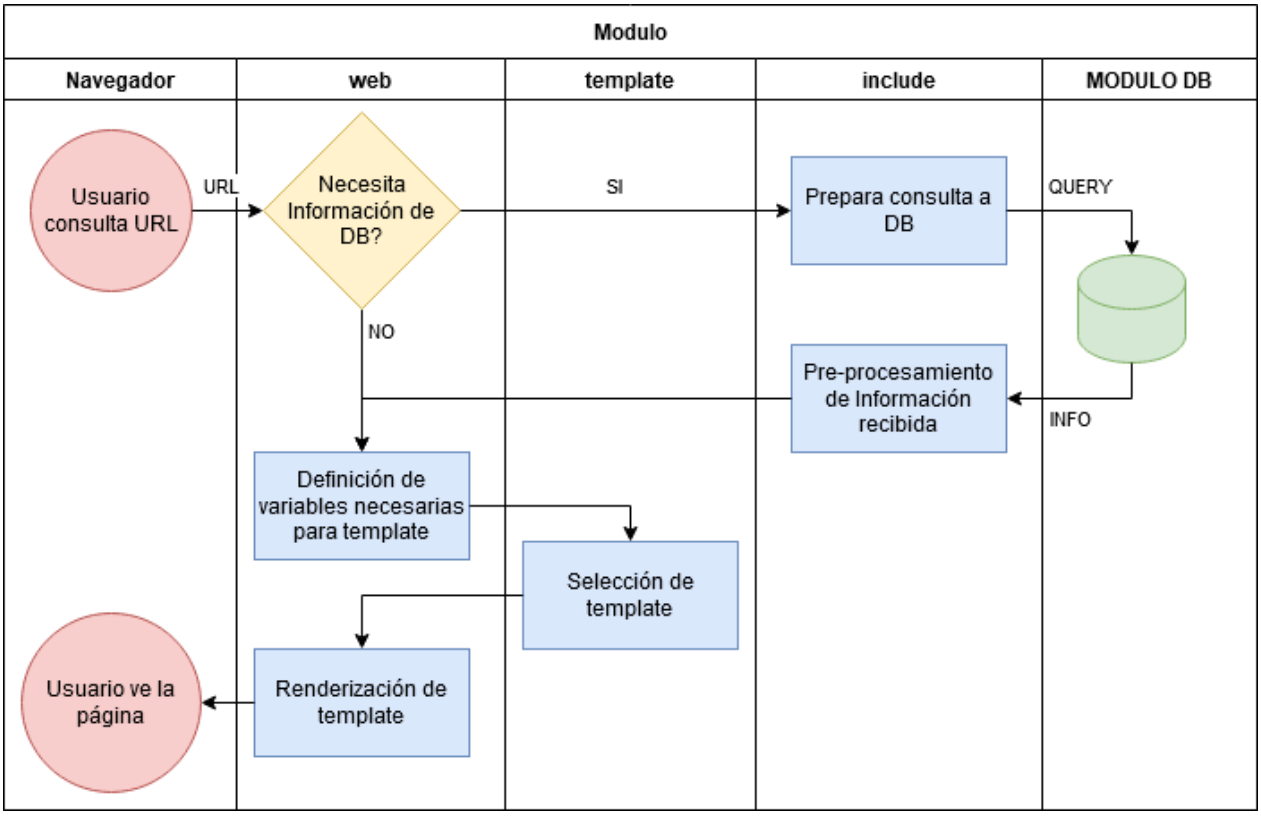

Figura 3.10: Flujo de un módulo abstracto

- `succeded`: La imagen ya ha sido procesada, se incluye adicionalmente el contenido procesado de la imagen, que contiene toda la información extraída de la imagen, pero solamente nos interesa el texto detectado
- `failed`: El procesamiento de la imagen ha fallado

Esta url estará disponible por 48 horas y luego se considerara inválida, en ese caso, es necesario enviar nuevamente la imagen para su procesamiento, mientras esto no ocurra, se pueden realizar todas las *requests* necesarias hasta obtener un resultado.

Por lo anterior, un posible diagrama de secuencia que represente los pasos necesarios para la extracción de un texto se puede ver en la figura 3.11.

3.5. Casos de Uso

Primero, al determinar el alcance de la aplicación, fue necesario determinar los casos de uso de esta, los cuales están resumidos en la figura 3.12.

Estos casos de uso se determinan a partir del objetivo que cada integrante del curso donde la herramienta esté disponible sea capaz de crear documentos y cambiar los permisos de estos, permitiendo así definir las personas que serán capaces de editar ese documento en particular. Esto se muestra en los primeros dos casos de uso.

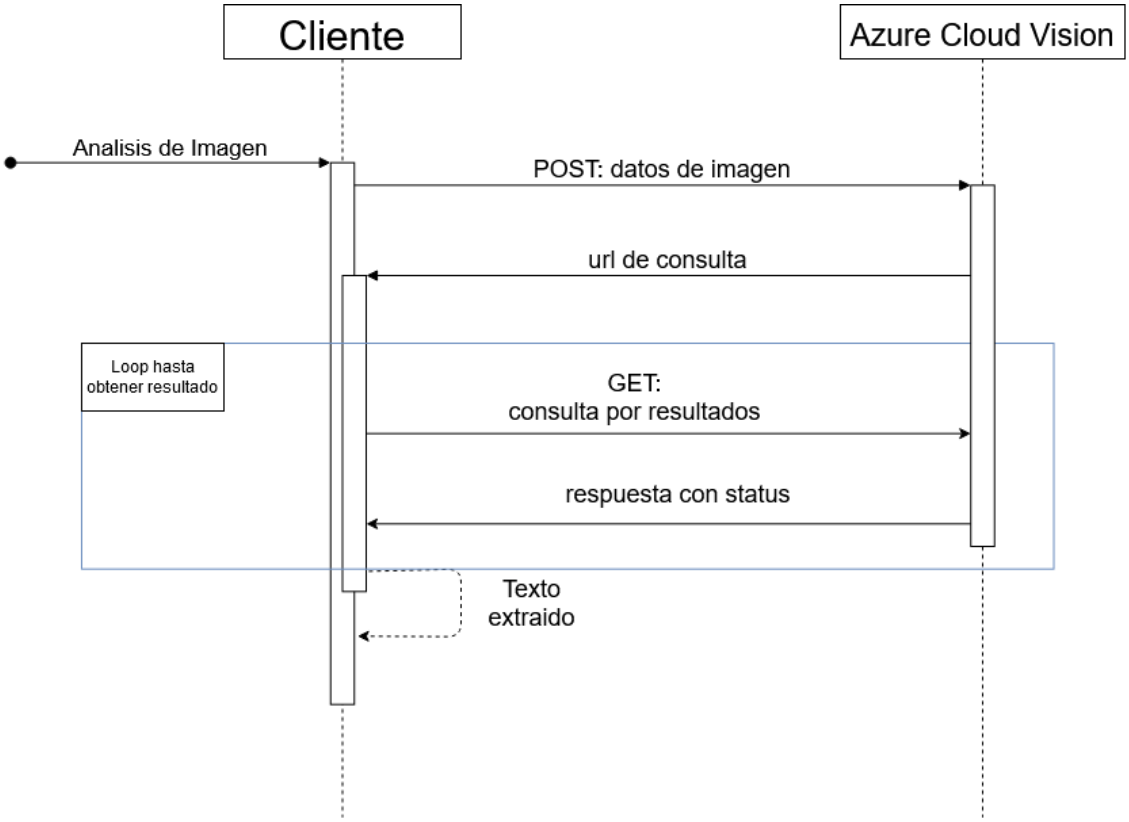

Figura 3.11: Diagrama secuencia funcionamiento plataforma OCR

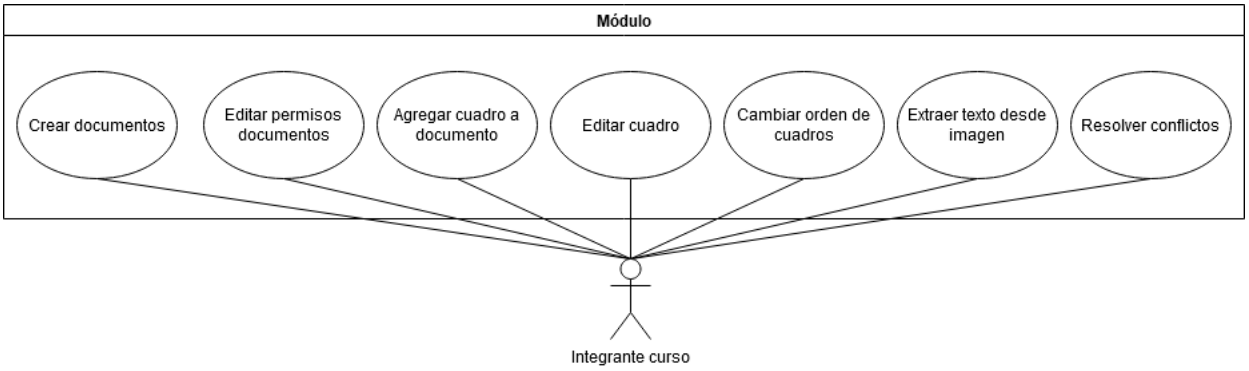

Figura 3.12: Casos de uso del módulo creado

Luego, una vez que un documento este creado y con los permisos correspondientes, la información de este será definida en cuadros, los cuales pueden contener texto o imágenes, según sea deseado. Se decide entonces que los miembros del curso que tengan permisos son capaces de crear cuadros y tienen poder de edicion sobre estos, que está definido en el tercer y cuarto caso de uso.

Además, en el proceso de edicion, un miembro del curso puede decidir que el orden de estos no es el correcto, por ende, se le da la posibilidad de editar este orden según lo considere necesario. Esto está definido en el quinto caso de uso.

Adicionalmente, cuando un usuario de la herramienta realiza un edicion de un cuadro, se puede producir el caso de que otro usuario también haya realizado una edicion en el mismo cuadro, por lo cual se genera un conflicto, y el usuario debe ser capaz de resolver estos conflictos, definido en el sexto caso de uso.

Finalmente, una de las cualidades del trabajo de esta memoria es la incorporación del sistema de reconocimiento de caracteres, que se discutirá con mayor profundidad en el capítulo 5 Por ahora, se define que un usuario de la herramienta debe ser capaz de utilizar este sistema para agregar texto, definido en el último caso de uso.

Capítulo 4

Diseño

En este capítulo se abordarán las decisiones tomadas en cuanto al diseño de la aplicación desarrollada. Primero, en la sección en la sección 4.1 se abordarán los distintos mockups creados y las funcionalidades de estos, luego, en la sección 4.2 se discutirán el diseño de las distintas tablas necesarias para el funcionamiento del módulo.

4.1. Diseño de Interfaces

Teniendo en consideración los casos de uso definidos en la sección 3.5, ahora se introducirán las diferentes interfaces diseñadas para el cumplimiento de estos. Sin embargo, cabe destacar que estos diagramas fueron realizados en las primeras etapas del desarrollo de la presente memoria, por lo que se realizaron cambios a estos durante su desarrollo. Aun así, presentan las funcionalidades más importantes de la herramienta desarrollada.

Los diagramas creados están basados en la interfaz ya existente de la plataforma *U-cursos*, por lo que se ocupa una captura de pantalla como base para cada diagrama.

4.1.1. Inicio

En la figura 4.1 se puede observar el título del documento y una lista de contenido que no fue desarrollada para esta memoria en particular.

Los cuadros mencionados anteriormente están claramente definidos en un color diferente al fondo, donde se aprecia que cada cuadro puede contener texto o imágenes.

Finalmente, se puede observar un botón al final de la página con la leyenda *Agregar sección*, que corresponde al tercer caso de uso especificado en la sección 3.5 y permite agregar un nuevo cuadro al final del documento actual.

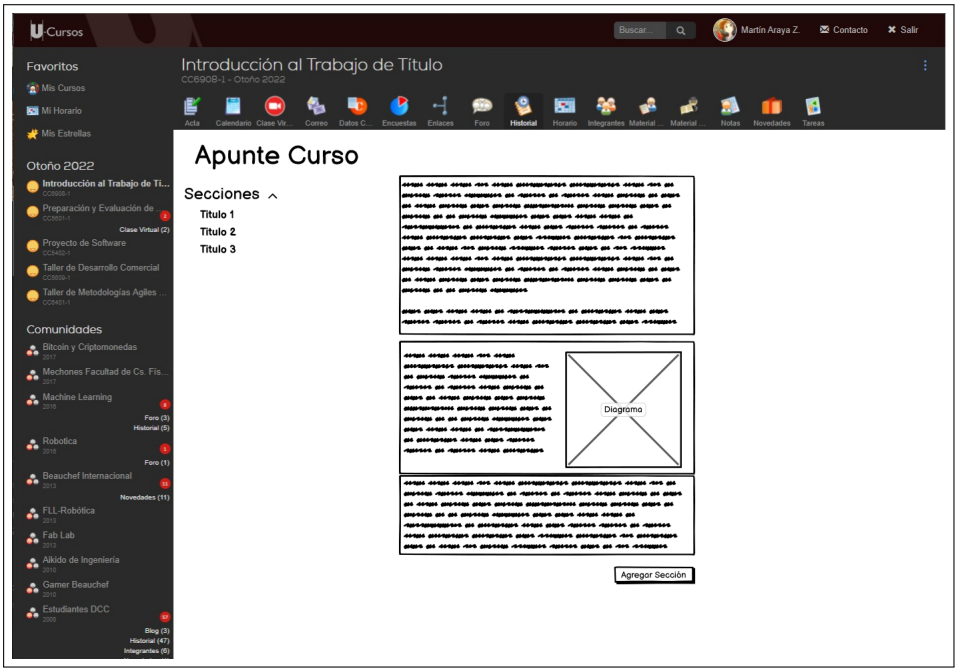

Figura 4.1: Página de Inicio del módulo

4.1.2. Agregar sección

En la figura 4.2 se puede apreciar la interfaz mediante la cual un usuario de la herramienta puede agregar un cuadro de texto al documento correspondiente.

En el cuadro donde el usuario puede ingresar el texto, además se tiene un botón con la figura de una cámara, que permitirá al usuario subir una imagen para realizar el proceso de extracción de texto mediante el sistema de *OCR* implementado en la herramienta.

Finalmente, se pueden observar dos flechas en los costados del cuadro de edicion, las cuales, en la fase de diseño, permitirían cambiar el orden del cuadro a agregar, sin embargo, esta funcionalidad fue cambiada en la herramienta final desarrollada.

Finalmente, en la figura 4.3 se muestra el resultado final, luego de que un usuario agregar el texto escrito al documento mediante el botón con la leyenda *Agregar*, se puede apreciar un cuadro de notificación dando a conocer al usuario que su cuadro fue agregado correctamente.

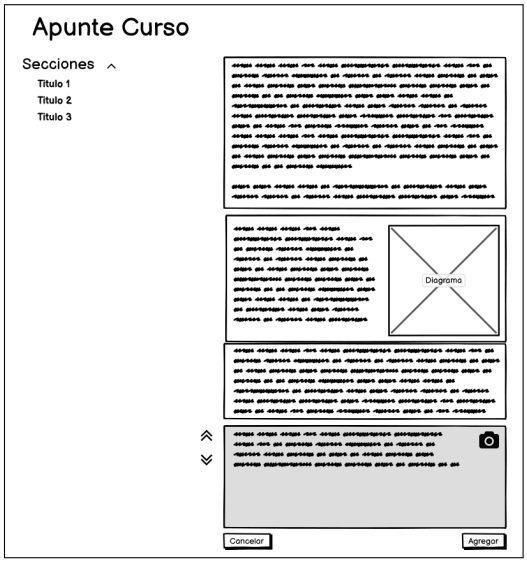

Figura 4.2: Página para agregar una nueva sección al texto

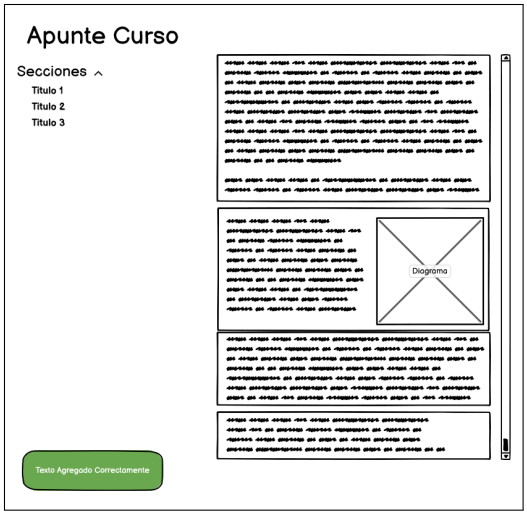

Figura 4.3: Página al agregar texto

4.1.3. Edición de cuadro

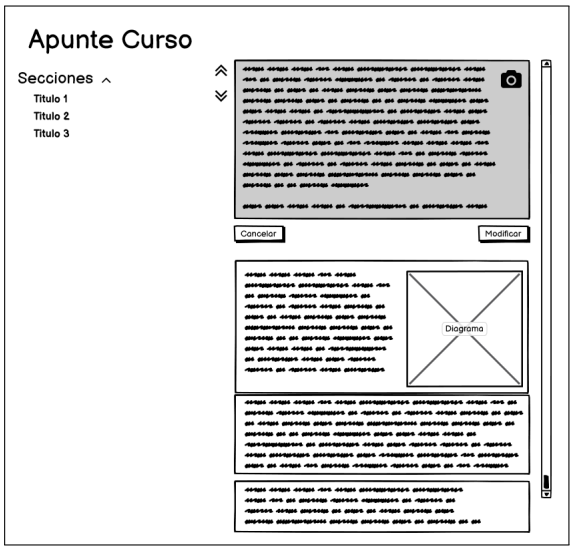

Figura 4.4: Edición de texto existente

Ahora en la figura 4.4 se puede apreciar la interfaz de edicion de un cuadro, esta aparece cuando el usuario realiza un doble click en uno de los cuadros ya existentes, el usuario podrá ver la misma interfaz con la que agrego un cuadro, pero con el texto existente agregado a esta.

Un punto importante considerado en esta memoria es la posibilidad de que dos usuarios realicen una modificación al mismo tiempo de un cuadro en particular, dado que la herramienta debe ser capaz de abordar esta problemática en detalle, esto será explicado en mayor profundidad a continuación.

4.1.4. Resolución de conflictos

Una de las consecuencias de generar una plataforma donde múltiples usuarios puedan realizar ediciones al mismo tiempo de forma asíncrona, se da el caso donde dos o más usuarios deciden modificar el mismo cuadro, esta ocurrencia se define como un conflicto y es necesario el diseño de un mecanismo que permita la resolución de estos.

Este mecanismo puede ser formulado de dos formas, como un sistema automático o un sistema manual. En este caso, como se desea que el contenido de los documentos creados este siempre en mano de los integrantes del curso que los contiene, se determina que el sistema debe ser de forma manual y la resolución de este debe ser de fácil resolución para el usuario que lo deba resolver.

Para esto, se decidió guardar el tiempo de realización de la última operación a cada cuadro en el servidor como T_c, y también se guarda de forma local el tiempo de carga de página del usuario T_u, de esta forma, se puede reconocer un conflicto cuando el usuario edite un cuadro

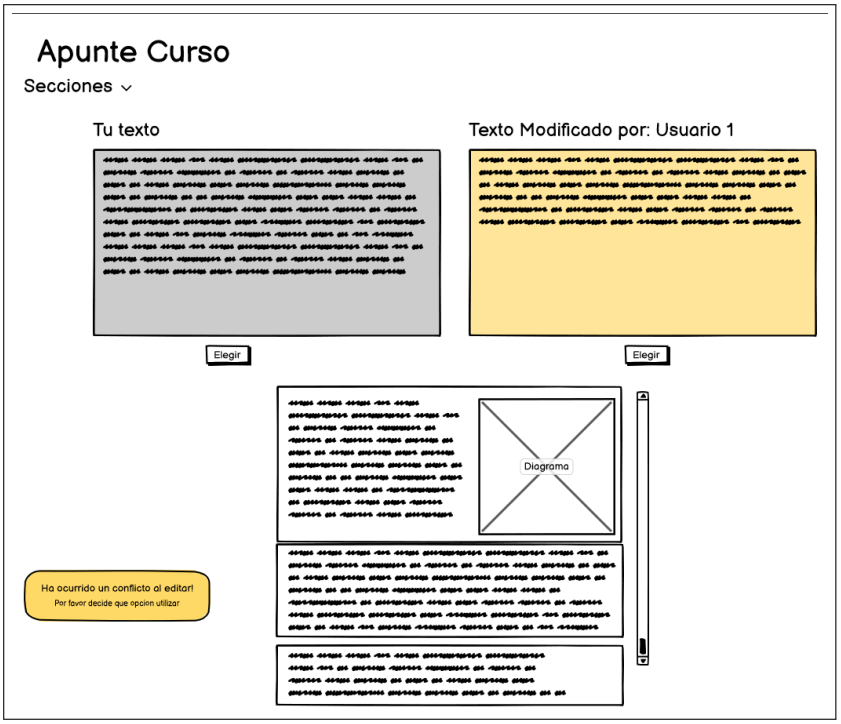

Figura 4.5: Interfaz resolución de conflicto

y se cumpla que $T_u < T_c$, lo cual significa que otro usuario realizo una modificación al cuadro de la cual nuestro primer usuario no tiene conocimiento.

Ante esto, se agregó la funcionalidad al usuario de poder observar cuando otros usuarios estén realizando la edicion de un cuadro, mediante un icono al costado del cuadro correspondiente. Debido a que esta funcionalidad no fue planeada en un comienzo, no se tiene diagrama al respecto.

Sin embargo, en caso de que la funcionalidad anterior no sea suficiente para evitar conflictos, se determinó que la mejor forma de actuar ante estos es dar a la persona que genera el conflicto (que llamaremos P_1) la opción de determinar si el texto editado por la otra persona (P_2) o el propio será el utilizado finalmente. Esta decisión fue tomada por las limitaciones de la generación de un *diff*, o archivo que muestra la diferencia entre textos, que sea fácilmente legible por cualquier usuario, ya que esta herramienta espera ser utilizada por personas de todas las facultades de la Universidad de Chile.

En la figura 4.5 se muestra el diseño de la interfaz de resolución de conflictos, en esta interfaz el usuario que resolverá el conflicto, donde podrá ver lado a lado ambos textos a comparar. En el costado izquierdo verá el texto escrito por sí mismo, y en el lado derecho el texto de la otra persona. Luego, cuando el usuario selecciona uno de los cuadros como correcto, podrá ver un cuadro de confirmación, como se puede ver en la figura 4.6.

Una edicion no visible en este diagrama es que al elegir un texto se le permitirá al usuario editarlo aún más, en caso de querer incorporar ideas de ambos textos mediante un tercer cuadro de edicion.

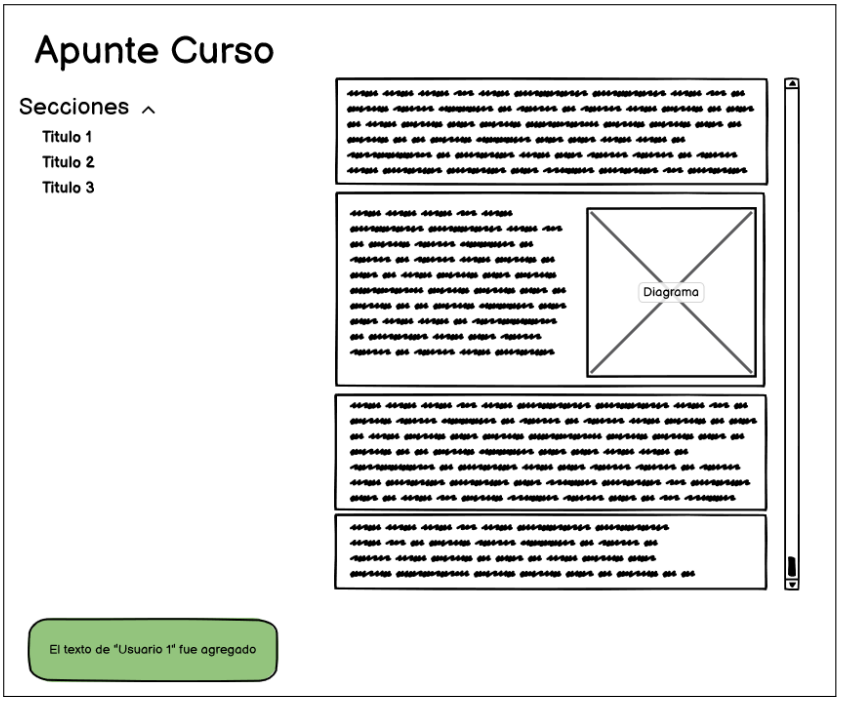

Figura 4.6: Conflicto resuelto mediante elección de texto

4.2. Modelo de datos

4.2.1. Sincronicidad

Debido a lo discutido en el capítulo 3, una gran parte de estudiantes realiza sus apuntes en papel, sin ocupar una plataforma digital. Como esta herramienta tiene como objetivo ser un complemento a la toma de apuntes y no un reemplazo, la sincronicidad no se definió como una característica deseada.

Aún más, debido a que el uso esperado de la herramienta desarrollada es por personas que estén en el proceso de "pasar en limpio", definido en el capítulo 3, es de esperar que estos usuarios escriban párrafos de gran tamaño mientras usen la herramienta, ante lo cual la sincronicidad de estos sería un problema, ya que la escritura de una persona se vería en conflicto con la de otra.

4.2.2. Diseño de tablas

Debido lo mencionado anteriormente, el modelo de datos de la herramienta debe ser lo suficientemente completo para abarcar todas las funcionalidades presentadas. Este debe ser capaz de mantener el estado actual de cada uno de los documentos presentes y los cuadros que lo componen. Dentro de los cuadros, se debe guardar el texto correspondiente y la fecha de modificación, de tal forma de poder detectar conflictos. Además, debe ser capaz de almacenar la información correspondiente a los conflictos para que el usuario final defina la resolución.

Con lo anterior, se determinó que serían necesarias 3 tablas, denominadas `DOCUMENTOS`, `CUADROS` Y `CONFLICTOS`, las cuales presentan las siguientes columnas.

- `DOCUMENTOS`
  - `ID`: Identificador único del documento
  - `CANAL`: Identificador del curso en el cual el documento fue creado
  - `TITULO`: Título del documento
  - `ORDEN`: Arreglo de identificadores de cuadros, representa el orden de estos dentro del documento
  - `ESTADO`: Estado del documento, puede ser borrado o activo
  - `ULT_MOD`: Datetime de última modificación al documento
  - `EXTRAS`: Información adicional del documento
  - `PERS_ID`: ID de la última persona que ha modificado el documento
  - `FECHA_M`: Fecha de la última modificación realizada al documento
  - `FECHA_C`: Fecha de creación del documento
- `CUADROS`
  - `ID`: Identificador único del cuadro
  - `DOCUMENTO`: Identificador del documento que contiene el cuadro
  - `ESTADO`: Estado del cuadro, puede ser activo o borrado
  - `CONTENIDO`: Contenido del cuadro, principalmente texto
  - `FECHA_M`: Fecha de la última modificación realizada al cuadro
  - `PERSONA_M`: ID de la última persona que ha modificado el cuadro
- `CONFLICTOS`
  - `ID`: Identificador del documento
  - `ID_CUADRO`: Identificador del cuadro donde se realizó el conflicto
  - `ID_DOCUMENTO`: Identificador del documento donde se realizó el conflicto
  - `CONTENIDO_CUADRO`: Contenido del nuevo cuadro que genero el conflicto
  - `ESTADO`: Estado del conflicto, puede ser activo o cerrado
  - `PERSONA`: Persona que desencadeno el conflicto

Mediante este modelo de datos, será posible almacenar toda la información relacionada con documentos y cuadros, cumpliendo de esta forma con los casos de uso 1, 2, 3, 4, 5 y 6, los cuales se refieren a la capacidad de crear cuadros, documentos y la posibilidad de editar estos. Mientras que el caso de uso 7 tiene más desarrollo en la implementación de esta herramienta, en el capítulo 5. Adicionalmente, se puede ver las relaciones entre las tablas en la figura 4.7.

Ahora, ya que se ha mostrado el diseño de la herramienta, en la 5 se discutirá la implementación de esta en mayor detalle.

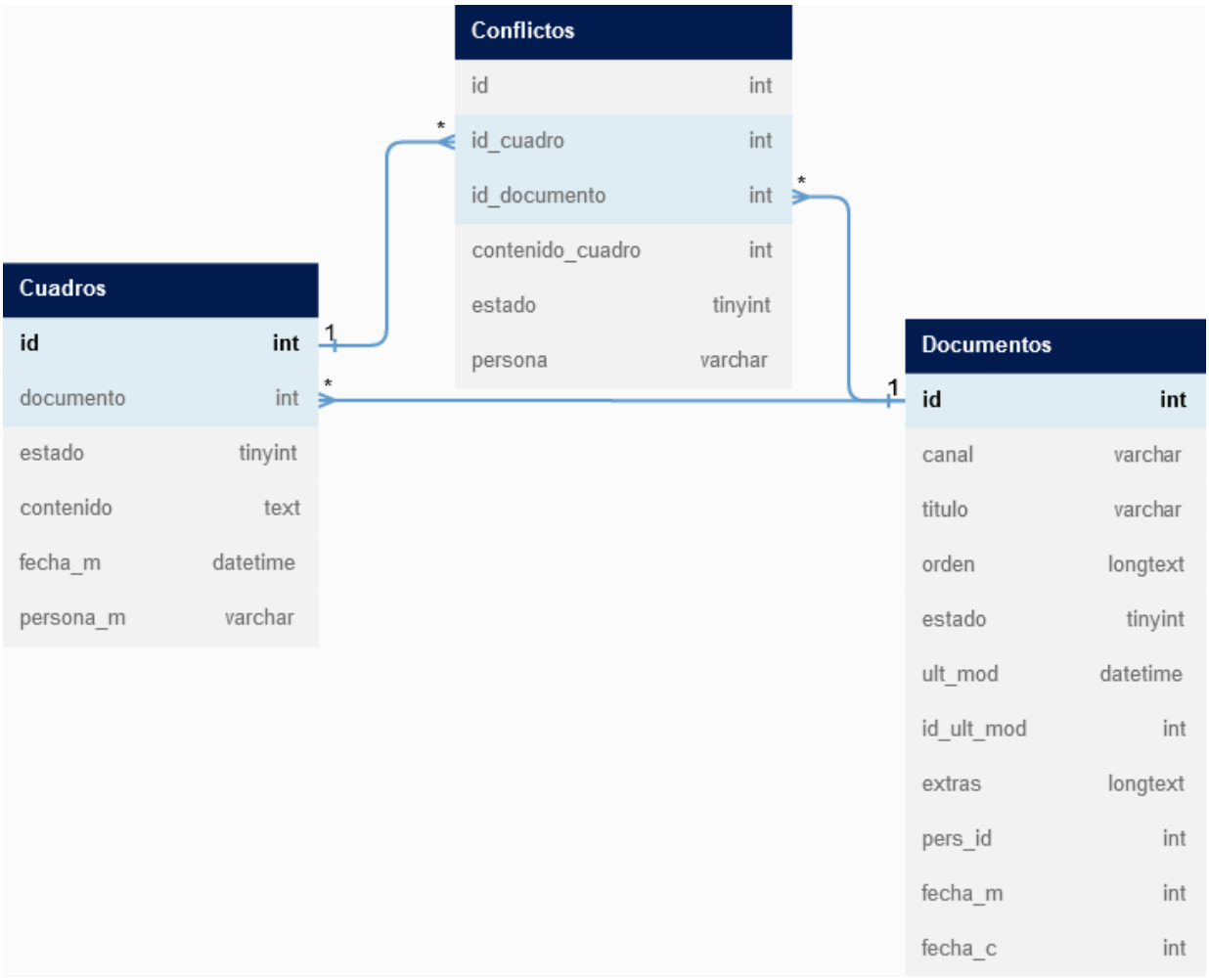

Figura 4.7: Diagrama relaciones de Base de Datos

Capítulo 5

Implementación

En este capítulo se hablará sobre la implementación de la herramienta desarrollada, pasando por las tablas implementadas en las bases de datos en la sección 5.1, como se desarrolla el documento, la forma en que se divide la información dentro de este, y las diferentes formas en las que un usuario interactúa con la herramienta en la sección 5.2.

Adicionalmente, se discutirá la implementación del sistema OCR seleccionado dentro del editor de texto en la sección 5.3, para terminar con los métodos de resolución de conflictos incorporados al módulo en la sección 5.4.

5.1. Esquemas Bases de datos

Primero, como se explicó en el capítulo de diseño 4, para el funcionamiento de esta memoria fue necesaria la implementación de 3 tablas en el sistema de Ucampus, que corresponde a MariaDB, cuyas columnas ya fueron descritas en la sección 4.2. En este sistema se creó la base de datos `UC_APUNTES` y se implementaron las tablas con las estructuras presentes en las figuras 5.1, 5.2, y 5.3.

Ahora, con la implementación del modelo de la base de datos explicado, se hará un recorrido por las distintas funcionalidades del sistema, explicando la implementación de estas.

5.2. Interacción con documentos

Primero, al ingresar al sistema, el usuario podrá ver una lista con los documentos creados en el sistema, junto con un botón para permitir la creación de nuevos documentos, como se puede observar en la figura 5.4.

Field	Type
DOC_ID	int(10)
DOC_CANAL	varchar(64)
DOC_TITULO	varchar(255)
DOC_ORDEN	longtext
DOC_ESTADO	tinyint(1)
DOC_ULT_MOD	datetime
DOC_ID_ULT_MOD	int(10)
DOC_EXTRAS	longtext
DOC_PERS_ID	int(11)
DOC_FECHA_M	int(11)
DOC_FECHA_C	int(11)

Figura 5.1: Estructura tabla `DOCUMENTOS`

Field	Type
CUA_ID	int(10)
CUA_DOCUMENTO	int(10)
CUA_ESTADO	tinyint(1)
CUA_CONTENIDO	text
CUA_FECHA_M	datetime
CUA_PERSONA_M	varchar(64)

Figura 5.2: Estructura tabla `CUADROS`

Field	Type
CONF_ID	int(10)
CONF_ID_CUADRO	int(10)
CONF_ID_DOCUMENTO	int(10)
CONF_CONTENIDO_CUADRO	text
CONF_ESTADO	tinyint(1)
CONF_PERSONA	varchar(64)

Figura 5.3: Estructura tabla `CONFLICTOS`

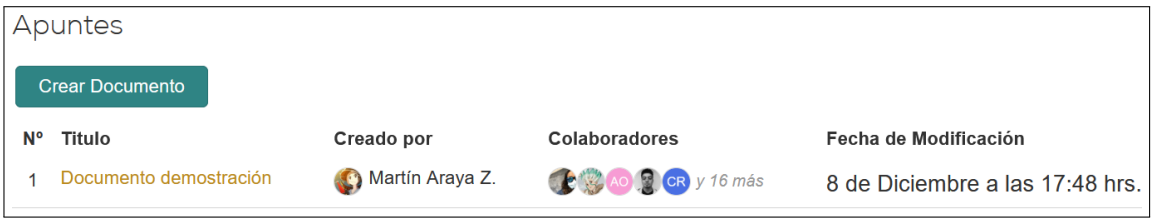

Figura 5.4: Lista de documentos dentro de un curso

Como se puede observar en la figura 5.4, se tiene que por cada documento, una fila con el número de documento, el título de este, una lista de los editores del documento y la fecha de la última modificación de este.

Para la extracción de los documentos, se utiliza el código dentro del archivo `index.php`, de la carpeta web, el cual se detalla en el código 5.1.

```
1 <?php
2 require( 'config.php' );
3
4 $documentos = Documento::get();
5
6 KERNEL::head();
7 require( template() );
8 KERNEL::foot();
```

Código 5.1: Contenido de archivo `index.php`

Este código, si bien corto, representa el funcionamiento general de un archivo de la carpeta web. Se puede apreciar que en la línea 2 se hace un requerimiento al archivo `config.php`, que contiene configuraciones compartidas por todos los archivos del módulo, correspondiente a permisos y acceso a archivos de la carpeta `include`.

Luego, en la línea 4, podemos ver que se define la variable documentos a partir del método `get` de la clase Documento, la cual está definida en el archivo `Documento.class.php` dentro de la carpeta `include`, detallado en el código 5.2.

Finalmente, se tienen las líneas 6, 7 y 8. La línea 6 realiza la renderización del *header* de la página, la línea 7 realiza la renderización del template correspondiente a este archivo de la carpeta `web`, en este caso, `index.html`, por último, la línea 8 renderiza el *footer* de la página.

Un fragmento del archivo `Documento.class.php` se detalla en el código 5.2.

```
1  <?php
2
3  class Documento extends Objeto {
4
5    static $BORRADO = 2;
6
7    static function get( $id = [], $canal = '' ) {
8      $ids = UTIL::toArray( $id );
9      if( ! $canal ) $canal = $_SESSION['modulo']['canal'];
10
11     $link = KERNEL::db();
12
13     $canal = $link->escapeSimple( $canal );
14
15     $sql = "
```

```
16 select DOC_ID, DOC_CANAL, DOC_TITULO, DOC_ORDEN, DOC_ESTADO, DOC_ULT_MOD, DOC_ID_ULT_MOD,
       DOC_EXTRAS, DOC_PERS_ID, DOC_FECHA_M, DOC_FECHA_C
17 from DOCUMENTOS
18 where DOC_CANAL = '$canal'
19 and DOC_ESTADO != 2
20 ";
21     $res = $link->query( $sql );
22     if( DB::isError( $res ) ) KERNEL::error( ERR_SQL, $sql );
23
24     $cols = [];
25     $personas = [];
26     $documentos = [];
27     while( $row = $res->fetchRow() ) {
28       $d = $documentos[ $row['DOC_ID'] ] = static::norm( $row );
29       $personas[ $d['id'] ] = 1;
30       if( $d['extras']['editores'] ) $cols += $d['extras']['editores'];
31     }
32     if( ! $documentos ) return [];
33
34     $user2rut = (array)UCURSOS::srchRut( array_keys( $cols ) );
35     $personas = UCURSOS::getDatosPersona( array_keys( $personas ) + $user2rut );
36     foreach( $documentos as $k => $d ) {
37       $documentos[$k]['creado_por'] = $personas[ $d['pers_id'] ];
38       $documentos[$k]['colaboradores'] = [];
39       foreach( array_keys( (array)$d['extras']['editores'] ) as $username ) {
40         $documentos[$k]['colaboradores'][ $user2rut[$username] ] = $personas[ $user2rut[
       $username] ];
41       }
42     }
43
44     UTIL::sort( $documentos, '_t_modificado' );
45
46     return is_array( $id ) ? $documentos : (array)$documentos[$id];
47   }
48
49   static function norm( $d ) {
50     $d['id'] = $d['DOC_ID'];
51     $d['canal'] = $d['DOC_CANAL'];
52     $d['titulo'] = $d['DOC_TITULO'];
53     $d['orden'] = (array)UTIL::json_decode( $d['DOC_ORDEN'] );
54     $d['estado'] = $d['DOC_ESTADO'];
55     $d['t_modificado'] = $d['DOC_ULT_MOD'];
56     $d['time_modificado'] = strtotime( $d['DOC_ULT_MOD'] );
57     $d['id_modificado'] = $d['DOC_ID_ULT_MOD'];
58     $d['extras'] = (array)UTIL::json_decode( $d['DOC_EXTRAS'] );
59     $d['pers_id'] = $d['DOC_PERS_ID'];
60
61     return $d;
62   }
```

Código 5.2: Contenido del archivo `Documento.class.php`

En este código, primero se tiene la definición del método get utilizado en el código 5.1. Dentro del método, primero se realiza la consulta *SQL* correspondiente a los documentos no borrados dentro del curso, entre las líneas 7 a 20.

Luego, se realiza una normalización de los datos entre las líneas 21 y 44, haciendo uso de la función estática `norm` presente entre las líneas 49 y 62.

La función `norm` cumple el propósito de obtener índices más fáciles de utilizar dentro del resto de archivos del módulo.

En particular, dentro de esta normalización, se agrega el arreglo `editores` para tener los datos de los usuarios que pueden editar cada uno de los documentos.

Finalmente, se ordenan los documentos recogidos según el último tiempo de modificación antes de retornarlos.

5.2.1. Creación y Edición de documentos

Ahora, si se utiliza el botón con la leyenda *Crear Documento* de la figura 5.4 para crear un nuevo documento, el usuario podrá ver la interfaz de la figura 5.5.

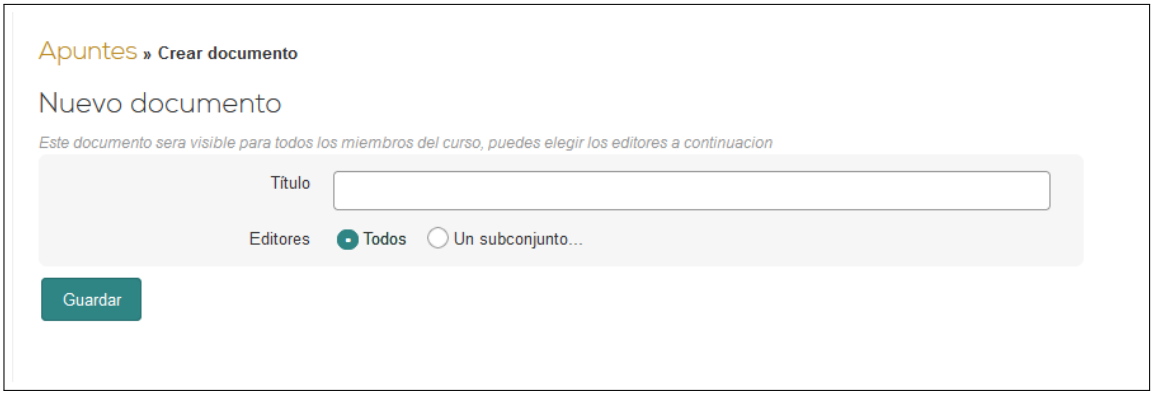

Figura 5.5: Interfaz creación documento

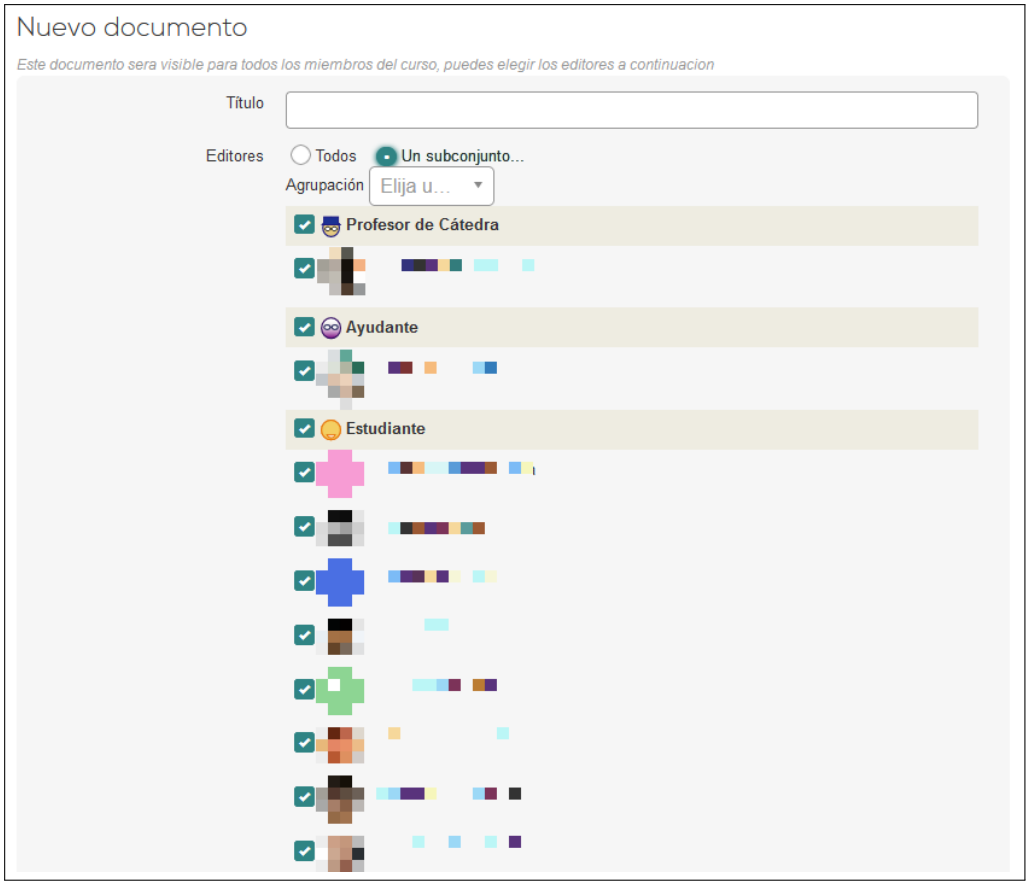

Figura 5.6: Interfaz selección de editores de documento a crear

En la figura 5.5 se puede apreciar primero un aviso sobre la visibilidad de los documentos creados. Luego, se puede elegir el título del documento a crear y los editores de este, los cuales por defecto corresponden a todos los integrantes del curso.

Sin embargo, si solamente se quiere seleccionar a un subconjunto de los integrantes, se mostrará el contenido de la figura 5.6, donde, por una consideración de privacidad, se han censurado los avatares y nombres de los integrantes del curso, en esta figura se puede observar que cuando el usuario hace clic para elegir un subconjunto de editores, se mostrara una lista con los integrantes del curso actual.

En esta lista el usuario podrá marcar todos los usuarios que decida que sean editores del documento a crear. Finalmente, con el botón de guardar se creará el documento con los parámetros elegidos.

Una vez creado el documento, el usuario verá la interfaz de la figura 5.7, donde se tiene un cuadro notificando la creación exitosa del documento con los parámetros definidos.

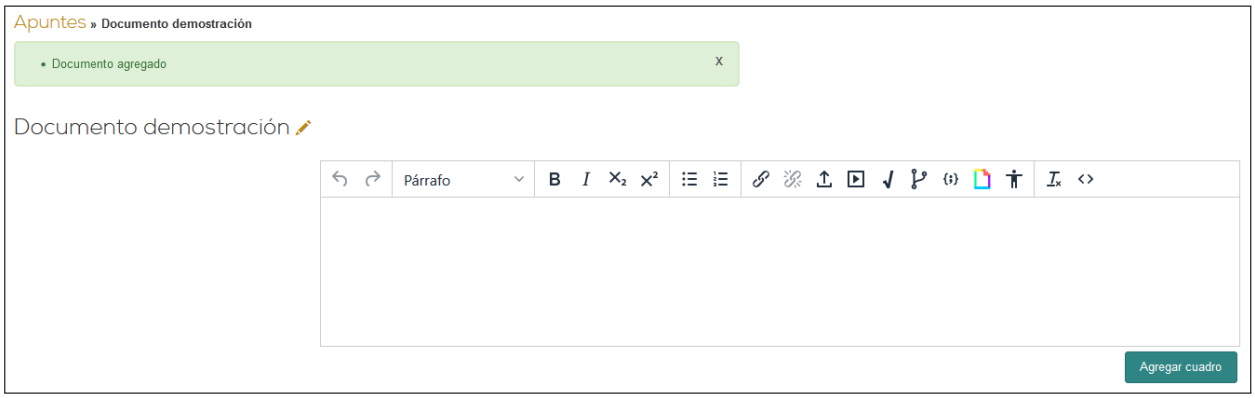

Figura 5.7: Interfaz confirmación creación de documento

Adicionalmente, se puede observar en la figura 5.7 un icono de un lápiz al costado del título del documento, mediante un clic a este, se podrá hacer edicion del título y de los editores del documento actual, mediante la interfaz visible en la figura 5.8.

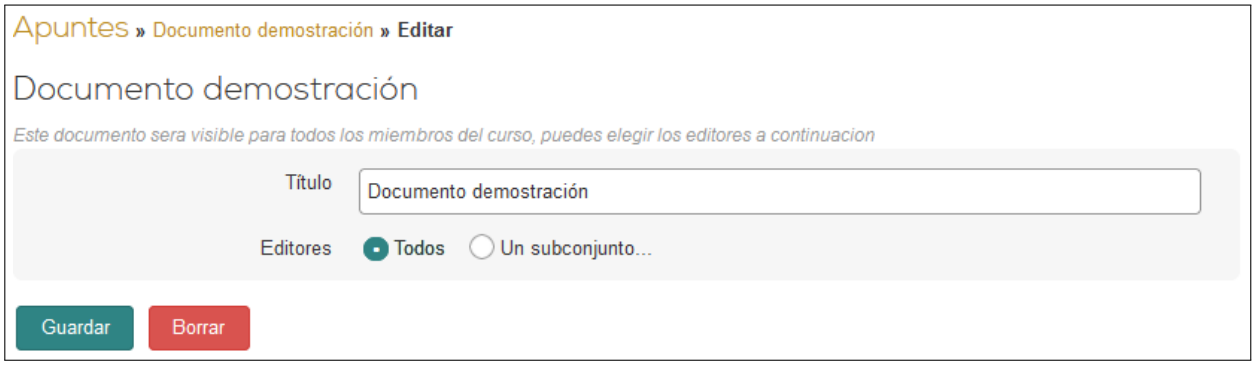

Figura 5.8: Interfaz edicion de documento

En la figura 5.8, se puede observar una interfaz muy similar a la figura 5.5, con la diferencia

de que la información del documento a editar ya está presente y se da la opción adicional de borrar el documento.

A continuación, se mostrarán los códigos relevantes a las interfaces mostradas, donde, debido a la simplicidad de este, el código de la carpeta `template` se omitirá en este caso.

En el código 5.3 se puede ver el contenido del archivo `crear.php` de la carpeta `web`, el cual se encarga de definir los parámetros a mostrar para la página de creación de documento, y también las acciones a realizar cuando el usuario hace un *submit* del formulario de creación.

```
1  <?php
2  require( 'config.php' );
3
4  $puede_ver_alumnos = TRUE;
5  $integrantes = UCURSOS::getIntegrantesCursoweb( $canal );
6  $integrantes = UTIL::srch( $integrantes, 'perfil:!oyente' );
7  $integrantes = UTIL::reindex( $integrantes, 'username' );
8  $id = (int)$_REQUEST['id'];
9
10 $d = Documento::get( $id );
11 $todos = $d['colaboradores'] ? count( $integrantes ) <= count( (array)$d['colaboradores'] )
       : TRUE;
12
13 if( $_REQUEST['accion'] == "crear_documento"){
14   $integrantes_uno = array_combine( array_keys( $integrantes ), array_fill( 0, count(
       $integrantes ), 1 ) );
15   $nuevo_documento = [
16     'id' => $d['id'] ? $d['id'] : $_REQUEST['id'],
17     'titulo' => $_REQUEST['titulo'],
18     'orden' => $d ? $d['orden'] : array(),
19     'extras' => $_REQUEST['editable_por'] ? $_REQUEST['c'] : [ 'editores' =>
       $integrantes_uno ],
20   ];
21   $id = Documento::upd( $nuevo_documento );
22   KERNEL::redirect('documento?id='.$id, $_REQUEST['id'] ? '+Documento editado': '+Documento
       agregado');
23 }
24
25 KERNEL::head( $d['id'] ? [ 'documento?id='.$d['id'] => $d['titulo'], 'Editar'] : 'Crear
       documento');
26 require( template() );
27 KERNEL::foot();
```

Código 5.3: Contenido del archivo `crear.php` para crear documentos

.

Primero, en las líneas 4 a 7, se realiza la creación de todos los parámetros importantes para mostrar el formulario, además, como esta interfaz también puede ser utilizada para la edición de un documento existente, se comprueba la existencia de este documento mediante las líneas 8 a 11.

Cuando solamente se realiza la carga de esta página, el parámetro `$_REQUEST['accion']` estará vacío, por ende se omitirá por el momento.

Finalmente, con los parámetros utilizados, se realiza la renderización de la página de la misma forma que en el código 5.1, con la edición de que, cuando se le pasa un parámetro a la función `KERNEL::head()`, se le agrega un título a la página.

Cuando el usuario presiona el botón con la leyenda *Guardar* se modificara el parámetro `$_REQUEST['accion']` para contener el valor `"Crear documento"`, en ese caso, se obtiene

la lista de editores a partir del formulario, el cual tiene que ser formateado de una forma particular para su uso posterior, como se puede ver en la línea 14.

Luego, en la línea 15 a 20 se crea el arreglo `$nuevo_documento`, conteniendo el `id` del documento en caso de estar editando, y los parámetros entregados por el formulario.

En la línea 21 se hace la operación `Documento::upd`, que corresponde a un `upsert`, función que está definida en el código 5.4.

Finalmente, se realiza la redirección del usuario a la URL correspondiente al documento creado mediante la función `KERNEL::redirect()`, la cual recibe dos parámetros, el primero corresponde a la URL a redireccionar, que toma como base la URL correspondiente al módulo. Mientras que el segundo parámetro es opcional y permite agregar un mensaje a la redirección, ocupando un símbolo "+" para mensajes positivos.

```
1  static function upd( $d ) {
2    $link = KERNEL::db();
3
4    $id = (int)$d['id'] ? (int)$d['id'] : 'null';
5    $canal = $_SESSION['modulo']['canal'];
6    $titulo = $link->escapeSimple( $d['titulo'] );
7    $orden = UTIL::json_encode( $d['orden'] );
8    $estado = (int)$d['estado'];
9    $id_modificado = (int)$d['id_modificado'];
10   $extras = UTIL::json_encode( $d['extras'] );
11   $pers_id = $d['pers_id'] ?? $_SESSION['modulo']['pers_id'];
12   $now = time();
13
14   $sql = "
15 insert into DOCUMENTOS ( DOC_ID, DOC_CANAL, DOC_TITULO, DOC_ORDEN, DOC_ESTADO, DOC_ULT_MOD,
       DOC_ID_ULT_MOD, DOC_EXTRAS, DOC_PERS_ID, DOC_FECHA_M, DOC_FECHA_C )
16 values ( $id, '$canal', '$titulo', '$orden', $estado, now(), $id_modificado, '$extras',
       $pers_id, $now, $now )
17 on duplicate key update
18 DOC_TITULO = values( DOC_TITULO ),
19 DOC_ORDEN = values( DOC_ORDEN ),
20 DOC_ULT_MOD = values( DOC_ULT_MOD ),
21 DOC_ID_ULT_MOD = values( DOC_ID_ULT_MOD ),
22 DOC_ESTADO = values( DOC_ESTADO ),
23 DOC_EXTRAS = values( DOC_EXTRAS ),
24 DOC_PERS_ID = values( DOC_PERS_ID ),
25 DOC_FECHA_M = values( DOC_FECHA_M )
26 ";
27   if( DB::isError( $link->query( $sql ) ) ) KERNEL::error( ERR_SQL, $sql );
28   if( ! $d['id'] ) $id = mysqli_insert_id( $link->connection );
29
30   return $id;
31 }
```

Código 5.4: Función `upd` del archivo `Documento.class.php`

Como se puede ver en el código 5.4, la función, luego de obtener los parámetros del documento a modificar en las líneas 4 a 12, define una operación `SQL`, en las líneas 15 a 26.

En esta operación, primero se realiza un *insert* a la tabla `DOCUMENTOS`, pero, en caso de que la llave primaria ya exista, se realiza la actualización de los valores de esta. Esta función, junto con la función `get` y `norm` definidas en el código 5.2, conforman la forma estándar de interacción con la base de datos, forma que se replica para las otras tablas definidas dentro del módulo.

5.2.2. Contenido de un documento

Ahora, una vez que se tiene un documento creado, el usuario podrá empezar a interactuar con cuadros en este. Para fines de la explicación de esta memoria, se empezará con un documento que ya contiene algunos cuadros creados, como se puede observar en la figura 5.9.

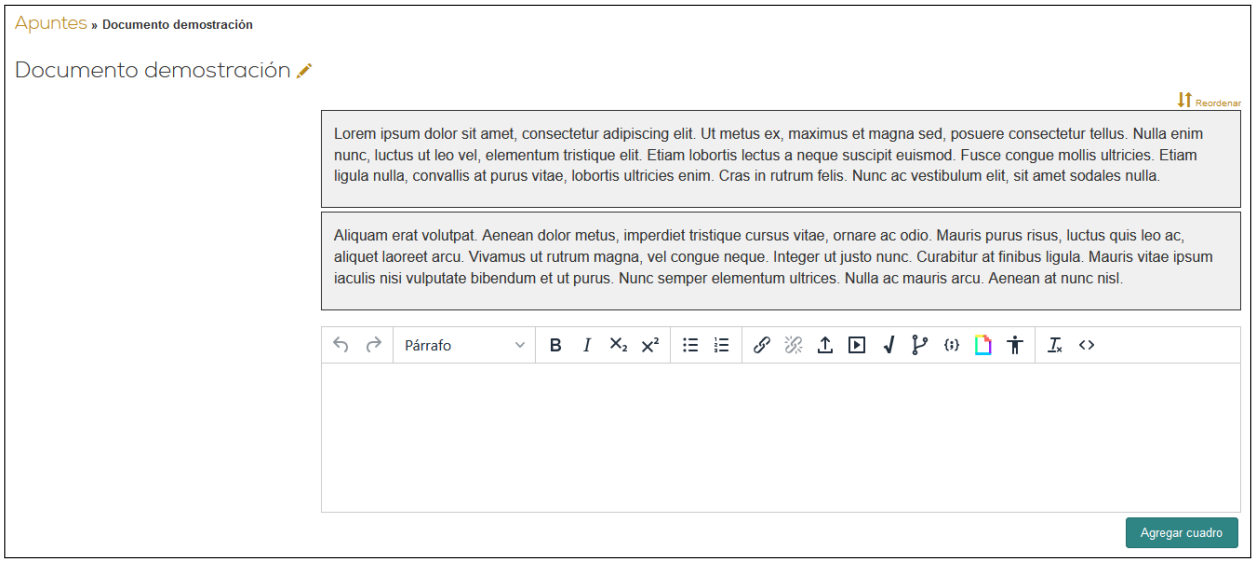

Figura 5.9: Contenido de un documento

Al realizar clic en un documento de la lista, el módulo mostrará los detalles del documento, el archivo encargado de esto corresponde a `documento.php` de la carpeta `web`, utilizando el template `documento.html`, estos archivos se explican en detalle en los fragmentos de código 5.5 y 5.6.

```
1  <?php
2  require( 'config.php' );
3
4  $id = (int)$_REQUEST['id'];
5
6  $d = Documento::get( $id );
7  if( ! $d ) KERNEL::error( ERR_NO_EXISTE );
19 $persona = UCURSOS::getDatosPersona( $pers_id );
20 $puede_editar = in_array( $persona['username'], array_keys( (array)$d['extras']['editores']
       ) );
21
22 $cuadros = Cuadro::get( $d['id'] );
23 $time = time();
79 KERNEL::head( $d['titulo'] );
80 require( template() );
81 KERNEL::foot();
```

Código 5.5: Definicion de variables iniciales y renderización de página en archivo `documento.php`

```
1  <div style="display: flex; justify-content: space-between">
2    <h1>{{$d['titulo']}} <a href="crear?id={$d['id']}"><span class="glyphicon glyphicon-pencil
       "></span></a></h1>
3  </div>
4
5  <div id="container">
10 <!-- foreach( $d['orden'] as $i => $o ) {-->
11   <div class="cuadro" style="display: flex;" id="{$o}">
12     <div id="viewers_{$o}" style="display: flex; flex-grow: 0; width: 20%; justify-content:
       flex-end;"></div>
13     <div class="content" id="{$o}" >
14       <div style="width: 100%;">
15         <div id="contenido_{$cuadros[$o]['id']}" class="cuadro_texto">
16           {KERNEL::formatear($cuadros[$o]['contenido'], TXT_ED_WYSIWYG | TXT_TODO | TXT_OCR)
     }
17         </div>
```

Código 5.6: Definicion de título de página y cuadros de documento en archivo `documento.html`

Primero, en el fragmento de código 5.5, se puede observar que se verifica la existencia del documento a consultar en las líneas 4 a 7. Luego, se obtienen los datos de la persona actualmente visitando la página para definir el parámetro `$puede_editar`, que definirá parte del funcionamiento del template.

Posteriormente, se obtiene la lista de cuadros asociados al documento, mediante la función `Cuadro::get()`, que recibe como parámetro el `id` del documento que se está revisando, para obtener los cuadros asociados a este. El funcionamiento de esta función es equivalente a la función mostrada en el código 5.2.

Finalmente, se hace la renderización del template correspondiente con el título del documento en el `header` en las líneas 79 a 81.

En el caso del template `documento.html`, en el fragmento de código 5.6, se muestra primero el título del documento y el icono de lápiz utilizado para la edicion, en las líneas 1 a 3. Este lápiz se puede ver en la figura 5.7.

Luego, en las líneas 5 a 17, se define el *div* que contendrá a los cuadros presentes en el documento, el cual, utilizando un fragmento de código *PHP*, hace un ciclo *for* por cada uno de los cuadros detectados en el orden recibidos y hace la renderización del contenido de un cuadro mediante la función `KERNEL::formatear`.

Esta función permite formatear el contenido de un cuadro, que se guarda como texto plano, mediante el editor de texto *tinymce*.

El editor de texto *tinymce* corresponde a una librería utilizada dentro de la plataforma *U-Cursos* como editor de texto en diferentes módulos. Este editor tiene la ventaja de que es altamente personalizable, permitiendo editar las opciones disponibles o incluso expandir estas mediante plugins.

El primer argumento consiste en el texto a formatear, y el segundo corresponde a los add-ons a utilizar, donde se ha añadido el add-on TXT_OCR cuya función será explicada en la sección 5.3.

5.2.3. Creación de cuadros

Una vez realizada la explicación del contenido de un documento en general, la primera funcionalidad realizada corresponde a la capacidad de un usuario de crear cuadros, para esto, el usuario ocupará el cuadro de texto presente en la parte inferior del documento.

Luego, ocupando el botón con la leyenda *Agregar Cuadro* el cuadro se agregara al documento. Como confirmación, se muestra un cuadro de color verde con el texto *Cuadro agregado* como se puede observar en la figura 5.10.

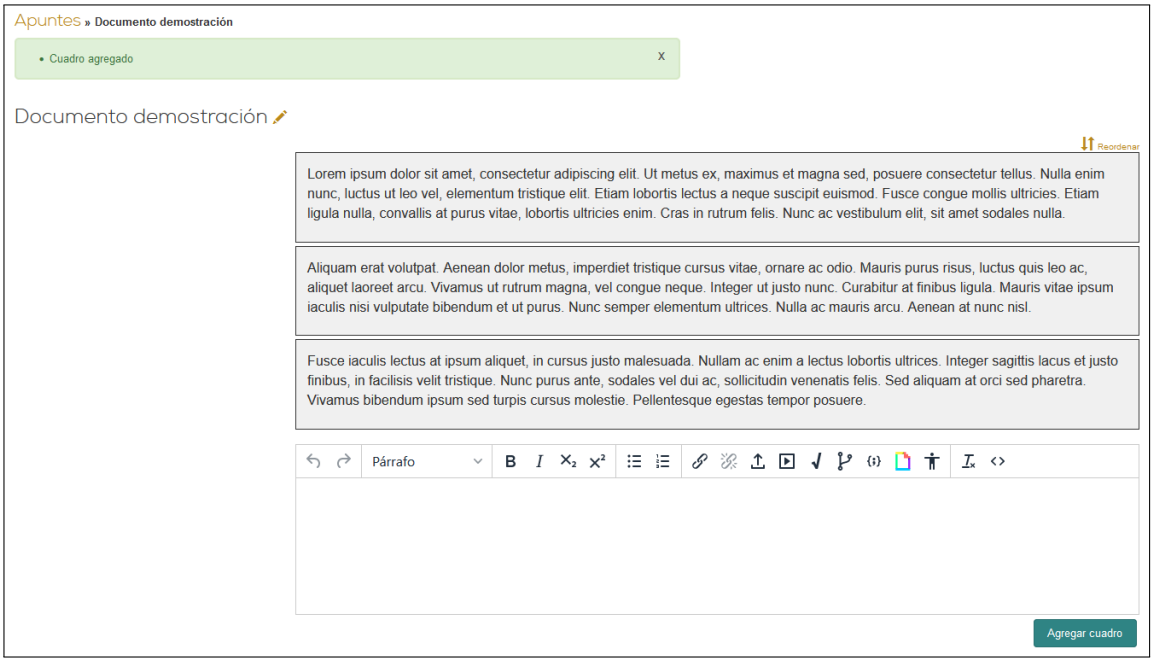

Figura 5.10: Notificación luego de agregar cuadro a documento

El cuadro para agregar nuevo texto al documento se define en el código 5.7.

```
<!-- if( $puede_editar ) {-->
<div style="display: flex; justify-content: center; margin-top: 1em;">
    <div id="editar_nuevo" style="width: 60%;">
      <form method="POST">
        <input type="hidden" name="accion" value="agregar_cuadro" />
        <input type="hidden" name="tiempo_carga" value="{$time}" />
        {KERNEL::textarea("cuadro", "", TXT_ED_WYSIWYG | TXT_TODO | TXT_OCR)}
        <div style="display:flex; justify-content: end; margin-top: 5px;">
          <button type="submit">Agregar cuadro</button>
        </div>
      </form>
    </div>
</div>
<!-- } -->
```

Código 5.7: Cuadro para ingresar texto de `documento.html`

El cuadro de texto definido en el código 5.7 solamente aparecerá cuando el usuario tiene el permiso de edicion definido en el código 5.5. El cuadro está contenido dentro de un formulario con dos *input* ocultos, que corresponden a la acción a realizar y el tiempo en que el usuario realizo la carga de la página.

Además, se realiza la carga del editor *tinymce* mediante la función `KERNEL::textarea`, cuyos parámetros corresponden al identificador dentro del formulario, el contenido inicial del editor, y los add-ons a agregar a este.

Finalmente, se tiene el botón para realizar el *submit* de este formulario.

```
1  if( $_REQUEST['accion'] == "agregar_cuadro" && $_REQUEST['cuadro'] ) {
2    $nuevos_cuadros = textoACuadros( $_REQUEST['cuadro'] );
3    foreach( $nuevos_cuadros as $nuevo_contenido ) {
4      if( ! $nuevo_contenido ) continue;
5
6      $nuevo_cuadro = [
7        'contenido' => UTIL::utf2iso( $nuevo_contenido ),
8        'doc_id' => $d['id'],
9        'pers_id' => $pers_id,
10     ];
11     $nuevo_cuadro_id = Cuadro::upd( $nuevo_cuadro );
12     $d['orden'][] = $nuevo_cuadro_id;
13     Documento::upd( $d );
14   }
15   KERNEL::redirect( 'documento?id='.$d['id'], '+Cuadro agregado' );
```

Código 5.8: Fragmento de código para agregar un cuadro de `documento.php`

Como se puede ver en el fragmento de código 5.8, primero se verifica la acción del formulario y que el cuadro tenga contenido en la línea 1. Luego, utilizando la función `textoACuadros()`, se realiza la separación de un texto muy grande a múltiples cuadros dependiendo de los saltos de línea utilizados por el usuario.

Luego, por cada cuadro detectado, se crea el arreglo con el contenido de este, el `id` del documento y el `id` de la persona que lo ha creado en las líneas 6 a 10, y, utilizando la función `Cuadro::upd()` se realiza la inserción de este nuevo cuadro.

La función `Cuadro::upd()` recibe como parámetro el arreglo correspondiente al cuadro a agregar y su funcionamiento es equivalente a la función `Documento::upd()` detallada en el código 5.4.

Una vez creado el nuevo cuadro, se actualiza el orden del documento utilizando el `id` recibido como parámetro de retorno de la función `Cuadro::upd` y se actualiza el documento con el nuevo orden.

Finalmente, en la línea 15, se realiza la redirección del usuario al documento con la notificación positiva que se ha agregado un cuadro correctamente.

5.2.4. Edición de cuadros

La siguiente funcionalidad del módulo a destacar es la capacidad de un usuario de editar los cuadros ya existentes en el documento, para esto, el usuario puede realizar un click doble a cualquiera de los cuadros existentes.

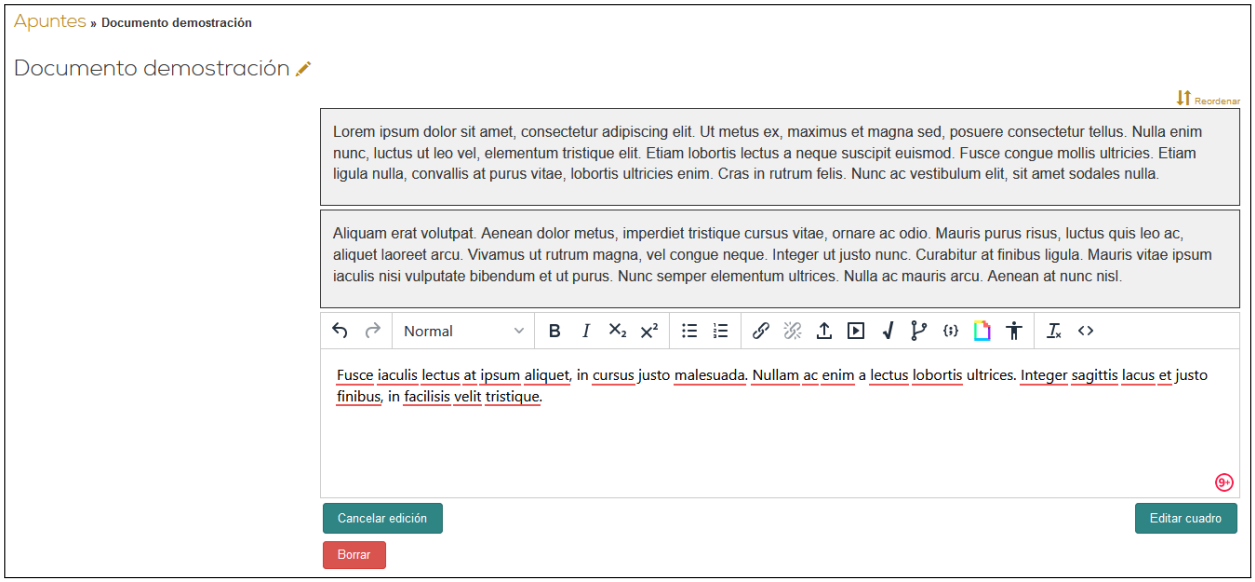

Figura 5.11: Inicio edición cuadro

Una vez realizada esta acción, el cuadro cambiará a una interfaz editable, como se muestra en la figura 5.11, donde el usuario, luego de editar el contenido del cuadro, tiene tres opciones.

1. Cancelar la edicion
2. Editar el cuadro
3. Borrar el cuadro

```
<!-- foreach( $d['orden'] as $i => $o ) {-->
  <div class="cuadro" style="display: flex;" id="{$o}">
    <div id="viewers_{$o}" style="display: flex; flex-grow: 0; width: 20%; justify-content:
    flex-end;"></div>
    <div class="content" id="{$o}" >
      <div style="width: 100%;">
        <div id="contenido_{$cuadros[$o]['id']}" class="cuadro_texto">
          {KERNEL::formatear($cuadros[$o]['contenido'], TXT_ED_WYSIWYG | TXT_TODO | TXT_OCR)
    }
        </div>
  <!-- if( $puede_editar ) {-->
          <div id="editar_{$cuadros[$o]['id']}" class="editar_texto">
            <form method="POST">
              <input type="hidden" name="accion" value="editar_cuadro" />
              <input type="hidden" name="cuadro_id" value="{{$cuadros[$o]['id']}}" />
              <input type="hidden" name="tiempo_carga" value="{$time}" />
              {KERNEL::textarea("cuadro_".$o, $cuadros[$o]['contenido'], TXT_ED_WYSIWYG |
    TXT_TODO | TXT_OCR)}
              <div style="display: flex; justify-content: space-between; margin-top: 5px;">
                <button type="button" onclick="cancelar_edicion({$cuadros[$o]['id']})" >
    Cancelar edición</button>
                <button type="submit">Editar cuadro</button>
              </div>
              <a class="boton confirm" href="?id={$d['id']}&accion=borrar&tiempo_carga={
    $time}&cuadro_id={$cuadros[$o]['id']}">Borrar</a>
            </form>
          </div>
  <!-- } -->
      </div>
```

Código 5.9: Cuadro para editar texto existente en archivo `documento.html`

Como se puede ver en el fragmento de código 5.9, cuando se realiza la renderización del contenido de cada uno de los cuadros de texto del documento, en las líneas 1 a 8. También se realiza la renderización de un formulario que contiene el editor *tinymce* con el contenido del cuadro, en las líneas 9 a 23.

En este formulario, se tienen como *inputs* ocultos la acción a realizar, el `id` del cuadro a editar y el tiempo de carga de la página, que permitirá la detección de conflictos y el editor en sí.

Finalmente, se tiene los botones para cancelar la edicion, editar el cuadro y borrar el cuadro respectivamente. El primero de estos solamente oculta el editor mediante la función `cancelar_edicion()` con el `id` del cuadro como parámetro. El segundo botón realiza el `submit` del formulario y el tercer botón es un enlace que redirige al usuario a una URL con los parámetros necesarios para realizar la eliminación del cuadro.

A continuación, se realizará la explicación detallada de las funcionalidades restantes, edicion del contenido y borrado de un cuadro.

Editar contenido de cuadro

La segunda opción, cuyo botón correspondiente está del lado derecho del cuadro y es de color verde, realiza la edicion del cuadro con el texto que el usuario ha modificado.

```
1  } elseif( $_REQUEST['accion'] == "editar_cuadro" ) {
2    $cuadro = $cuadros[ $_REQUEST['cuadro_id'] ];
3    if( $_REQUEST['cuadro_id'] && ! $cuadro ) KERNEL::mensaje( '-Error al modificar el cuadro'
       );
4
5    $cuadro['contenido'] = $_REQUEST[ 'cuadro_'.$cuadro['id'] ];
6    $cuadro['pers_id'] = $pers_id;
7
8    if( ! KERNEL::mensaje() ) {
9      $t_modificado = strtotime( $cuadro['t_modificado'] );
10     if( $_REQUEST['tiempo_carga'] < $t_modificado ){
11       $conflicto = [
12         'cuad_id' => $cuadro['id'],
13         'doc_id' => $d['id'],
14         'cuad_contenido' => $cuadro['contenido'],
15         'pers_id' => $pers_id,
16       ];
17       Conflicto::upd( $conflicto );
18       KERNEL::redirect( 'conflicto?doc_id='.$d['id'] );
19
20     } else{
21       Cuadro::upd( $cuadro );
22       Documento::upd( [ 'id_modificado' => $cuadro['id'] ] + $d );
23       KERNEL::redirect( 'documento?id='.$d['id'], '+Cuadro editado' );
24     }
25   }
```

Código 5.10: Fragmento de código para editar un cuadro en archivo `documento.php`

En el fragmento 5.10 se detalla el código del archivo `documento.php` que se encarga de la modificación del contenido de un cuadro.

Primero, en las líneas 2 y 3, se realiza la verificación que el cuadro a editar corresponda a un cuadro perteneciente al arreglo de cuadros correspondiente al documento abierto.

Luego, se obtiene el nuevo contenido del cuadro y el identificador de la persona que ha realizado la modificación. Después, si no existe un mensaje de error, se toma el tiempo de la última modificación realizada al cuadro a editar, en las líneas 5 a 10.

De esta forma, se puede detectar si existe un conflicto cuando el tiempo de carga de la página sea más antiguo que la última modificación realizada al cuadro, cuya resolución se verá en detalle en la sección 5.4.

En el caso de que no se produzca un conflicto, en las líneas 21 a 23, se utiliza la función `Cuadro::upd` para editar el contenido del cuadro, se actualiza el `id` del último cuadro editado en el documento y se hace la redirección del usuario al documento con el mensaje positivo *Cuadro editado*.

Una vez que este proceso se completa, el usuario verá el mensaje de confirmación que aparece en la figura 5.12.

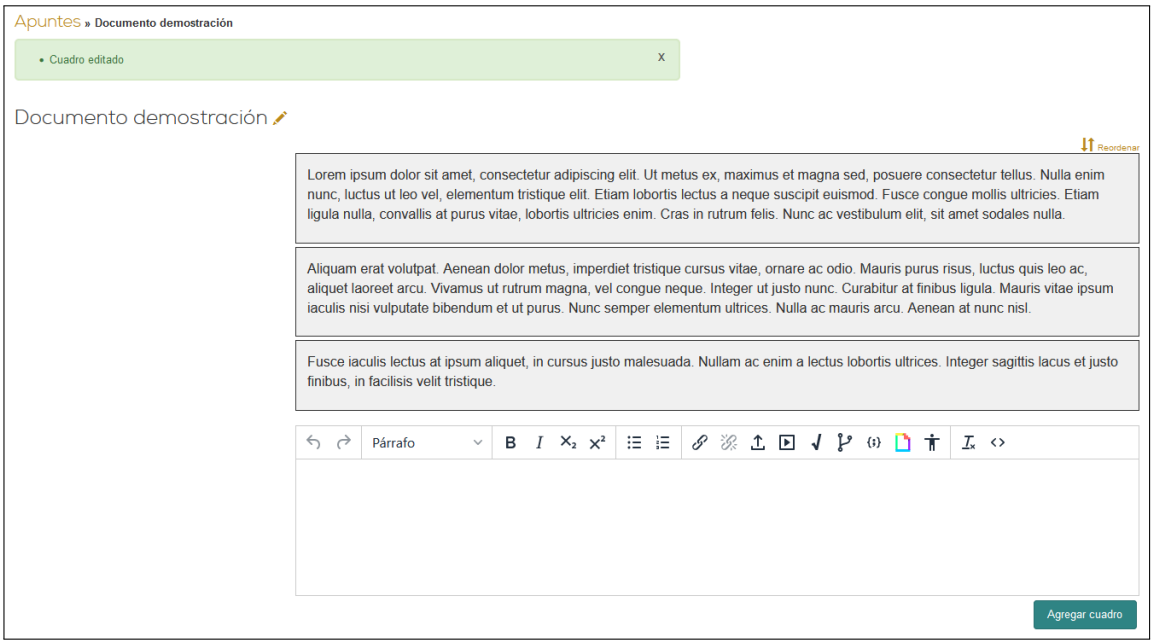

Figura 5.12: Notificación luego de edición de un cuadro

Borrado de cuadro

La última acción, borrar un cuadro, se realiza mediante el botón de color rojo que se puede apreciar en la figura 5.11, una vez presionado, se le pedirá una confirmación al usuario mediante el diálogo representado en la figura 5.13.

Luego, si el usuario decide confirmar la acción, se realizará una nueva carga del documento con los cuadros restantes y verá un cuadro de confirmación como se aprecia en la figura 5.14.

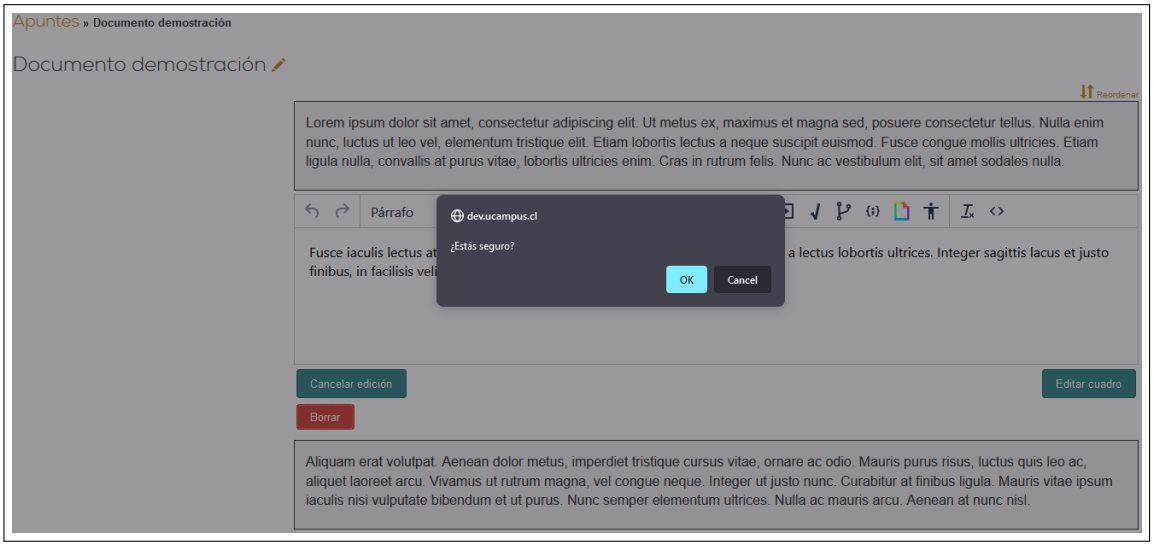

Figura 5.13: Confirmación borrado cuadro

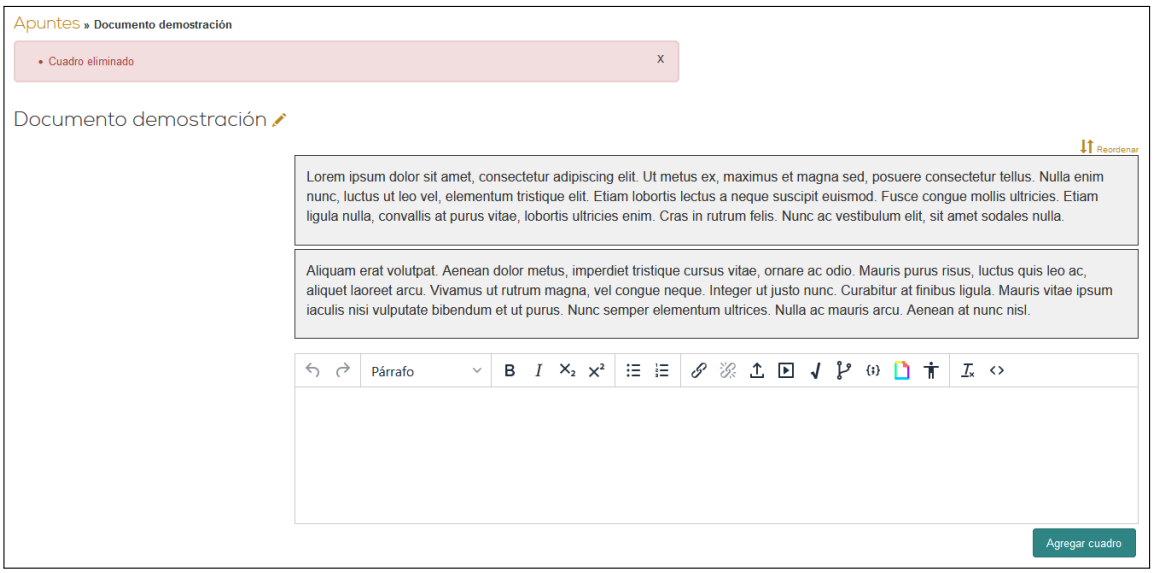

Figura 5.14: Cuadro borrado

```
1 }  elseif( $_REQUEST['accion'] == "borrar" ) {
2   $cuadro_id = $_REQUEST["cuadro_id"];
3   $d['orden'] = array_diff( $d['orden'], [ $cuadro_id ] );
4   Cuadro::upd( [ 'estado' => Cuadro::$BORRADO ] + $cuadros[$cuadro_id] );
5   Documento::upd( $d );
6   KERNEL::redirect( 'documento?id='.$d['id'], '-Cuadro eliminado' );
```

Código 5.11: Fragmento de código para borrar un cuadro en archvio `documento.php`

En el código 5.11 se puede ver las acciones que se realizan al borrar un cuadro. Primero, se obtiene el `id` del cuadro a partir de los parámetros dentro de la URL que se pueden ver en la línea 20 del código 5.9.

Luego, se elimina este *id* del arreglo `orden` del documento en la línea 3. Posteriormente,

en la línea 4 y 5 se actualiza el estado del cuadro a borrado y se actualiza el documento actual con el orden editado, respectivamente.

Finalmente, se realiza la redirección del usuario con el mensaje en rojo “Cuadro eliminado” que se puede ver en la figura 5.14.

5.2.5. Cambio de orden

Ahora, una forma diferente de interactuar con los cuadros del documento corresponde a cambiar el orden en que estos están presentes dentro del documento en sí.

Para esto, el usuario podrá ver un enlace en la parte superior derecha de la lista de cuadros, el cual, al ser presionado, mostrara un *handler* para cada cuadro del documento, como se puede apreciar en la figura 5.15.

En este ejemplo, se mueve el tercer cuadro a la segunda posición, como se muestra en la figura 5.16. El usuario tendrá libertad de mover todos los cuadros de la forma en que le parezca correspondiente, solamente cuando se vuelva a presionar el enlace de reordenar y desaparezcan los *handlers* de los cuadros es que se realizara un cambio en el servidor sobre el estado del documento.

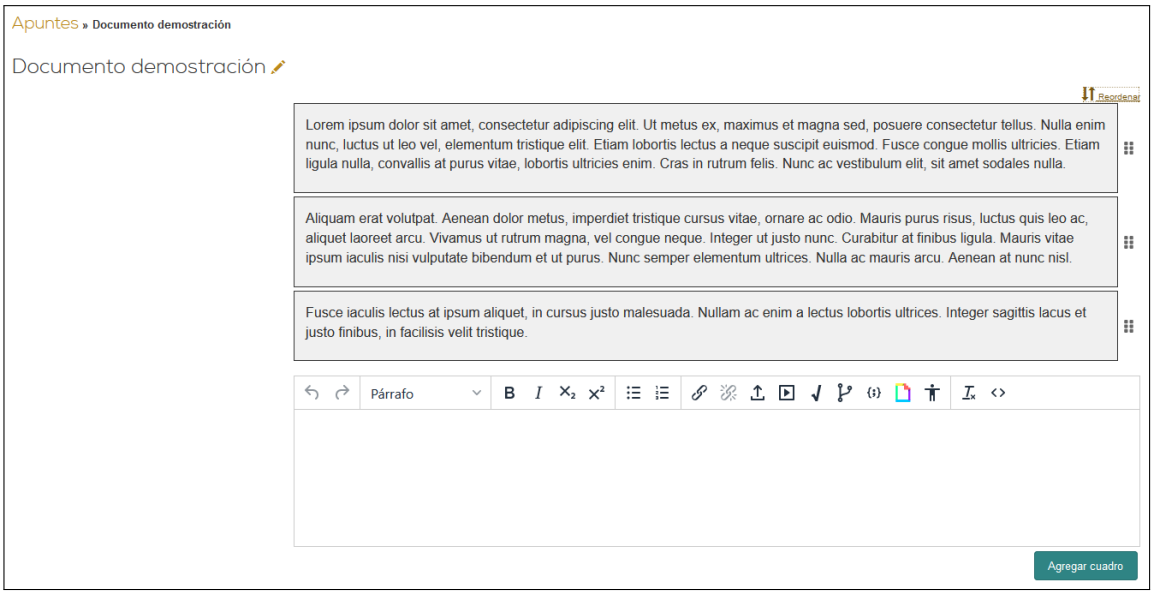

Figura 5.15: Inicio acción mover

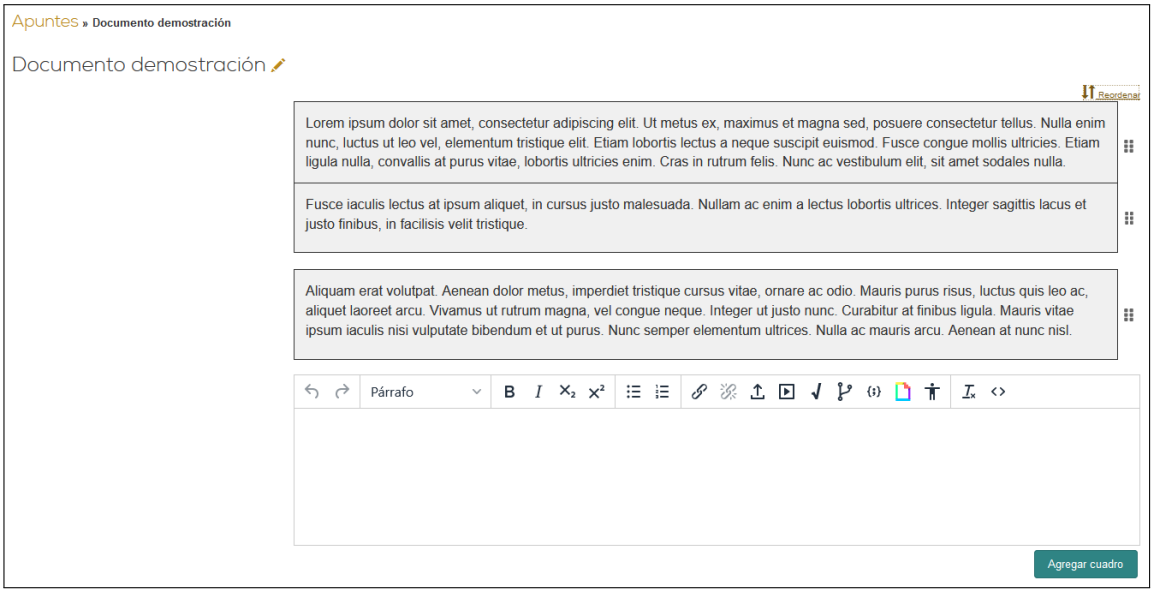

Figura 5.16: Cuadro movido

Ahora, en el código 5.12 se muestra la implementación de esta funcionalidad en el archivo `documento.html` de la carpeta `include`.

```
<!-- if( $puede_editar && count($cuadros) > 1 ) {-->
  <a href="javascript:editar_orden()" style="margin: 0 20% 0 auto;font-size: 0.8em;"><span class="glyphicon glyphicon-sort"></span> Reordenar</a>
<!-- } -->

<!-- foreach( $d['orden'] as $i => $o ) {-->
  <div class="cuadro" style="display: flex;" id="{$o}">
    <div id="viewers_{$o}" style="display: flex; flex-grow: 0; width: 20%; justify-content: flex-end;"></div>
    <div class="content" id="{$o}" >
      <div style="width: 100%;">
        <div id="contenido_{$cuadros[$o]['id']}" class="cuadro_texto">
          {KERNEL::formatear($cuadros[$o]['contenido'], TXT_ED_WYSIWYG | TXT_TODO | TXT_OCR)
    }
        </div>
  <!-- if( $puede_editar ) {-->
          <div id="editar_{$cuadros[$o]['id']}" class="editar_texto">
            <form method="POST">
              <input type="hidden" name="accion" value="editar_cuadro" />
              <input type="hidden" name="cuadro_id" value="{{$cuadros[$o]['id']}}" />
              <input type="hidden" name="tiempo_carga" value="{$time}" />
              {KERNEL::textarea("cuadro_".$o, $cuadros[$o]['contenido'], TXT_ED_WYSIWYG | TXT_TODO | TXT_OCR)}
              <div style="display: flex; justify-content: space-between; margin-top: 5px;">
                <button type="button" onclick="cancelar_edicion({$cuadros[$o]['id']})" >Cancelar edición</button>
                <button type="submit">Editar cuadro</button>
              </div>
              <a class="boton confirm" href="?id={$d['id']}&accion=borrar&tiempo_carga={$time}&cuadro_id={$cuadros[$o]['id']}">Borrar</a>
            </form>
          </div>
  <!-- } -->
    </div>
    <div class="sort-handler"></div>
  </div>
</div>
```

Código 5.12: Implementación de movimiento de un cuadro en archivo `documento.html`

En las líneas 1 a 3 del código 5.12, se puede observar que el icono de *sort* solamente aparecerá cuando el usuario tiene permiso de edicion y exista más de un cuadro.

En la línea 29, se puede ver la definición del *handler* que aparece cuando el usuario presiona el botón reordenar, este es un objeto clickeable oculto que permite el arrastre de los cuadros en la interfaz.

Para realizar esta acción de movimiento, fue necesario un archivo adicional en la carpeta `web`, llamado `documento.js`, este archivo contiene los scripts necesarios para el funcionamiento correcto de la interfaz, el contenido de este archivo se puede ver en el código 5.13.

```
23 function editar_orden() {
24   if( ! editando ) {
25     $( ".content" ).unbind( "dblclick" );
26     $( ".sort-handler" ).show();
27     $( "#editar_texto" ).hide();
28     sortable.sortable( "enable" );
29     editando = true;
30     return;
31   }
32   $( ".sort-handler" ).hide();
33   sortable.sortable( "disable" );
34   $( ".content" ).dblclick( function( params ) { editar_cuadro( this.id ); } );
35   //console.log( sortable.sortable( "toArray" ) );
36   $.ajax( {
37     url: session_modulo_url+"documento?id={$d['id']}&accion=mover",
38     data: { order: sortable.sortable( "toArray" ), }
39   } );
40   editando = false;
41 }
120 //se ejecuta luego de la carga del DOM
121 $( function () {
122   sortable = $("#container").sortable( {
123     axis: "y",
124     handle: ".sort-handler",
125     update: function( event, ui ) {
126       $( ui.item ).find( 'textarea' ).each( function () {
127         tinymce.execCommand( 'mceAddEditor', true, $( this ).attr( 'id' ) );
128       } );
129     },
130     start: function( event, ui ) {
131       $( ui.item ).find( 'textarea' ).each( function () {
132         tinymce.execCommand( 'mceRemoveEditor', false, $( this ).attr( 'id' ) );
133       } );
134     },
135   } );
136
137   sortable.sortable( "disable" );
138   if( puede_editar ){
139     $( ".content" ).dblclick( function( params ) {
140       editar_cuadro( this.id );
141       vw.edit( this.id );
142     } );
143   }
144
145   vw.init();
146 } );
```

Código 5.13: Fragmento de scripts de archivo `documento.js`

Para entender el código 5.13, primero es necesario ver desde la línea 126 en adelante, que corresponde a código que se ejecuta luego de la carga del *DOM* de la página.

En este fragmente, en las líneas 127 a 140, se define un objeto `sortable` sobre el `container` que contiene todos los cuadros. Se define el eje de movimiento y el objeto que se utiliza

para el movimiento.

Luego, se definen dos funciones, `start` para el momento de hacer clic sobre uno de los *handlers* y la función `update` para cuando este elemento se "suelta".

Estas funciones tienen como objetivo remover el contenido del editor en esa posición, y reincorporar el contenido al editor correspondiente, respectivamente. Una vez definido el objeto, este se desactiva inmediatamente, para activarse según las acciones del usuario.

Posteriormente, en las líneas 23 a 41 se tiene la función `editar_orden()` que se ejecuta cada vez que se presiona el botón reordenar que se puede ver en la figura 5.15.

Esta función, si la variable *editando* contiene el valor `false` deshabilita el doble clic sobre los cuadros, muestra los *handlers* de los cuadros y activa el objeto `sortable` definido anteriormente.

En caso contrario, si la variable *editando* contiene el valor `true`, se ocultan los *handlers*, se desactiva el objeto `sortable` y se reactiva la funcionalidad de doble clic para editar los cuadros. Luego, se realiza el envío de la información del nuevo orden de los cuadros a la URL del documento con la acción "mover" como parámetro.

En el archivo `documento.php` se tiene el fragmento de código para procesar el cambio de orden.

```
1  <?php
2  require( 'config.php' );
3
4  $id = (int)$_REQUEST['id'];
5
6  $d = Documento::get( $id );
7  if( ! $d ) KERNEL::error( ERR_NO_EXISTE );
8
9  if( $_REQUEST['accion'] == "timestamp" ) {
10   KERNEL::streamAPI( $d['time_modificado'] );
11
12 } elseif( $_REQUEST['accion'] == "mover" ) {
13   Documento::upd( [ 'orden' => $_REQUEST['order'] ] + $d );
14   KERNEL::streamAPI( "200" );
15 };
```

Código 5.14: Implementación para cambiar el orden de cuadros en archivo `documento.php`

En el código 5.14, primero, se hace la verificación de la existencia del documento, luego, se realiza, en las líneas 12 a 15 se actualiza el orden del documento actual con el nuevo orden, y, ocupando la función `KERNEL::streamAPI()` se envía el *string* "200" como confirmación.

Debido a que este cambio del documento fue realizado como una operación ajax, como se puede ver en la línea 36 del código 5.13, la actualización del orden no implica una recarga de la página, por lo que el usuario podrá seguir trabajando en el documento instantáneamente.

5.3. Implementación OCR

Finalmente, ya que se ha realizado la explicación de la implementación necesaria para la interacción del usuario, se procederá a la explicación del sistema OCR utilizado en el sistema.

Este sistema, como se detalló en el capítulo 4 corresponde a la plataforma *Azure Computer Vision* ofrecido por la empresa *Microsoft*.

5.3.1. Implementación de Interfaz

Para el uso de esta se creó un nuevo botón al editor de texto *tinymce*, el cual abrirá un cuadro para que el usuario elija la imagen a procesar, como se puede ver en la figura 5.17.

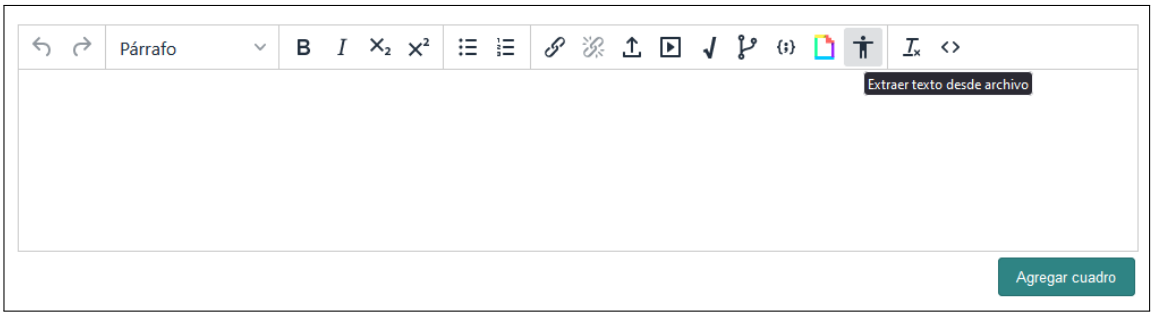

Figura 5.17: Botón para extraer texto en editor

Luego de seleccionar la figura en su dispositivo, el usuario tendrá que esperar unos segundos para el procesamiento del texto, como se puede ver en la figura 5.18.

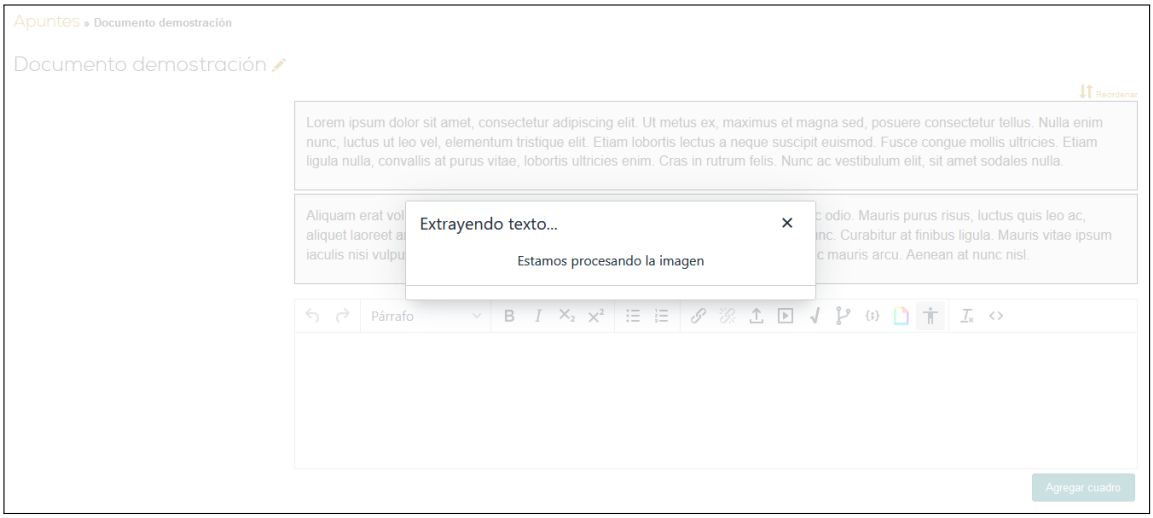

Figura 5.18: Cuadro de espera extracción de texto

Para la implementación del botón, fue necesario el añadir un botón al editor de texto *tinymce* implementado en la plataforma, para eso, se siguieron las instrucciones del editor [8].

Para esto, fue necesario la creación de la carpeta `ucampus_ocr` dentro de la carpeta `template` del módulo. La cual contiene el archivo `plugin.js` encargado de la creación del botón y las acciones de este.

```
1    editor.ui.registry.addButton( 'ucampus_ocr', {
2      icon: 'accessibility-check',
3      tooltip: 'Extraer texto desde archivo',
4      active: true,
5      onAction: function( api ) {
6        if( ! inp ) {
7          inp = document.createElement( 'input' );
8          inp.type = 'file';
9          inp.style = 'position:fixed;top:0;left:0;opacity:0.001';
10             document.body.appendChild( inp );
11         inp.addEventListener( 'change', function( e ) {
12           ajaxUpload( e.target );
13           inp.value = '';
14         } );
15       }
16       inp.click();
17     },
18   } );
19 } );
```

Código 5.15: Definición de boton de `input.js`

En el código 5.15 se hace el registro del botón y la acción que este realiza, que corresponde a crear un elemento `input` dentro del documento y realizar clic a este, para que el usuario pueda elegir el archivo de su computador a cargar, y que será procesado por la función `ajaxUpload`.

Luego, en el código 5.16 y se muestra el envío de la información de la imagen y el código encargado de mostrar el panel de espera al usuario.

```
1    var ajaxUpload = function( e ) {
2      if( ! e || ! e.files || ! e.files.length ) return;
3
4      var ajaxData = new FormData();
5      ajaxData.append( 'file', e.files[0] );
6      var xhr = new XMLHttpRequest();
7      xhr.addEventListener( 'load', function () {
8        var response = JSON.parse( xhr.responseText );
9        console.log(response);
10       if ( response['error'] != 0 ){
11         editor.notificationManager.open( {
12                     text: response['error'],
13                     type: 'error',
14                     timeout: 3000
15                   } );
16       }
17       else {
18         editor.insertContent( JSON.parse(xhr.responseText)['texto'] );
19       }
20       xhr.dialog.close();
21     } );
```

Código 5.16: Definición de `ajaxUpload` e `EventListener` de `input.js`

En las líneas 7 a 21 del código 5.16, se realiza la creación del `EventListener` que se encargara de colocar el texto dentro del editor (línea 18) o mostrar una notificación en caso de no recibir un texto de respuesta (líneas 11 a 15).

Ahora, una vez definido el `EventListener` se muestra el cuadro al usuario que aparece

en la figura 5.18, el cual está definido en el código 5.17.

```
1       xhr.dialog = editor.windowManager.open( {
2         icon: 'upload',
3         title: 'Extrayendo texto...',
4         body: {
5           type: 'panel', // The root body type - a Panel or TabPanel
6           items: [ {
7             type: 'htmlpanel', // A HTML panel component
8             html: '<div id="upload-dialog" style="text-align:center">Estamos procesando la
    imagen<span class="espera"/></div>'
9           } ]
10        },
11        buttons: [],
12        onCancel: function() {
13          xhr.abort();
14        }
15      } );
16
17      xhr.open( 'POST', 'extraer_texto' );
18      xhr.send( ajaxData );
19    };
```

Código 5.17: Definición de cuadro y POST de información en `input.js`

Luego de mostrar el cuadro, en las líneas 17 y 18 se realiza el `POST` de la información de la imagen desde el usuario.

5.3.2. Primer acercamiento conexión API

En un primer acercamiento, se utilizó la API directamente, enviando la información de la imagen a los servidores de Microsoft, y esperando de forma inmediata una respuesta, para esto, se crearon dos archivos.

- `extraer_texto.php`, en la carpeta `web`, encargado de recibir la información del `POST` realizado desde el *Front-end* y enviarlo al archivo correspondiente de la carpeta `include`.
- `ocr.class.php` archivo encargado de realizar la conexion con la API y recibir los resultados de esta.

Debido a que el archivo `extraer_texto.php` solamente hace un llamado a una función del archivo `ocr.class.php` se omitirá y se realizara la explicación únicamente del segundo archivo, que se detalla en el código 5.18.

```
1  <?php
2
3  class Ocr extends Objeto
4  {
5    static function textoDesdeImagen( $ruta_imagen )
6    {
7      $error = 0;
8      $max_tries = 10;
9      $headers = [
10       // Request headers
11       'Content-Type: application/octet-stream',
12       'Ocp-Apim-Subscription-Key: '.InfoNucleo::getParametro( 'apuntes', 'api_key_ocr' ),
13     ];
```

```
14      $body = UTIL::file_get_contents( $ruta_imagen );
15
16      $response = wget_apuntes(InfoNucleo::getParametro( 'apuntes', 'url_ocr' ), $body, 10,
        [], $headers);
17
18      preg_match( "/Operation-Location: ([\S]+)/", $response, $m);
19
20      $response_2 = ["status" => "notStarted"];
21      $tries = 0;
22      while(in_array($response_2["status"], ['notStarted', 'running']) && $tries < $max_tries)
        {
23        sleep(0.5);
24        $response_2 = UTIL::json_decode(UTIL::wget($m[1], [], 10, [], $headers));
25        $tries++;
26      }
27      $text = "";
28      foreach($response_2['analyzeResult']['readResults'][0]['lines'] as $line ){
29        $text = $text.'<p>'.$line['text'].'</p>';
30      }
31      if($tries == $max_tries){
32        $error = "El sistema se esta demorando mucho, intenta nuevamente despues de un tiempo"
        ;
33      }
34      return ['texto' => $text, 'error'=> $error];
35    }
36  }
```

Código 5.18: Primera implementación de llamado a API en `ocr.class.js`

En el código 5.18 se define la totalidad del código necesario para realizar el envío y recolección de resultados usando la API en esta primera iteración.

Primero, se hace la definición de los *headers* y el *body* de la *request* a realizar, en las líneas 9, 14 a 16.

Luego, es necesario utilizar una expresión regular para conseguir la URL donde estará ubicado el resultado de la consulta, a partir del *header* de la respuesta, como se explicó en el capítulo 2.

Luego, se hace un *busy-waiting* del resultado en las líneas 22 a 26, donde se lee la respuesta de hacer una consulta a la url recibida durante el *regex* anterior, este ciclo se romperá si se encuentra información o si se cumple un máximo de intentos definidos en la variable `$max_tries`. Una vez terminado el *busy-waiting*, si se encuentra texto en la respuesta, este se retorna para que el código 5.16 lo ingrese en el cuadro de edición.

5.3.3. Complicaciones con primer acercamiento

Este primer acercamiento mostró problemas de forma casi inmediata, debido a que ocurrió que, durante horas, la API no produjo ningún resultado. En pruebas, eliminando el límite del ciclo de *busy-waiting*, se detectó que, en ciertos casos, que se obtuviera un resultado en la URL obtenida podría demorar horas.

Aún más, en caso de error, si el usuario decide intentar nuevamente, se realiza una nueva primera consulta a la API para iniciar el proceso con la misma imagen anterior, perdiéndose el proceso realizado anteriormente, que eventualmente daría un resultado.

Debido a esto, fue necesaria la ideación de una segunda forma de consultar la API, que evitara en lo posible el ciclo de *"busy-waiting"* y que sea capaz de detectar si una imagen ya ha sido procesada anteriormente.

5.3.4. Implementación final de conexión con API

Debido a las consideraciones explicadas anteriormente, fue necesario el diseño de una nueva implementación, esta comenzó por el diseño de una tabla en la base de datos que permita guardar los datos necesarios para detectar si una imagen ya ha sido procesada por el sistema y poder evitar consultar a la API más de lo necesario.

Esta tabla, nombrada `IMAGENES` se diseñó con las siguientes columnas:

- `MD5`: Corresponde al hash MD5 [24] de la imagen a guardar, se utiliza este hash como una implementación sencilla de asegurar que permitirá identificar si una imagen existe en la tabla o no.
- `URL`: Corresponde a la URL regresada por la API después de la primera consulta, donde se puede consultar por el resultado del procesamiento de esta.
- `TEXTO`: Corresponde al texto extraído desde la imagen por la API.
- `FECHA`: Corresponde a la fecha en que se inició el procesamiento de la imagen.
- `FECHA_M`: Corresponde a la fecha de la última modificación a los atributos correspondientes a la imagen.

Y la implementación de esta tabla se realizó como se muestra en la figura 5.19.

Field	Type
IMG_MD5	varchar(32)
IMG_URL	longtext
IMG_TEXTO	longtext
IMG_FECHA	datetime
IMG_FECHA_M	datetime

Figura 5.19: Estructura tabla `IMAGENES`

Ahora, utilizando esta tabla se realizaron las siguientes modificaciones a los códigos `extraer_texto.php` y `ocr.class.php`.

```php
<?php
require('config.php');

$ruta_imagen = $_FILES['file']['tmp_name'];
$md5 = md5_file( $ruta_imagen );
$imagen = Ocr::get( $md5 );

if( ! $imagen || ! $imagen['reutilizable'] ) {
  $result_api = Ocr::textoDesdeImagen( $ruta_imagen );
  Ocr::upd( [
    'texto' => $result_api['texto'],
    'url' => $result_api['url'],
    'md5' => $md5,
  ] );
  KERNEL::streamAPI( $result_api );

} else {
  if( $imagen['texto'] ) KERNEL::streamAPI( [
    'texto' => $imagen['texto'],
    'error'=> "",
    "extra" => "imagen ya procesada"
  ] );
  $result_api = Ocr::consultarUrlImagen( $imagen['url'] );
  Ocr::upd( [ 'texto' => $result_api['texto'] ] + $imagen );
  KERNEL::streamAPI( $result_api + [ "extra" => "nueva consulta imagen existente" ] );
}
```

Código 5.19: Segunda implementación codigo `extraer_texto.php`

El código 5.19 representa la implementación final del archivo `extraer_texto.php`, la cual se puede dividir en dos partes.

Primero, se recibe la imagen desde el usuario, se calcula el hash MD5 de esta y se revisa si esta existe en la base de datos, en las líneas 4, 5 y 6 respectivamente.

Luego, dependiendo de si la imagen existe o se considera reutilizable, parámetro que es explicado en el codigo 5.20), se toma la decisión entre iniciar el proceso de extracción del texto, consultar la URL ya provista, o retornar al usuario el texto ya extraído.

La primera opción está definida entre las líneas 9 a 15 del código, donde se llama a la función de la clase `Ocr` para realizar el proceso de extracción de texto en su totalidad, para luego actualizar los valores de la imagen en la base de datos y finalmente realizar el envío del resultado de la operación para ser mostrado al usuario.

Luego, en las líneas 18 a 22, se tiene el caso cuando la imagen ya ha sido procesada y se tiene el texto extraído en la base de datos, en ese caso, simplemente se realiza el envío de los datos ya presentes en la base de datos. La función `KERNEL::streamAPI` realiza la salida de la función cuando se ejecuta, por ende las líneas siguientes no se ejecutan.

Finalmente, en caso de que la imagen haya sido procesada, pero no se tiene texto, se utiliza la función `Ocr::consultarUrlImagen` para verificar el estado de la operación de extracción, se actualiza el texto en la base de datos y se hace el envío del resultado.

El código de `ocr.class.php` sigue la misma estructura general del código 5.2, con una función `get`, `upd` y `norm` modificadas para consultar y actualizar la tabla `IMAGENES`. En particular, la función `norm` tiene una actualización adicional, como se puede ver en el fragmento de código 5.20.

```
23  static function norm( $i ){
24    $i['md5'] = $i['IMG_MD5'];
25    $i['url'] = $i['IMG_URL'];
26    $i['texto'] = $i['IMG_TEXTO'];
27    $i['fecha_creacion'] = $i['IMG_FECHA'];
28    $i['t_modificado'] = $i['IMG_FECHA_M'];
29    $i['reutilizable'] = $i['texto'] || strtotime( $i['IMG_FECHA'] ) >= strtotime( "-2 days"
       );
30    return $i;
31  }
```

Código 5.20: Función `norm` de `ocr.class.php`

En la línea 29 del fragmento de código 5.20, se puede observar que el parámetro `reutilizable` de arreglo se define como la presencia de texto o una fecha de creación con menos de dos días de antigüedad. De esta forma, si ya se tiene texto presente en la base de datos, siempre será usado. Y en caso de no existir texto y la URL usada tiene más de dos días de antigüedad, significa que es necesaria una nueva URL debido al funcionamiento de la API explicado en el capítulo 2.

Luego, la función `textoDesdeImagen()` utilizada en el archivo `extraer_texto.php` se muestra en el fragmento de código 5.21.

```
1   static function textoDesdeImagen( $ruta_imagen ) {
2     $api_key = InfoNucleo::getParametro( 'apuntes', 'api_key_ocr' );
3     $url_ocr = InfoNucleo::getParametro( 'apuntes', 'url_ocr' );
4     $headers = [
5       // Request headers
6       'Content-Type: application/octet-stream',
7       'Ocp-Apim-Subscription-Key: '.$api_key,
8     ];
9     $body = UTIL::file_get_contents( $ruta_imagen );
10
11    $response = wget_apuntes( $url_ocr, $body, 10, [], $headers );
12
13    preg_match( "/Operation-Location: ([\S]+)/", $response, $m );
14
15    return static::consultarUrlImagen( $m[1] );
16  }
```

Código 5.21: Función `textoDesdeImagen` de `ocr.class.php`

El fragmento 5.21 realiza la primera parte del proceso de uso de la API, enviando la información de la imagen y obteniendo la URL correspondiente a consultar, a realizar en el código.

Este funcionamiento es equivalente a las líneas 9 a 18 del código 5.18 de la primera iteración.

Ahora, para la segunda función, `consultarUrlImagen()`, tenemos el fragmento de código 5.22.

```
1   static function consultarUrlImagen( $url ){
2     $api_key = InfoNucleo::getParametro( 'apuntes', 'api_key_ocr' );
3     $error = 0;
4     $max_tries = 5;
5     $headers = [
6       // Request headers
7       'Content-Type: application/octet-stream',
8       'Ocp-Apim-Subscription-Key: '.$api_key,
9     ];
```

```
10      $response_2 = [ "status" => "notStarted" ];
11      $tries = 0;
12      while( in_array( $response_2["status"], [ 'notStarted', 'running' ] ) && $tries <
        $max_tries ) {
13        sleep(0.5);
14        $response_2 = UTIL::json_decode( UTIL::wget( $url, [], 10, [], $headers ) );
15        $tries++;
16        // print_a($response_2);
17      }
18      $text = "";
19      foreach( $response_2['analyzeResult']['readResults'][0]['lines'] as $line ){
20        $text = $text.$line['text']."<br/>";
21      }
22      if( $tries == $max_tries ) {
23        $error = "El sistema se esta demorando mucho, intenta nuevamente despues de un tiempo"
        ;
24      }
25      return [ 'texto' => $text, 'error'=> $error, 'intentos' => $tries, 'url' => $url, '
        last_estado' => $response_2["status"] ];
26    }
```

Código 5.22: Función `consltarUrlImagen` de `ocr.class.php`

En el fragmento de código 5.22 se puede observar una similitud con las líneas 20 a 34 del código 5.18, donde, luego de tener definidos los *headers* y parámetros, se realiza un *busy-waiting* esperando la respuesta sobre el estado del procesamiento de la imagen. Si se encuentra un texto, este se define dentro de la variable `$text`, y, en caso de que se ha alcanzado el máximo de intentos, se agrega un estado de error.

Finalmente, en el arreglo a retornar, en la línea 25, se han agregado otros parámetros para propósitos de *debug*, que en una implementación oficial dentro de la plataforma deberán ser eliminados.

5.4. Resolución de conflictos

En este punto de este documento, ya se ha realizado la explicación de todas las formas en que un usuario puede modificar el estado de un documento y de los cuadros dentro de este. Sin embargo, el objetivo de esta memoria es ser capaz de generar una herramienta que pueda ser utilizada por múltiples miembros de un mismo curso.

Por ende, un punto importante a implementar fue la forma de resolver conflictos entre múltiples usuarios de la herramienta. Estos conflictos, como fueron definidos en el capítulo 4, se diseñó que el usuario que haya generado el conflicto sea el encargado de su solución, por ende, se implementó la interfaz mostrada en la figura 5.20.

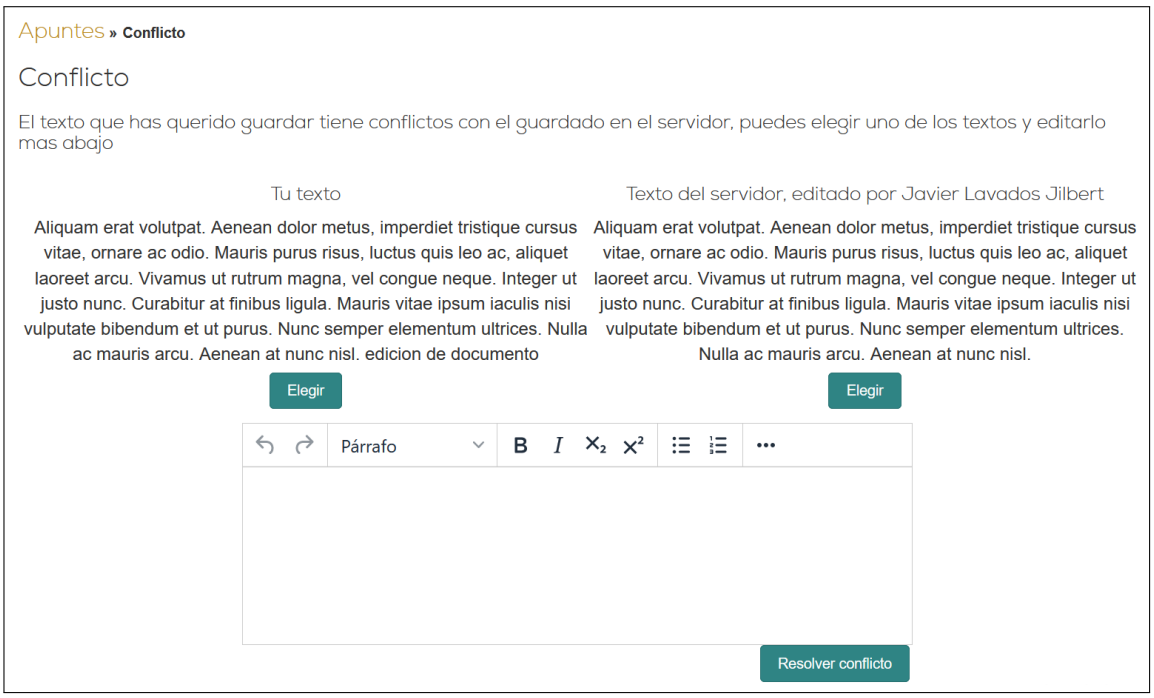

Figura 5.20: Interfaz conflicto

Como se puede observar en la figura 5.20, el usuario que haya generado el conflicto al momento de editar podrá ver, en el costado derecho, el texto que intento editar y en el costado izquierdo el texto de la otra persona.

Utilizando los botones con la leyenda *elegir*, el usuario será capaz de seleccionar uno de los textos, el cual será colocado en el cuadro de edicion de más abajo, permitiendo al usuario realizar nuevas ediciones en caso de ser necesario, como se puede ver en la figura 5.21.

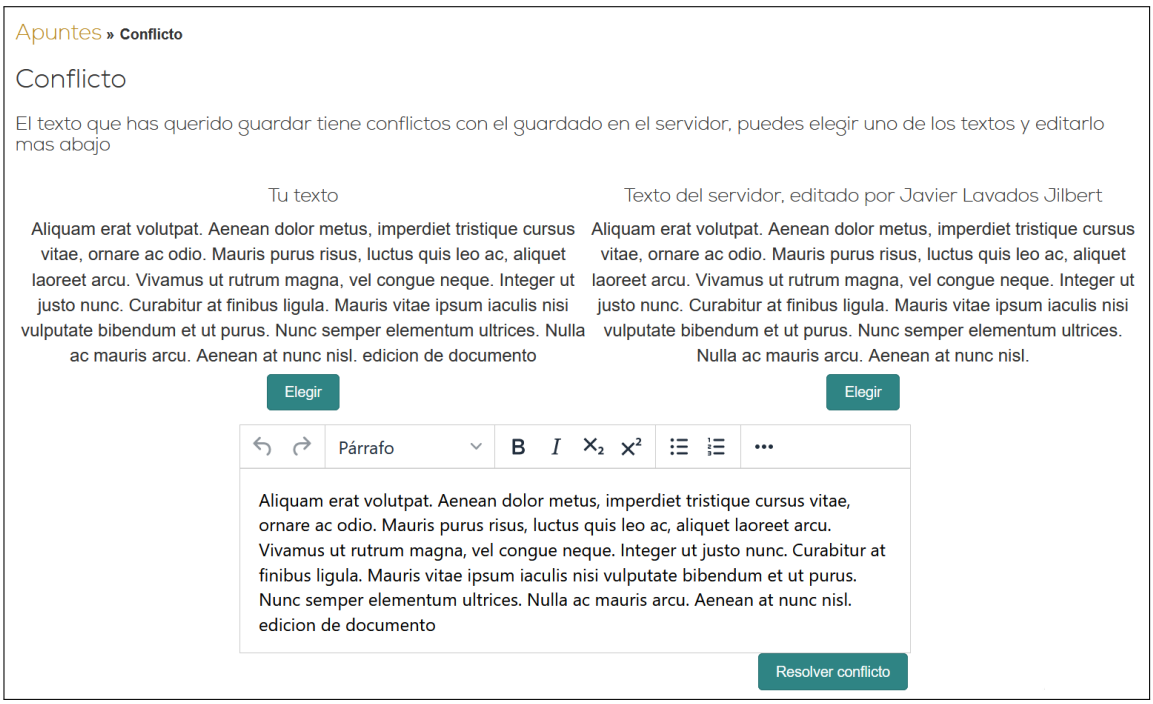

Figura 5.21: Cuadro de edicion relleno con texto de usuario

Ahora, para la detección de un conflicto, se ocupa el mismo fragmento de código que el utilizado para la edicion de texto, que se repite en el código 5.23.

```
1  } elseif( $_REQUEST['accion'] == "editar_cuadro" ) {
2    $cuadro = $cuadros[ $_REQUEST['cuadro_id'] ];
3    if( $_REQUEST['cuadro_id'] && ! $cuadro ) KERNEL::mensaje( '-Error al modificar el cuadro'
       );
4
5    $cuadro['contenido'] = $_REQUEST[ 'cuadro_'.$cuadro['id'] ];
6    $cuadro['pers_id'] = $pers_id;
7
8    if( ! KERNEL::mensaje() ) {
9      $t_modificado = strtotime( $cuadro['t_modificado'] );
10     if( $_REQUEST['tiempo_carga'] < $t_modificado ){
11       $conflicto = [
12         'cuad_id' => $cuadro['id'],
13         'doc_id' => $d['id'],
14         'cuad_contenido' => $cuadro['contenido'],
15         'pers_id' => $pers_id,
16       ];
17       Conflicto::upd( $conflicto );
18       KERNEL::redirect( 'conflicto?doc_id='.$d['id'] );
19
20     } else{
21       Cuadro::upd( $cuadro );
22       Documento::upd( [ 'id_modificado' => $cuadro['id'] ] + $d );
23       KERNEL::redirect( 'documento?id='.$d['id'], '+Cuadro editado' );
24     }
25   }
```

Código 5.23: Fragmento de código para editar un cuadro de `documento.php`

En el código 5.23 se puede observar, en la línea 10, que se hace una comparación entre el tiempo de carga de la página por el usuario y el tiempo en que se realizó la última modificación al cuadro que se está editando.

Si se da el caso que el tiempo de carga sea anterior al tiempo de la última edicion, significa que se ha producido un conflicto. Para la resolución de este conflicto, se crea un arreglo dentro de la variable `$conflicto` con el `id` del cuadro, el `id` del documento donde ha ocurrido el conflicto, el contenido que se pretende cambiar en el cuadro, y la persona responsable sobre la creación del conflicto.

Utilizando este arreglo, se realiza un *upsert* (update + insert) en la tabla `CONFLICTOS` y se realiza una redirección del usuario hacia la URL donde se resuelve el conflicto.

El archivo `conflicto.php` de la carpeta `web` se encarga de la renderización de esta pagina, este archivo se puede ver en su totalidad en el código 5.24.

```
1  <?php
2  require( 'config.php' );
3
4  $id = (int)$_REQUEST['doc_id'];
5  if( ! $id ) KERNEL::error( ERR_NO_EXISTE );
6
7  $d = Documento::get( $id );
8  if( ! $d ) KERNEL::error( ERR_NO_EXISTE );
9
10 $conflicto = Conflicto::get( $pers_id, $d['id'] );
11 $conflicto = array_pop( array_reverse( $conflicto ) );
12
13 $cuadro_servidor = Cuadro::get( $id, $conflicto['cuad_id'] );
14 $time = time();
15 $persona = UCURSOS::getDatosPersona($cuadro_servidor['pers_id']);
16
17 if(  $_REQUEST['accion'] == "editar" ){
18   $cuadro_editado = $cuadro_servidor;
19   $cuadro_editado['contenido'] = $_REQUEST['cuadro'];
20   $t_modificado = strtotime( $cuadro_servidor['fecha_modificacion'] );
21
22   //se marca el conflicto como resuelto
23   Conflicto::upd( [ 'estado' => 2 ] + $conflicto );
24   if( $_REQUEST['tiempo_carga'] < $t_modificado ) {
25     Conflicto::upd( $cuadro_editado );
26     KERNEL::redirect( 'conflicto?doc_id='.$d['id'] );
27   }
28   else{
29     Cuadro::upd( $cuadro_editado );
30     Documento::upd( [ 'id_modificado' => $cuadro_editado['id'] ] + $d );
31     KERNEL::redirect( 'documento?id='.$d['id'], '+Cuadro editado' );
32   }
33 }
34
35 KERNEL::head( 'Conflicto' );
36 require( template() );
37 KERNEL::foot();
```

Código 5.24: Código para resolucion de conflictos en `conflicto.php`

En el código 5.24 se puede observar, que, en las líneas 4 a 8, se verifica la existencia del documento en el cual se está realizando la resolución de conflicto.

Luego, en las líneas 10 a 15 se crean las variables necesarias para la renderización de la página, que se realiza en las líneas 35 a 37.

Luego, cuando el usuario decide confirmar la resolución del conflicto, se ejecuta el código de las líneas 17 a 33, donde, en un principio, se copia la información del cuadro existente en el servidor, y se modifica el contenido de este, se actualiza el tiempo de modificación, para luego marcar el conflicto actual como resuelto.

Sin embargo, se puede dar el caso en que, mientras se realizaba la resolución de un conflicto, se haya realizado otra modificación al cuadro en cuestión, por ende, se verifica nuevamente la fecha de modificación del cuadro.

En caso de detectar un nuevo conflicto se añaden los datos del cuadro como una nueva fila en la tabla `CONFLICTOS` y se redirecciona al usuario nuevamente para resolver un nuevo conflicto.

En caso de que no se detecte un nuevo conflicto, se actualizan los datos del cuadro en la tabla `CUADROS`, se modifica el `id` del último cuadro modificado en los datos del documento actual y se redirecciona al usuario al documento.

5.4.1. Avatares al costado de cuadros

Una idea que surgió durante el desarrollo de esta memoria fue el intentar disminuir las posibilidades de que se generen conflictos, para evitar que se pierda el trabajo hecho por una persona y hacer más sencilla la colaboración para todos los editores de un documento.

Con esto en mente, la solución implementada a este problema consiste en mostrar el avatar de todos los usuarios que estén editando un cuadro dentro del documento. De esta forma, el usuario podrá visualizar rápidamente si otro usuario está realizando una edicion de un cuadro, como se puede observar en la figura 5.22.

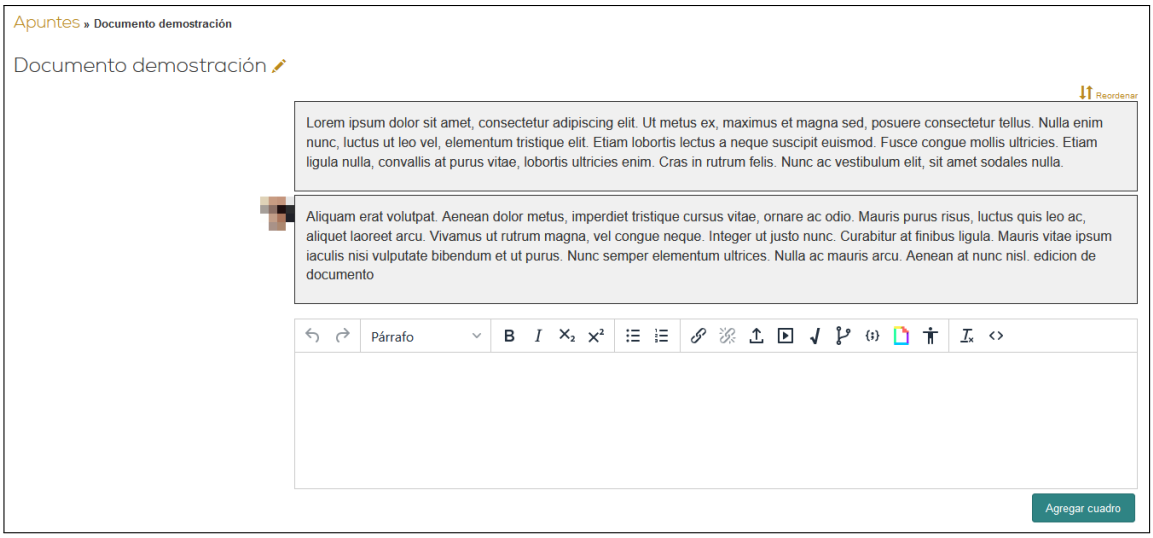

Figura 5.22: Recuadro anonimizado de una persona al costado de un cuadro

Para la realización de esta funcionalidad, fue necesaria la implementación de un sistema de mensajes utilizando un sistema ya existente dentro de *U-Cursos*.

El sistema existente permite la difusión de un mensaje a todos los subscriptores de un canal específico. Mediante el uso de un canal por cada documento, se puede enviar un mensaje cada vez que un usuario empieza la edicion de un cuadro, de esta forma, se puede comunicar de forma rápida el estado de todas las personas editando un cuadro en particular.

El código encargado de esta funcionalidad está dentro del archivo `documento.js` y se explicará a continuación empezando por la iniciación del `socket` para la conexión en el código 5.25.

```
1  vw = {
2    socket: null,
3    init: function() {
4      vw.socket = io.connect( info_host_clicks )
5      .on( 'connect', function() {
6        vw.socket.emit( 'init', 'apuntes_documento_{$d["id"]}' );
7        vw.socket.emit( 'pub', '"connect"' );
8      } )
9      .on( 'error', function() { console.log( 'Error al conectar' ); } )
10     .on( 'disconnect', function() { console.log( 'Desconectado' ); } )
11     .on( 'message', vw.show );
12
13   },
```

Código 5.25: Inicio de socket en `Documento.js`

Se puede observar que se define la conexión con el host existente en la línea 4, donde, al momento de conectar, se envía un mensaje con el nombre del canal al que se desea conectar, mediante el mensaje con el identificador `"init"` y el contenido corresponde al canal a conectar, en la línea 6.

Luego, se envía un mensaje a difundir, mediante el identificador `"pub"` con el contenido `"connect"`, el comportamiento al difundir estos mensajes se verá en el código 5.27.

Finalmente, en las líneas 9 a 11, se define las acciones a realizar al recibir un error, una desconexión, o un mensaje en general.

```
1    edit: function( cuad_id ) {
2      cuad_id_editando = cuad_id;
3      vw.socket.emit( 'pub', JSON.stringify( {
4        id: pers_username,
5        cuad_id: cuad_id,
6        editando: 1,
7        alias: pers_alias,
8        foto: pers_avatar
9      } ) );
10   },
11   stop_edit: function(cuad_id){
12     cuad_id_editando = null;
13     vw.socket.emit( 'pub', JSON.stringify( {
14       id: pers_username,
15       cuad_id: cuad_id,
16       editando: 0,
17     } ) );
18   },
```

Código 5.26: Definición de `vw.edit` y `vw.stop_edit` en `Documento.js`

Ahora, en el código 5.26 se definen dos funciones que son utilizadas cuando el usuario empieza a editar o termina la edicion de un cuadro, se puede ver en las líneas 2 y 12 que se edita el valor de la variable `cuad_id_editando`, y luego se hace la emisión de un mensaje con el identificador `"pub"` que contiene la información necesaria para identificar a la persona que está editando o dejo de editar el cuadro correspondiente.

```
1  show: function( data ) {
2    var data = JSON.parse( data );
3    if( data == 'connect') {
4      // si otro usuario se conecta deberia enviarle el cuadro que estoy editando en ese
     momento
5      if (cuad_id_editando == null) return;
6      vw.socket.emit( 'pub', JSON.stringify( {
7        id: pers_username,
8        cuad_id: cuad_id_editando,
9        editando: 1,
10       alias: pers_alias,
11       foto: pers_avatar
12     } ) );
13
14   } else if( data.editando == 1 ) {
15     if( data.id == pers_username )  return;
16
17     if( $( '#viewers_' + data.cuad_id + '> img' ).length < 5 ) {
18       $( "#viewers_" + data.cuad_id ).append( $( '<img/>', {
19         id: data.id,
20         src: data.foto,
21         class: 'foto chica',
22         title: data.alias,
23       } ) );
24     }
25
26   } else if( data.editando == 0 ){
27     if( data.id == pers_username ) return;
28     $( "#viewers_" + data.cuad_id + ' #' + data.id ).remove();
29
30   } else if( data == 'disconnect' ) {
31     $( "div[id*='viewers'] img" ).remove();
32     vw.socket.emit( 'pub', '"connect"' );
33   }
34 },
35 }
```

Código 5.27: Definicion de `vw.show` en `Documento.js`

Finalmente, se tiene la implementación de la función show en el código 5.27, esta función está encargada de leer la información recibida en cada mensaje enviado al canal y actuar acordemente.

Primero, si recibe el mensaje con el identificador `"connect"` significa que otro usuario ha realizado una conexión al documento y, en caso de que se esté realizando la edicion de un cuadro, se le hace envío de un mensaje con el identificador `"pub"` que será distribuido a todos los subscriptores al canal. De esta forma el usuario que se ha conectado recibirá un mensaje de todos los usuarios que estén editando un cuadro en ese momento.

Luego, si dentro de la variable `data` el parámetro `editando` tiene valor 1, quiere decir que se ha recibido un mensaje de que un usuario a iniciado la edicion de un cuadro.

En este caso, si el usuario del mensaje corresponde al mismo usuario, este se ignora. Si corresponde a otro usuario, se agregará su avatar al costado del cuadro correspondiente, que se puede observar en la línea 7 del código 5.12.

Luego, en caso de que un usuario termine la edicion de un cuadro, se eliminara el avatar correspondiente de la misma sección de código anterior.

Por último, si un usuario se desconecta, que ocurre cuando el usuario cambia de página

dentro del sitio, no se tiene identificación del usuario que ha realizado la desconexión, por lo que se elimina la totalidad de usuarios y se realiza un mensaje con el parámetro `"connect"` que permitirá recopilar la información de todos los usuarios editando un cuadro en ese momento.

Con la implementación de la herramienta ya discutida, se realizaron diferentes validaciones del sistema creado, las cuales serán discutidas en el capítulo 6.

Capítulo 6

Validación

En los capítulos 4 y 5 se discutieron las decisiones realizadas y la implementación de la herramienta desarrollada. Sin embargo, esto no es suficiente para decidir si el trabajo realizado es utilizable por usuarios finales y si es útil para los fines que fueron definidos. Para esto, se realizaron 2 validaciones principales, una dentro del Centro Tecnológico Ucampus y otra con estudiantes de diferentes Facultades y carreras de múltiples universidades.

Se discutirán las diferentes fortalezas y debilidades de las validaciones realizadas y el conocimiento que se puede extraer de estas.

6.1. Validación con Centro Tecnológico Ucampus

La primera instancia de validación a discutir fue la realizada dentro del centro tecnológico Ucampus, que fue realizada de forma presencial y consistió en una reunión con el equipo de desarrollo. En esta reunión, se realizó una presentación del estado actual del módulo, mostrando las diferentes funcionalidades existentes en ese momento.

6.1.1. Primera validación

La primera validación realizada de esta forma, fue realizada durante el mes de septiembre, fue realizada antes de la implementación del sistema OCR al módulo. Participaron el profesor guía Willy Maikowski y 4 otros miembros del equipo de desarrollo del centro.

El feedback consistió principalmente en algunas críticas a decisiones tomadas en ese momento, que se tomaron en cuenta para mejorar el módulo final, algunas de estas críticas fueron con respecto al sistema de resolución de conflictos, el espacio en algunos elementos, y la forma de realizar movimientos de cuadros, que en su momento consistía en un botón por cada cuadro que recargaba la totalidad de la página al ser utilizado.

Otro punto importante que se dio feedback, fue la idea de diferentes fases dentro de

la toma de apuntes, que fue un aspecto no considerado en un comienzo, pero que termino siendo una definición importante en el trabajo realizado en esta memoria. Junto con ideas para desarrollos de otras herramientas para el resto de fases de toma de apuntes, algunas de estas son discutidas en el capítulo 7.

6.1.2. Segunda validación

La segunda validación se realizó a finales del mes de octubre, y participó el profesor guía Willy Maikowski y 5 otros miembros del equipo de desarrollo. En esta segunda instancia de validación ya se tenía prácticamente la totalidad de funcionalidades e interfaces creadas.

El feedback recibido en esta ocasión se centró principalmente el diseño escogido para el módulo y las diferencias que este tenía con respecto al diseño ocupado normalmente en la plataforma *U-Cursos*, por ejemplo, originalmente para editar el título de un texto se tenía un botón en la lista de todos los documentos, que fue cambiado por un icono de un lápiz dentro del documento en sí.

Finalmente, un punto a considerar para ambas validaciones es que el feedback no fue realizado por usuarios finales del módulo, sino que por desarrolladores de la plataforma, por ende, este feedback tiene un alto valor técnico, pero no necesariamente feedback con respecto a la usabilidad del módulo, por ende, fue necesaria la validación de estudiantes para lograr un balance entre ambas.

6.2. Validación con estudiantes

Debido a la necesidad de obtener una retroalimentación más concreta desde estudiantes universitarios, se hizo una búsqueda de estos a través de diferentes plataformas y se realizó una entrevista a cada uno.

En total, se realizaron 5 entrevistas a estudiantes de distintas Universidades, estas entrevistas consistieron en 3 partes principales. Primero, se les pidió a los entrevistados realizar una serie de acciones dentro del módulo, para luego realizar una encuesta para obtener un puntaje SUS (*System Usability Scale*) y finalmente tener una conversación personal para obtener una apreciación más objetiva sobre la opinión de la herramienta realizada.

Los estudiantes entrevistados fueron personas de distintas facultades y universidades, lo que permite tener una amplia perspectiva en la retroalimentación recibida, estas se listan a continuación:

- Universidad de Chile - Facultad de Ciencias Físicas y Matemáticas
- Universidad de Chile - Facultad de Ciencias Sociales
- Florida International University - Department of Criminology and Criminal Justice
- Universidad de la República de Uruguay - Centro Universitario Regional Litoral Norte

6.2.1. Acciones realizadas

Las acciones a realizar por los estudiantes fueron las siguientes, para cada entrevista se limpio la totalidad de documentos y cuadros creados dentro de la herramienta, para que cada entrevistado tuviera la misma experiencia.

1. Crear documento con permiso para todos los miembros del curso
2. Crear documento solo con permiso para el entrevistado
3. Borrar uno de los documentos creados
4. Abrir el documento e insertar un cuadro
5. Editar el cuadro insertado
6. Agregar un segundo cuadro con la función OCR
7. Intercambiar los cuadros de orden

Luego de las primeras acciones, el entrevistador inicio sesión con otro usuario dentro del mismo curso, para simular a múltiples personas realizando una edicion al documento, y se le pidió al usuario elegir un cuadro para editar sin explicar el propósito de las imágenes para ver su efectividad.

Finalmente, mediante el uso de la segunda sesión iniciada se fuerza un conflicto en la edicion del entrevistado y se le pidió solucionarlo como le pareciera conveniente.

6.2.2. Escala SUS

La escala SUS es un cuestionario estandarizado que está designado para medir la usabilidad percibida de una herramienta [10].

La versión estándar tiene 10 ítems cada uno con 5 respuestas posibles, que tienen valores de 1 a 5, desde “Totalmente de acuerdo” a “Totalmente en desacuerdo”. Las preguntas impares están planteadas de forma positiva y las preguntas impares de forma negativa [10].

Las preguntas que componen a la escala se listan a continuación:

1. Creo que me gustaría utilizar este sistema con frecuencia
2. Encontré el sistema innecesariamente complejo
3. Pensé que el sistema era fácil de usar
4. Creo que necesitaría el apoyo de un técnico para poder utilizar este sistema
5. Encontré que las diversas funciones de este sistema estaban bien integradas
6. Pensé que había demasiada inconsistencia en este sistema

Puntaje	87,5	87,5	87,5	90	92,5

Tabla 6.1: Resultados encuesta SUS

7. Me imagino que la mayoría de la gente aprendería a utilizar este sistema muy rápidamente
8. Encontré el sistema muy complicado de usar
9. Me sentí muy seguro usando el sistema
10. Necesitaba aprender muchas cosas antes de empezar con este sistema

Una vez que se tienen las respuestas a cada una de las preguntas, se le asigna un valor a cada ítem desde 0 a 4. Para las preguntas impares, se toma el valor de la respuesta menos 1 y para preguntas pares se toma el valor 5 menos el valor de la respuesta. Luego, se suma el puntaje para cada ítem y se multiplica por $2,5$.

Esto produce un puntaje final, el cual, si corresponde a 70 hacia arriba, se considera como una herramienta que presenta un alto grado de usabilidad percibido por los usuarios evaluadores, mientras que cualquier valor menor indica alguna problemática grave que debería ser solucionada.

6.2.3. Resultados Escala

Luego de la realización de las acciones de cada entrevistado, se le pidió responder la encuesta SUS y se recopilaron sus resultados en la tabla 6.1, donde se puede observar que los puntajes obtenidos fueron principalmente positivos, teniendo un puntaje promedio de 89 puntos.

Este puntaje, si bien es considerablemente positivo, debe ser considerado tomando en cuenta que la cantidad de estudiantes no es representativa de la mayoría de un curso. Por lo que se plantea la continuación de la verificación para obtener un resultado más concreto en el futuro.

6.2.4. Comentarios Individuales

Por último, se realizó una conversación abierta con cada alumno entrevistado, para obtener una perspectiva más general sobre la experiencia del uso de la herramienta.

Ya que discutir cada punto hablado con cada estudiante sería una tarea bastante larga, se discutirán los puntos repetidos entre los estudiantes.

Utilidad del sistema OCR

El sistema OCR fue recibido positivamente por la totalidad de los estudiantes entrevistados, quienes lo consideran como un elemento que ayudaría considerablemente el transferir apuntes desde papel a digital.

Sin embargo, también se hizo notar que la funcionalidad es difícil de encontrar de forma natural en la aplicación y más de una persona tuvo que pedir ayuda para encontrar el botón correspondiente, por lo que se considera necesaria una evaluación de la presentación de la funcionalidad.

Avatares al costado de cuadros

Cuando se realizó la acción de editar un cuadro con intervención del entrevistador para mostrar avatares, en ningún caso se entendió la intención de estos por los entrevistados, quienes siempre optaron por editar el cuadro que tenía el avatar al costado.

Aun así, un comentario que se repitió es que, una vez explicada la funcionalidad de estos avatares, se entendió su propósito de inmediato y consideran que sería algo que tendrían en cuenta en el futuro.

Finalmente, con la validación mostrada en este capítulo, podemos extraer las conclusiones del trabajo realizado en el capítulo 7.

Capítulo 7

Conclusión

Una vez realizado el análisis, diseño, implementación y verificación del módulo planteado en esta memoria, se pueden extraer las siguientes conclusiones de este.

7.1. Conclusiones trabajo realizado

Sobre el trabajo realizado en esta memoria, se logró el análisis, diseño e implementación del módulo planteado, el cual se considera en un estado utilizable, pero que aún puede ser mejorado en varios aspectos de este. Se pudo realizar la incorporación del sistema utilizando la API *Azure Cloud Vision* para la extracción de texto desde imágenes y se creó un sistema que permite la organización de la información en documentos que contienen cuadros, los cuales contienen el texto creado por los usuarios. Se permite agregar, editar, borrar y cambiar el orden de los cuadros creados, y se tiene un sistema para resolver los conflictos que ocurran, debido a que la edicion de un documento puede ocurrir por más de una persona al mismo tiempo.

Considerando lo anterior, se considera que el objetivo general "diseñar e implementar un sistema de apuntes para curso en la plataforma *U-Cursos*, que permita la edición en tiempo diferido de sus contenidos, sin dejar de lado a quienes prefieran mantener sus apuntes en papel ofreciendo una metodología de transcripción automática a digital" fue alcanzado de forma exitosa.

Con respecto a los objetivos específicos, la mayoría de estos fueron cumplidos, con la excepción del último punto "Validar el sistema creado con miembros de un curso real, para evaluar la usabilidad de lo realizado, durante un periodo de 3 semanas, y recopilar las opiniones de estos". La validación realizada se puede considerar insuficiente con respecto a este y se considera uno de los puntos importantes a evaluar para un trabajo futuro.

Este trabajo se considera con un gran valor potencial para funcionar como un complemento a la experiencia de estudio de un alumno, debido a que se da una plataforma que, de forma sencilla, permite el compartir los apuntes realizados y poder recibir mejoras a partir de otros estudiantes mediante el sistema de edicion y conflictos presentados. Además, debido a la

incorporación de la extracción de caracteres mediante el sistema OCR se da una comodidad adicional a estudiantes que solamente realicen sus apuntes en papel y lápiz, permitiéndoles interactuar con la plataforma al mismo nivel que un alumno que trabaje con herramientas digitales.

Finalmente, y de una forma más personal, uno de los aspectos a mejorar se considera la correcta estimación de los tiempos necesarios para avanzar en las distintas fases del trabajo, especialmente para la etapa de escritura del informe, donde se dio la impresión de que el tiempo no sería suficiente durante la escritura.

7.2. Trabajo a futuro

Con respecto al trabajo futuro que sería posible a partir de esta memoria, es importante considerar primero los puntos débiles del trabajo realizado, además, se agregan ideas que surgieron durante el desarrollo de este trabajo.

7.2.1. Debilidades trabajo realizado

Uno de los puntos débiles detectados corresponde a la resolución de conflictos en la plataforma, por un lado, la visualización de avatares al costado de los cuadros, vista en la sección 5.4.1, se encontró insuficiente cuando fue evaluada por los usuarios finales de la aplicación. Por otro lado, la forma de resolución de conflictos pide al usuario la comparación de dos textos en su totalidad, los cuales pueden tener una extensión considerable. Una solución posible sería resaltar las líneas o párrafos que contienen diferencias, logrando de esta forma que la comparación sea más sencilla.

Otra debilidad encontrada al trabajo realizado corresponde a la capacidad de detectar otros tipos de conflicto, debido a que la edicion no es la única acción que se puede hacer de forma paralela entre varias personas. Por ejemplo, puede ser interesante ver el resultado cuando se combinan las acciones de agregar cuadros, mover cuadros o editar cuadros por múltiples usuarios a la vez.

Finalmente, un punto de posible mejora correspondería a alertar al usuario cuando otros han realizado algún tipo de cambio, funcionalidad que puede apoyarse en el sistema ya existente de mensajes, pero que por consideraciones de tiempo no fue implementada en esta memoria.

7.2.2. Ideas adicionales

En la sección 3.2 se habló de las diferentes fases en el proceso de toma de apuntes, y, debido a que esta memoria se centró en una sola de estas partes, se puede realizar un desarrollo para abordar las dos restantes, por ejemplo, un sistema que permita a diferentes usuarios enviar

mensajes pequeños que puedan ser consolidados por una herramienta, para apoyar el proceso de toma de apuntes durante la clase.

Por otro lado, para apoyar el proceso de consolidación de apuntes antes de evaluación, puede ser posible el desarrollo de una herramienta que permita la recopilación de dos o más documentos producidos por el módulo desarrollado en esta memoria, agrupándolos en un único documento con la información relevante de ambos. De esa forma realizar resúmenes más rápida y efectivamente.

Finalmente, una idea adicional que puede ser incorporada directamente al trabajo realizado en esta memoria, es la incorporacion de otra API que permita la extraccion de texto a partir de una grabacion de una clase. Debido a que segun experiencia del autor, estudiantes del area de la salud tienden a grabar sus clases mas que tomar apuntes de estas.

Bibliografía

[1] Jaided Ai. Demo. https://www.jaided.ai/easyocr/. Accedido el 21/04/2022.

[2] Ber, Patrik Er, Lars-ola Damm, Jeanette Eriksson, Tony Gorschek, Kennet Henningsson, Per Jönsson, Simon Kågström, Drazen Milicic, Frans Mårtensson, Kari Rönkkö, and Others. Software quality attributes and trade-offs. *Blekinge Institute of Technology*, 97(98):19, 2005.

[3] Jamie Costley and Mik Fanguy. Collaborative note-taking affects cognitive load: the interplay of completeness and interaction. *Educational Technology Research and Development*, 69(2):655–671, 2021.

[4] Peter Denning, Jim Horning, David Parnas, and Lauren Weinstein. Wikipedia risks. *Communications of the ACM*, 48(12):152–152, 2005.

[5] Google. Convert pdf and photo files to text - google docs help. https://support.google.com/drive/answer/176692?hl=en, 2022. Accedido el 21/04/2022.

[6] Google. Google docs. https://docs.google.com/document/u/0/, 2022. Accedido el 21/04/2022.

[7] Iam. Iam handwriting database. https://fki.tic.heia-fr.ch/databases/iam-handwriting-database, 2022. Accedido el 07/04/2022.

[8] Tiny Technologies Inc. Tinymce: Custom toolbar menu button. https://www.tiny.cloud/docs/demo/custom-toolbar-menu-button/. Accedido el 05/12/2022.

[9] Keil Jacobs. A comparison of two note taking methods in a secondary english classroom. https://soar.wichita.edu//handle/10057/1388, 2022. Accedido el 19/04/2022.

[10] James Lewis and Jeff Sauro. Item benchmarks for the system usability scale. *Journal of Usability Studies*, 13:158–167, 05 2018.

[11] Microsoft. Microsoft azure portal: Microsoft azure. https://azure.microsoft.com/en-us/get-started/azure-portal, 2022. Accedido el 10/12/2022.

[12] Fausto Morales. Faustomorales/Keras-OCR: A packaged and flexible version of the craft text detector and Keras Crnn recognition model. https://github.com/faustomorales/keras-ocr, 2019. accedido el 21/04/2022.

[13] Pam A Mueller and Daniel M Oppenheimer. The pen is mightier than the keyboard: Advantages of longhand over laptop note taking. *Psychological science*, 25(6):1159–1168, 2014.

[14] Aakash Kumar Nain and Sayak Paul. captcha ocr tflite. `https://github.com/tulasiram58827/ocr_tflite/blob/main/colabs/captcha_ocr_tflite.ipynb`, Aug 2016. Accedido el 21/04/2022.

[15] Notion. Notion. `https://www.notion.so/`, 2022. Accedido el 19/04/2022.

[16] Openai. `https://beta.openai.com/docs/introduction/overview`. Accedido el 21/04/2022.

[17] Ben Stokes. I blew $720 on 100 notebooks from alibaba and started a paper website business. `https://daily.tinyprojects.dev/paper_website`, Dec 2021. Accedido el 21/04/2022.

[18] Microsoft Swhite-msft. Bing spell check api v7 reference. `https://docs.microsoft.com/en-us/rest/api/cognitiveservices-bingsearch/bing-spell-check-api-v7-reference`, Jun 2022. Accedido el 23/04/2022.

[19] Rebecca Tarnopol. How to ocr documents for free in google drive. `https://business.tutsplus.com/tutorials/how-to-ocr-documents-for-free-in-google-drive--cms-20460`, 2022. Accedido el 19/04/2022.

[20] Keras Team. Keras documentation: About keras. `https://keras.io/about/`, 2022. Accedido el 21/04/2022.

[21] Teton-l and Jake Is. The data model behind notion's flexibility. `https://www.notion.so/blog/data-model-behind-notion`, 2022. Accedido el 19/04/2022.

[22] Heather L Urry, Chelsea S Crittle, Victoria A Floerke, Michael Z Leonard, Clinton S Perry Iii, Naz Akdilek, Erica R Albert, Avram J Block, Caroline Ackerley Bollinger, Emily M Bowers, and Others. Don't ditch the laptop just yet: A direct replication of mueller and oppenheimer's (2014) study 1 plus mini meta-analyses across similar studies. *Psychological Science*, 32(3):326–339, 2021.

[23] Jakob VoSS. Measuring wikipedia. *Proceedings of ISSI 2005: 10th International Conference of the International Society for Scientometrics and Informetrics*, 1, 01 2005.

[24] Wikipedia. Md5. `https://en.wikipedia.org/wiki/MD5`, Nov 2022. Accedido el 05/12/2022.

[25] Wikipedia. Wikipedia. `https://en.wikipedia.org/wiki/Main_Page`, 2022. Accedido el 21/04/2022.

[26] Wikipedia. Wikipedia editing policy. `https://en.wikipedia.org/wiki/Wikipedia:Editing_policy`, 2022. Accedido el 21/04/2022.

[27] Dennis M Wilkinson and Bernardo A Huberman. Cooperation and quality in wikipedia. In *Proceedings of the 2007 international symposium on Wikis*, pages 157–164, 2007.