Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

SCHEMATIZE: App para el reconocimiento óptico de diagramas mentales dibujados a mano

Alumno: Juan Alberto Medina Martínez

Tutor: Prof. D. Francisco Charte Ojeda

Dpto: Informática

UNIVERSIDAD DE JAÉN

D./Dª Prof. D. Francisco Charte Ojeda, tutor(es) del Trabajo Fin de Grado titulado: **SCHEMATIZE: App para el reconocimiento óptico de diagramas mentales dibujados a mano**, que presenta Juan Alberto Medina Martínez, autoriza(n) su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, Julio de 2022

El estudiante

El tutor

CHARTE OJEDA FRANCISCO - 25996290B

Firmado digitalmente por CHARTE OJEDA FRANCISCO - 25996290B
Fecha: 2022.07.02 20:30:52 +02'00'

Juan Alberto Medina Martínez

Prof. D. Francisco Charte Ojeda

Tabla de contenidos

1. Introducción y motivación **1**

1.1. Introducción 1

1.2. Motivación 2

1.3. Objetivos 2

1.3.1. Objetivo general 2

1.3.2. Objetivos específicos 3

1.4. Estructura de la memoria 4

2. Especificación del proyecto **5**

2.1. Descripción de la situación inicial 5

2.2. Restricciones 5

2.3. Metodología de desarrollo software 6

2.4. Gestión y organización 7

2.5. Estimación de tamaño y esfuerzo del proyecto 8

2.6. Inventario y presupuesto 12

2.6.1. Estimación de recursos 12

2.6.2. Estimación de costes 13

3. Análisis de requisitos **15**

3.1. Diagrama de casos de uso 16
3.2. Requisitos funcionales 16
3.3. Requisitos no funcionales 19

4. Diseño **21**
4.1. Diagrama de clases 21
4.2. Diagramas de secuencia 23
4.3. Diseño de interfaz 25
4.3.1. Wireframe 25
4.3.2. Prototipo 26

5. Desarrollo **27**
5.1. Decisiones previas al desarrollo 27
5.2. Lenguaje, entorno y herramientas 29
5.2.1. Herramientas necesarias 29
5.3. Implementación 31
5.3.1. Aplicación base 32
5.3.2. Pruebas OCR 32
5.3.3. Detección de figuras 32
5.3.4. Reconocimiento y distinción de figuras 34
5.3.5. Detección de líneas 34
5.3.6. Reconocimiento de texto 36
5.3.7. Creación del diagrama 37

6. Pruebas y resultados **39**
6.1. Ejemplo 1 40

6.2. Ejemplo 2 41

6.3. Ejemplo 3 42

6.4. Ejemplo 4 43

7. Conclusión y observaciones finales **45**

7.1. Conclusiones 45

7.2. Aspectos a mejorar 46

8. Apéndices **47**

8.1. Manual de instalación 47

8.2. Manual de uso 49

Bibliografía II

Lista de figuras

3.1. Diagrama de casos de uso 16

4.1. Diagrama de clases 22

4.2. Diagrama de secuencia del caso de uso descrito en RF-01 23

4.3. Diagrama de secuencia del caso de uso descrito en RF-02 24

4.4. Wireframe 25

4.5. Prototipo 26

5.1. Comparación después de haber realizado las primeras transformaciones a la imagen 33

5.2. Distinción de contornos 33

5.3. Rectángulos conteniendo cada contorno 34

5.4. Obtención y tratamiento de relaciones 35

5.5. Relleno de píxeles vacíos para Tesseract 36

5.6. Creación de una elipse 37

6.1. Prueba 1 40

6.2. Resultado 1 40

6.3. Prueba 2 41

6.4. Resultado 2 41

6.5. Prueba 3 42
6.6. Resultado 3 42
6.7. Prueba 4 43
6.8. Resultado 4 43

8.1. Localización de la opción de generación del archivo APK en Android Studio 47
8.2. Ejecución del archivo APK desde el almacenamiento del teléfono 48
8.3. Confirmación de la instalación 48
8.4. Primera opción 49
8.5. Segunda opción 49
8.6. Pantalla tras terminar la creación del diagrama 50

Lista de tablas

2.1. Comparativa entre modelos 6
2.2. Tareas y estimaciones del proyecto 7
2.3. Cálculo de peso de casos de uso 8
2.4. Cálculo de peso de actor 8
2.5. Peso de los Factores Técnicos 9
2.6. Peso de los Factores Ambientales 10
2.7. Horas por persona dependiendo de los Factores Ambientales 11
2.8. Gastos amortizables 13
2.9. Salarios y gastos 13

3.1. Requisitos funcionales 16
3.2. RF-01 Crear diagrama a partir de imagen ya guardada 17
3.3. RF-02 Crear diagrama a partir de foto tomada de la cámara 18

5.1. Comparación entre plataformas 28

Capítulo 1

Introducción y motivación

En este capítulo se presentan la motivación y los objetivos necesarios para el desarrollo de este Trabajo de Fin de Grado. Se explica y muestra además la estructura que seguirá este documento.

1.1. Introducción

La importancia de los diagramas, ya sea para plasmar conocimientos o para transmitirlos, es simplemente innegable. Como prácticamente todo, cada persona tendrá una opinión distinta al respecto pero hasta los más negacionistas le asociarán cierto nivel de valor a este método de estudio o enseñanza más gráfico.

Tal y como pasa el tiempo y las tecnologías continúan evolucionando, cada vez más tipos de documentos manuscritos requieren el traslado a formato digital. Ya sea por cuestiones de seguridad, para facilitar su almacenamiento o simplemente portabilidad. Los diagramas, tanto mapas conceptuales, mentales o de cualquier otro tipo no son una excepción, por lo que se necesitarían mecanismos para digitalizarlos y poder almacenarlos de manera más flexible.

1.2. Motivación

La motivación principal de este trabajo proviene de la necesidad previamente nombrada en la introducción. Se requiere alguna manera en la que agilizar el traslado a formato digital, porque ya existen diversas alternativas para crear diagramas y esquemas, pero recrear los ya existentes desde el principio resultaría ser una tarea de lo más tediosa. De esta manera, lo ideal sería la existencia de algún tipo de aplicación con la capacidad de realizar este paso de manera instantánea, con la simple facilidad de realizar una fotografía al diagrama y obtenerlo de manera inmediata en un formato modificable y más flexible de almacenar digitalmente.

1.3. Objetivos

En esta sección presentamos los objetivos que necesitarán cumplirse a lo largo del desarrollo del proyecto SCHEMATIZE. Dividiendo en una visión general y especificando los objetivos en sí.

1.3.1. Objetivo general

- Diseño y desarrollo de una aplicación capaz de extraer de un diagrama manual las entidades propias de un mapa mental y generar una versión editable digitalmente del mismo. Esta aplicación constaría de una interfaz simple desde la que se podrá acceder a las opciones de tomar una imagen de la cámara o del almacenamiento del dispositivo. De esta imagen se reconocerán los elementos geométricos, las conexiones que las unen y el texto que contengan y de esos datos se creará y almacenará el diagrama en su nuevo formato.

1.3.2. Objetivos específicos

- Aplicación del proceso de desarrollo de software. Estudio de metodologías y modelos para la realización de cada etapa.
- Estudio y búsqueda de información técnicas de reconocimiento óptico de caracteres disponibles públicamente. Exposición de ventajas y desventajas de cada una.
- Análisis y propuestas de técnicas para la realización de las tareas de reconocimiento óptico de figuras y relaciones entre ellas.
- Examen de formatos flexibles disponibles para la creación de diagramas mentales y búsqueda de plataformas capaces de trabajar con estos formatos. Se explicará la manera en la que serán creados estos diagramas por el programa.
- Diseño de la aplicación que aúne las funcionalidades de reconocimiento óptico distintas. Elaboración de una interfaz adecuada a la plataforma que se seleccione.
- Selección y justificación de herramientas usadas y necesitadas: lenguaje, plataforma, bibliotecas y componentes.
- Implementación de una aplicación que utilizando las técnicas de reconocimiento apropiadas que facilite la digitalización de diagramas manuales.
- Comprobación de funcionalidad y calidad de la aplicación mediante la realización de pruebas.
- Redacción de manuales de instalación y uso.

1.4. Estructura de la memoria

La estructura del presente documento se divide en nueve capítulos, ordenados de la siguiente manera:

- **Capítulo 1. Introducción y motivación** Apartado actual en el que se presentan la motivación, los objetivos y la estructura de la memoria, así como una ligera introducción al tema.
- **Capítulo 2. Especificación** En este capítulo se explica con detalle la planificación para cumplir y resolver los objetivos anteriormente expuestos.
- **Capítulo 3. Análisis de requisitos** Obtención y clasificación de los requisitos identificados para el proyecto, incluyendo diagramas de casos de uso y narrativas para unas descripciones más elaboradas.
- **Capítulo 4. Diseño** Modelado de solución y comportamiento de la aplicación para cumplir los objetivos necesarios haciendo uso de diagramas de clases y de secuencia.
- **Capítulo 5. Desarrollo** En este capítulo se expondrán las diversas herramientas encontradas para las necesidades de reconocimiento óptico y tratamiento de imagen del proyecto. Justificación de las herramientas elegidas ante las alternativas de cada una. Este apartado también incluye todo el proceso de desarrollo de la aplicación, con información del progreso y sobre las dificultades encontradas a lo largo del proyecto y sus soluciones.
- **Capítulo 6. Pruebas** En este apartado se mostrarán las pruebas realizadas al prototipo y los resultados obtenidos.
- **Capítulo 7. Conclusiones** Este es el último capítulo de la memoria referente al proyecto. Contiene la conclusión a la que se ha llegado tras el desarrollo del proyecto así como algunas observaciones.
- **Capítulo 8. Apéndices** Manuales de instalación y uso de la aplicación.

Capítulo 2

Especificación del proyecto

2.1. Descripción de la situación inicial

Como se ha explicado previamente, realizar un traslado de diagramas mentales a formato digital desde el principio de forma manual resulta una tarea muy tediosa. Y está claro que este tedio aumentará de manera proporcional al volumen de datos, también pudiendo incrementar la posibilidad de fallos cometidos por el encargado de la tarea en cuestión. No solo el ahorro de tiempo, sino además la comodidad y poder evitar fallos humanos son los principales motivos por los que se crean este tipo de herramientas para ayudar en el proceso.

2.2. Restricciones

La duración del proyecto no podrá exceder las horas que equivalen a los 12 créditos que se le asocian, es decir, la duración total no podrá superar las 300 horas. En este tiempo se incluyen todas las etapas del ciclo de vida del proyecto, sin incluir el mantenimiento.
Tampoco se dispone de dispositivos en los que realizar pruebas y compilar recursos sobre iOS o Macintosh, por lo que una restricción adicional será el no poder desarrollar la aplicación para ninguno de estos sistemas operativos.

2.3. Metodología de desarrollo software

Es necesario escoger de entre tantas metodologías disponibles la mejor posible para nuestro proyecto, pues será decisiva a la hora de estructurar y planificar el proceso de desarrollo.
Puede parecer apresurado, pero se descartan metodologías de desarrollo ágil [1]. Es cierto que ofrecen características que vendrían bien en cualquier proyecto (como los resultados anticipados y la flexibilidad de Scrum [2]), pero siguen estando más enfocados a equipos y no tendría demasiado sentido en el caso en el que nos encontramos.

Modelo	Ventajas	Desventajas
En cascada [3]	Fácil y sencillo de implementar al ser lineal Se realizan pruebas después de cada etapa importante	Rígido, no se pueden deshacer cambios en caso de error Sin objetivos claros el proceso será mucho más difícil de hacer sin errores
Prototipos [4]	El riesgo de construir productos que no cumplan los objetivos deseados se reduce muy drásticamente Se ofrece una disponibilidad más temprana que facilita la evaluación del progreso	Los prototipos podrían suponer un aumento excesivo de la carga de trabajo En proyectos pequeños la viabilidad de la metodología es discutible, pues sería difícil la justificación del prototipado
Incremental [5]	Se simplifica al estar dividido en incrementos Se puede ver el producto antes de finalizarse	Si un incremento ya se ha comenzado sus requisitos no admiten modificaciones Se necesita establecer metas claras para poder ver el estado del proyecto
Espiral [6]	Desarrollo rápido y continuo por etapas Es factible añadir funcionalidad o cambios adicionales en cada etapa	Para un mejor funcionamiento hace falta seguir el protocolo del modelo estrictamente La existencia de capas intermedias hace que se genere más documentación
Desarrollo Rápido de Aplicaciones (RAD) [7]	Reutilización de componentes agiliza el desarrollo Permite finalización y entrega más veloz Ciclos de desarrollo de menor tamaño	Aumentan los problemas con el tamaño del proyecto Mayor dificultad para conocer el progreso Requiere participación activa de los usuarios

Tabla. 2.1: Comparativa entre modelos

De entre todos los modelos mostrados, el que parece más adecuado para nuestro proyecto es el incremental, debido a que en comparación con los modelos en cascada y en espiral se goza de más flexibilidad, aunque sea solamente entre incrementos. Por otro lado, las dimensiones del proyecto también consiguen que RAD y el modelo de prototipos pierdan bastante relevancia ya que es muy probable que apenas se tenga

la oportunidad de reutilizar código ni se cuente con el lujo de tener suficiente tiempo para realizar mucho prototipado.

2.4. Gestión y organización

Para poder organizar el tiempo que habrá que dedicar al desarrollo de la aplicación se intentará estimar el tiempo individual de cada una de las tareas y sus prioridades. Buscamos los menores esfuerzo y error posibles así que recurriremos a algunas estrategias ágiles, a pesar de no usar una metodología ágil. Las elegidas serán algunas de las más simples como tallas de camisetas [8] y *Fibonacci* [9], que consisten en asignar una talla (XS,S,M,L,X,XL) o un número de la sucesión de Fibonacci respectivamente para que sea visible el esfuerzo y el tamaño relativo para cada tarea.

Tarea de alto nivel	**Tareas**	**Tallas**	**Fibonacci**
Interfaz simple y fácil de usar	Hacer interfaz simple	S	8
	Implementar botones fáciles de entender	S	5
Toma de fotos de diagramas	Poder acceder a las imágenes del dispositivo	XS	3
	Poder tomar imágenes desde la cámara	S	5
Reconocimiento de elementos del diagrama	Reconocimiento de texto	XL	55
	Reconocimiento de figuras	XL	55
	Reconocimiento de relaciones entre figuras	L	34
Crear diagrama	Guardado del archivo generado	XS	3

Tabla. 2.2: Tareas y estimaciones del proyecto

2.5. Estimación de tamaño y esfuerzo del proyecto

A la hora de analizar el esfuerzo y el tamaño habrán de tenerse en cuenta que el tiempo del que se dispone es hasta la entrega del TFG y que el personal con el que contamos es únicamente el autor del mismo.
Dicho esto, se estimará el tamaño del proyecto haciendo uso de Puntos de función (o puntos de caso de uso) [10]. Modelos más matemáticos como COCOMO [11] se han ido descartando debido a la incertidumbre a la hora de estimar las líneas de código de la aplicación.

Término	Significado	Obtención
UCP	Puntos de Caso de Uso	UUCP X TCF X EF
UUCP	Puntos de Caso de Uso sin ajustar	UUCW + UAW
UUCW	Peso de Caso de Uso sin ajustar	Suma de complejidades por caso de uso[1]
UAW	Peso de Actor sin ajustar	Determinación del tipo de actor

Para realizar la estimación como tal, habrá que calcular los puntos de función (UCP) como $UCP = UUCP \times TCF \times EF$. Empezaremos calculando los puntos de caso de uso sin ajustar (UUCP), que están determinados por la suma de los pesos de caso de uso y de actor $UUCP = UUCW + UAW$

Función	Complejidad	Peso
Generar diagrama desde cámara	Alta	15
Generar diagrama desde galería	Alta	15

Tabla. 2.3: Cálculo de peso de casos de uso

Tipo de Actor	Ejemplo	Peso
Simple	Otro sistema a través de una API	1
Promedio	Otro sistema a través de un protocolo	2
Complejo	Una persona a través de interfaz gráfica	3

Tabla. 2.4: Cálculo de peso de actor

[1] Baja = 5, Media = 10, Alta = 15

Tenemos un UUCW de 30, y como solamente tenemos un actor en este caso, un usuario a través de interfaz gráfica, el UAW es 3. Ahora que tenemos un UUCP DE 33, solo necesitamos calcular los factores de ajuste TCF y EF asignando para cada uno valores de 0 a 5 para cada uno de los factores individuales. Las fórmulas que los determinan son $TCF = 0{,}6 + (0{,}01 \times \Sigma Impactos)$ y $EF = 1{,}4 + (-0{,}03 \times \Sigma Impactos)$ respectivamente.

Factor	**Peso**	**Valor**	**Impacto**
Sistema distribuido	2	0	0
Objetivos de rendimiento	2	2	4
Eficiencia de usuario final	1	3	3
Procesamiento interno complejo	1	5	5
Código reutilizable	1	1	1
Facilidad de instalación	0.5	4	2
Facilidad de uso	0.5	5	2.5
Portabilidad	2	1	2
Facilidad de cambio	1	1	1
Concurrencia	1	2	2
Objetivos especiales de seguridad	1	2	2
Acceso directo a terceros	1	0	0
Necesidad de entrenamiento especial	1	0	0
Total			24.5

Tabla. 2.5: Peso de los Factores Técnicos

$$TCF = 0{,}6 + (0{,}01 \times 24{,}5) = 0{,}845$$

Factor	Peso	Valor	Impacto
Familiaridad con el proceso de desarrollo	1.5	1	1.5
Experiencia en la aplicación	0.5	0	0
Experiencia en orientación a objetos	1	4	4
Capacidad del analista líder	0.5	1	0.5
Motivación	1	0	0
Estabilidad de los requerimientos	2	5	10
Personal a tiempo parcial	-1	3	-3
Dificultad del lenguaje de programación	-1	1	-1
Total			12

Tabla. 2.6: Peso de los Factores Ambientales

$$EF = 1{,}4 + (-0{,}03 \times 12) = 1{,}04$$

Una vez hemos calculado todo tendremos como resultado $UCP = 33 \times 0{,}845 \times 0{,}68 = 29{,}0004$. En cuanto al significado de los puntos de función, por sí solos no tienen, podríamos inferir que un proyecto con más UCP tiene muchas más probabilidades de necesitar más tiempo en llevarse a cabo.

Calcularemos ahora una aproximación del esfuerzo del proyecto, en cuanto a horas por persona. Usando los puntos de función, anteriormente se sugería usar 20 horas por punto, pero fue mejorándose y ajustándose con el tiempo.
Esta asignación varía dependiendo de los valores de los factores ambientales asociados previamente, han de contarse cuántos de los seis primeros tienen un valor menor a 3 y cuántos de los dos últimos tienen uno mayor. Con el resultado obtenido podemos comprobar las horas que necesitará una persona para cada punto de función en la siguiente tabla.

Horas/Persona	**Condición**
20	Valor ≤ 2
28	Valor ≤ 4
36	Valor ≥ 5

Tabla. 2.7: Horas por persona dependiendo de los Factores Ambientales

En el caso en el que nos encontramos es 4, por lo que una persona tardaría 28 horas en completar un UCP. Con lo que tenemos un esfuerzo total de $29{,}0004 \times 28 = 812{,}01$ horas de desarrollo.

2.6. Inventario y presupuesto

2.6.1. Estimación de recursos

- Especificaciones equipo informático:
  - CPU: AMD Ryzen 7 Pro 4700G
  - RAM: 16 GB
  - Almacenamiento: 1 TB HDD y 1 TB SSD
- Especificaciones terminal móvil:
  - Modelo: Xiaomi Mi A1
  - Procesador: Qualcomm Snapdragon 625 @ 2.0 GHz de ocho núcleos
  - RAM: 4 GB
  - Almacenamiento: 64 GB
- Software:
  - PlantUML
  - Android Studio
- Salario:
  - 6 horas diarias (suponiendo horario de estudiante)
  - 64 días laborales
- Servicios y alquiler:
  - Electricidad
  - Agua
  - Internet (Fibra Óptica 300 MB simétricos)

2.6.2. Estimación de costes

El tiempo de desarrollo será de unos 3 meses, con un total de 64 jornadas laborales (2.13 meses) al descontar los fines de semana. Procedemos ahora a calcular los gastos de amortización del equipamiento utilizado para el desarrollo del proyecto y los gastos de los servicios y suscripciones.

Concepto	Precio	Amortización	Mensualidad
Ordenador	1100€	5 años	18.33€
Teléfono Móvil	120€	4 años	2.5€
Total ((18.33 + 2.5) x 2.13 meses)			44.37€

Tabla. 2.8: Gastos amortizables

Concepto	Mensualidad	Total
Salario	1116.26€[2]	2377.64€
Servicios (Alquiler, electricidad, agua)	320€	681.60€
Internet	30€	63.90€
Total		3123.14€

Tabla. 2.9: Salarios y gastos

Esto nos deja con un precio total de 3167.51€.

[2]Sueldo de media jornada ya que el sueldo anual de un analista programador es de 26.790,31€ según el convenio colectivo publicado en [12]

Capítulo 3

Análisis de requisitos

Para asegurar un buen desarrollo de la aplicación y calidad en el producto final será necesario entender y analizar los requisitos que habrán de cumplirse. Incluso cuando el proyecto aparenta tener pocos requerimientos, facilita la distinción entre las características más importantes, así como establecer las restricciones que habrán de tenerse en cuenta a lo largo del desarrollo.

La aplicación funcionará de una manera bastante simple para el usuario, se le permitirá escoger si la imagen de la que se extraerá el diagrama mental se tomará directamente con la cámara del dispositivo o será una procedente del almacenamiento del mismo. A continuación la aplicación comenzará el proceso de digitalización en el que reconocerá y extraerá de la imagen la geometría, los textos y las conexiones que la constituyan. Para finalizar, el programa tomará todos los datos obtenidos y generará un nuevo archivo que contendrá el diagrama en el nuevo formato editable digitalmente.

3.1. Diagrama de casos de uso

En el caso de este proyecto, tenemos únicamente al usuario como actor, que será el que utilice las funciones de la aplicación.

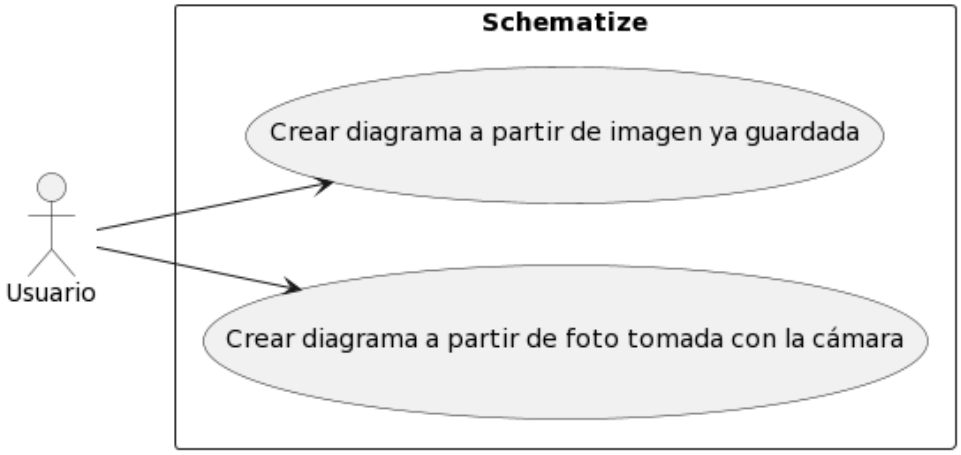


Figura 3.1: Diagrama de casos de uso

3.2. Requisitos funcionales

Código	Requisito funcional
RF-01	Crear diagrama a partir de imagen ya guardada
RF-02	Crear diagrama a partir de foto tomada de la cámara

Tabla. 3.1: Requisitos funcionales

Los requisitos funcionales se han extraído tratándose de manera parecida a casos de uso, pues en estos se muestran muy bien el comportamiento de cada uno [13].

El primer requisito funcional es el que permitirá la interacción en la que el usuario seleccionará una imagen ya guardada en el dispositivo para extraer el diagrama mental de ella. En el caso de no tener imágenes en el almacenamiento el usuario tendrá que cancelar la operación. También se muestra que si el diagrama deseado no es válido o correcto puede volver a intentarse de nuevo (con la misma imagen o alguna otra almacenada) hasta que se consiga.

RF-01	**Crear diagrama a partir de imagen ya guardada**
Actores	Usuario y Aplicación
Precondiciones	Ninguna
Poscondiciones	Ninguna
Escenario principal	1. El usuario selecciona la opción *Seleccionar Imagen* 2. El usuario selecciona la imagen de la que extraer el diagrama mental 3. La aplicación genera el diagrama reconocido en la imagen en formato editable 4. La aplicación informa que el diagrama ha sido creado y lo guarda 5. Fin del escenario con éxito
Escenarios alternativos	3a. La aplicación no consigue reconocer ni generar un diagrama 1. Regresar al paso 2 3b. La aplicación consigue un diagrama incorrecto 1. Regresar al paso 2
Excepciones	2a. El usuario no tiene imágenes guardadas 1. El usuario no puede continuar y cancela la acción 2. Fin del escenario sin éxito 2b. El usuario decide cancelar la acción 1. El usuario pulsa el botón *Cancelar* 2. Fin del escenario sin éxito

Tabla. 3.2: RF-01 Crear diagrama a partir de imagen ya guardada

El RF-02, segundo y último requisito funcional define la manera en la que el usuario será capaz de usar la aplicación también tomando la foto del dibujo del que extraer el diagrama directamente con la cámara del dispositivo. Del mismo modo que en el RF-01, el usuario podrá cancelar la acción cuando quiera y se podrá volver a intentar en caso de no obtener el diagrama deseado, solo que en este caso se reintentará tomando una foto de nuevo con la cámara. También se tienen en cuenta los posibles problemas que puedan ocurrir a la hora de abrir la cámara, ya que puede estar usándose por otras aplicaciones y podría darse la situación en la que sea necesario reiniciar la aplicación para solucionar el conflicto.

RF-02	**Crear diagrama a partir de foto tomada de la cámara**
Actores	Usuario y Aplicación
Precondiciones	Ninguna
Poscondiciones	Ninguna
Escenario principal	1. El usuario selecciona la opción *Tomar fotografía* 2. La aplicación abre la cámara del dispositivo 3. El usuario toma la foto del dibujo del que desee extraer el diagrama mental 4. La aplicación genera el diagrama reconocido en la imagen en formato editable 5. La aplicación informa que el diagrama ha sido creado y lo guarda 6. Fin del escenario con éxito
Escenarios alternativos	4a. La aplicación no consigue reconocer ni generar un diagrama 1. Regresar al paso 1 4b. La aplicación consigue un diagrama incorrecto 1. Regresar al paso 1
Excepciones	3a. La aplicación no consigue abrir la cámara del dispositivo 1. Fin del escenario sin éxito 3b. El usuario decide cancelar la acción 1. El usuario apaga la cámara del dispositivo 2. Fin del escenario sin éxito

Tabla. 3.3: RF-02 Crear diagrama a partir de foto tomada de la cámara

3.3. Requisitos no funcionales

En esta parte se encuentran los requisitos que no describen ni información que almacenar ni funciones que realizar, solamente características que definen el funcionamiento de la aplicación, por esto mismo también pueden conocerse como atributos de calidad.

1. La aplicación deberá tener una interfaz simple e intuitiva.

2. La aplicación deberá ser fácil de entender y de usar.

3. La aplicación podrá seguir funcionando en segundo plano en caso de necesitar un tiempo excesivo en algunas tareas.

4. La interfaz de la aplicación deberá adaptarse a la pantalla del dispositivo en el que se ejecute (*responsive design*).

5. La aplicación deberá tolerar errores y poder seguir funcionando y permitir intentos posteriores.

Capítulo 4

Diseño

Este capítulo se dedicará al diseño de software, etapa fundamental en el desarrollo de cualquier proyecto, pues la capacidad de cumplir los requisitos establecidos se verá directamente afectada dependiendo de lo bien que se realice.

Asegurarnos de dedicar suficiente tiempo al diseño será beneficioso, ya que un mayor esfuerzo invertido en esta etapa conllevará una disminución importante en los problemas que puedan surgir durante el desarrollo debidos a una insuficiente planificación. Además de diagramas de clases y de secuencia para aportar información sobre la solución que se ha modelado, también realizamos en este capítulo el diseño de la interfaz gráfica de la aplicación, en la que se busca claridad y simplicidad.

4.1. Diagrama de clases

Utilizando este diagrama se pretende facilitar la visualización de las clases que pertenecen al sistema y la manera en la que se relacionan.

Las clases se encuentran distribuidas intentando simular el orden en el que se utilizan en la aplicación. Comenzaríamos seleccionando si tomaremos la imagen de la Cámara o del Gestor de archivos, una vez tomada la imagen, cada Unidad de reconocimiento se encargará de realizar la detección y extracción del tipo de elemento que le corresponda y esto nos generará todos los elementos que necesitamos. Para finalizar, el Diagrama será simplemente una colección de Figuras, Textos y Conexiones.

Figura 4.1: Diagrama de clases

4.2. Diagramas de secuencia

La interacción entre el usuario y la aplicación es bastante simple de describir, tanto que podría dar la impresión de que no se dan los suficientes detalles o se ha invertido un esfuerzo insuficiente en la planificación. A pesar de esto sabemos que un exceso de diagramas puede convertirse en un inconveniente al incrementar la complejidad más de lo necesario.

Se han creado dos diagramas de secuencia para los dos casos de uso que englo-ban toda la funcionalidad de la aplicación.

En el primer diagrama de secuencia se muestra el orden en el que se lleva a cabo la operación de usar una imagen ya guardada de la que extraer el diagrama, bastará con seleccionar esta opción desde la interfaz inicial para que se abra el gestor de archivos del dispositivo y poder elegir la foto en cuestión. También se refleja el caso en el que no se tienen imágenes para seleccionar o simplemente el usuario decide no realizar la operación, que resultarán en la cancelación de la misma.

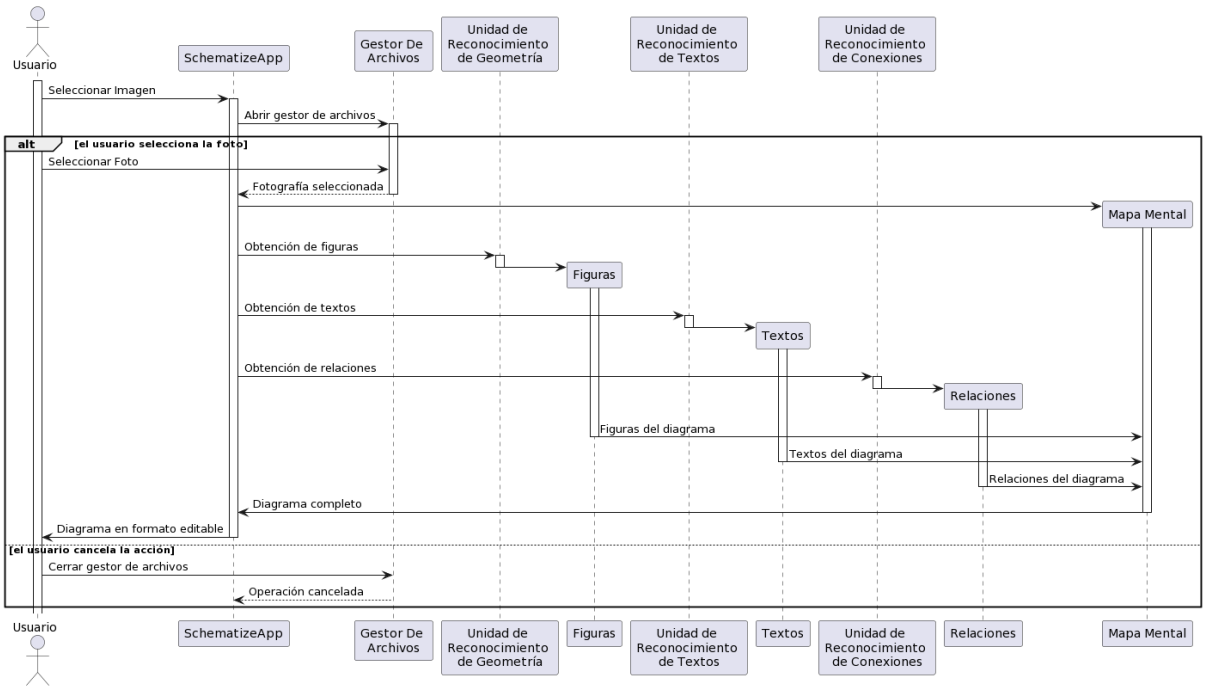


Figura 4.2: Diagrama de secuencia del caso de uso descrito en RF-01

El segundo y último diagrama es muy similar al primero, solo que seleccionar esta opción hará que lo que se abra sea la cámara del dispositivo. En este caso también se tiene en cuenta la posibilidad de que ocurra algún fallo que no permita la apertura de la cámara, resultando en no poder realizar la operación y finalizando la secuencia.

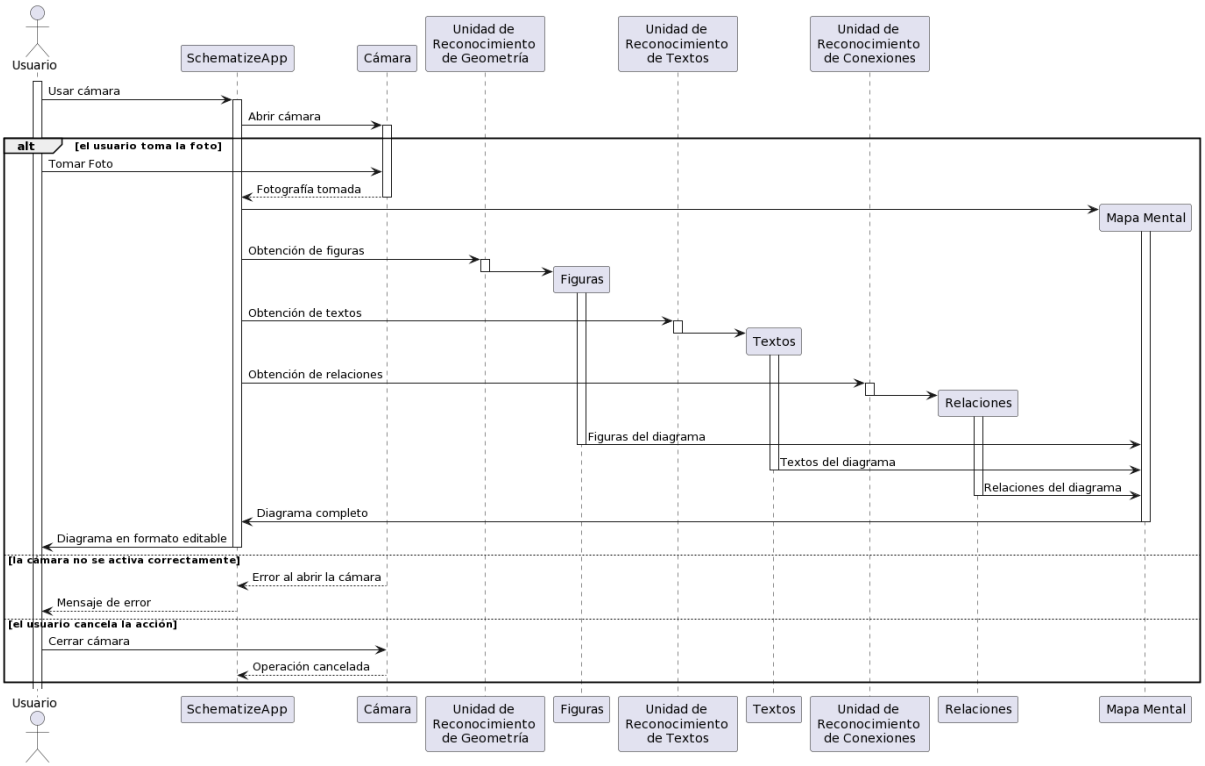


Figura 4.3: Diagrama de secuencia del caso de uso descrito en RF-02

4.3. Diseño de interfaz

Debido a necesitar poca interacción con el usuario, se ha optado por darle a la aplicación un diseño simple, una única pantalla en la que se tienen dos botones, uno para cada opción y una imagen que se sustituiría por la que elija el usuario para extraer el diagrama.

4.3.1. Wireframe

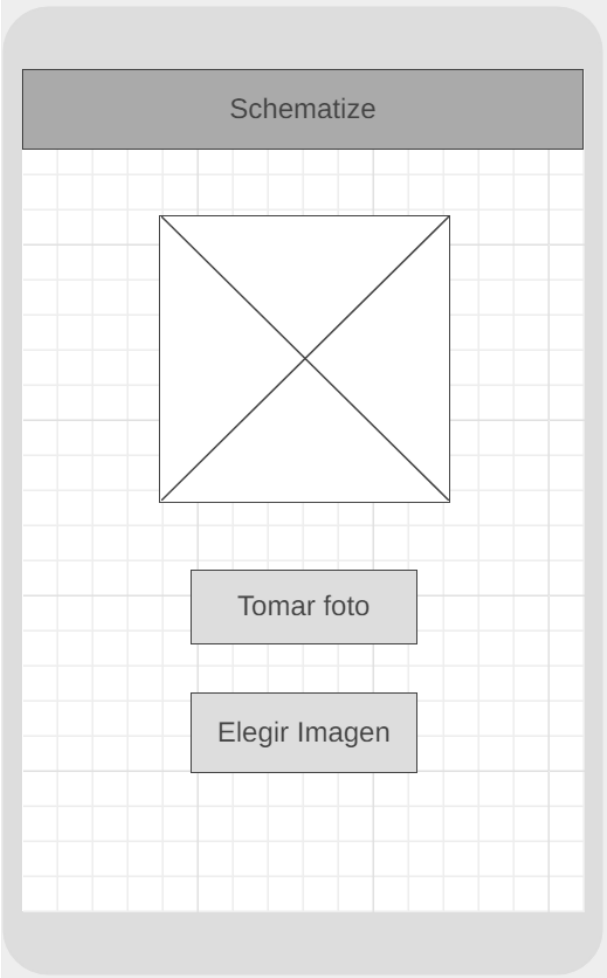


Figura 4.4: Wireframe

4.3.2. Prototipo

Figura 4.5: Prototipo

Capítulo 5

Desarrollo

En este capítulo será expuesto el proceso de implementación de la aplicación, así como las decisiones tomadas a lo largo del mismo. Se explicarán las herramientas que se puedan ajustar las necesidades de la aplicación y se justificarán las opciones que sean seleccionadas. Finalmente se analizará la implementación, dividida en secciones correspondientes a las partes que se han ido completando de manera secuencial.

5.1. Decisiones previas al desarrollo

Antes de comenzar a desarrollar la solución software, es necesario elegir la plataforma para la que se hará. No solo por las ventajas y las características de la plataforma en sí, sino también para poder seleccionar con más precisión las herramientas que se usarán a lo largo de todo el proceso de desarrollo. En la tabla 5.1, se muestran los aspectos que se han contrastado de varias plataformas, siendo estos:

- Capacidad computacional para un mejor rendimiento y disminución de tiempos de ejecución
- Necesidad de instalación de la aplicación
- Portabilidad, en sentido de poder hacer uso de la aplicación en distintos lugares
- Facilidad de actualizar la aplicación para el usuario, teniendo en cuenta las distintas posibilidades dentro de cada plataforma
- Experiencia del equipo de desarrollo en programas para la plataforma en cuestión

Ventaja	Aplicación Escritorio	Aplicación Móvil	Aplicación Web
Mejor rendimiento	✓	✗	✗
El usuario no requiere instalación	✗	✗	✓
Portabilidad	✗	✓	✓
Simplicidad de actualizaciones	✗	✓	✓
Conocimientos previos	✓	✗	✓

Tabla. 5.1: Comparación entre plataformas

Teniendo en cuenta estas comparaciones, desarrollar la solución como una aplicación web parece ser la opción más sensata. Sin embargo, nos decantamos por desarrollar una aplicación móvil, principalmente porque a día de hoy prácticamente todo el mundo (o al menos toda la gente que necesitaría hacer uso de la funcionalidad de la aplicación) tiene un teléfono móvil con cámara, lo que contribuye a dos puntos de gran importancia para el proyecto.

Por un lado estaría a disposición de cualquiera y esto ayudaría a que a la aplicación se le dé el uso casual (que pueda utilizarse en cuanto surja la necesidad o se le ocurra al usuario, sin que haga falta ningún tipo de planificación previa) que se pretende, y por otro lado el hecho de desarrollarse para dispositivos que por regla general tienen una cámara integrada permitirá su uso sin ninguna complicación ni necesidad de pasos adicionales como tener que importar la imagen desde el dispositivo desde el que se tomó la fotografía.

Se ha decidido desarrollar la solución para teléfonos móviles, sin embargo, no para todos, pues la aplicación estará destinada solo para su uso en Android, ya que es el que la gran mayoría de terminales[1] utiliza. Así que nos centraremos en la búsqueda de las mejores tecnologías y herramientas para el desarrollo en Android.

[1] 74 % en 2020.

5.2. Lenguaje, entorno y herramientas

Para seleccionar un lenguaje de programación para aplicaciones móviles, buscamos información para encontrar las mejores opciones posibles, de las cuales destacan Java, Kotlin y C++. El lenguaje que utilizará es Java, pues es en el que más experiencia tengo y el que más sencillo me parece, esto ahorraría el tiempo de aprender Kotlin (a pesar de ser muy parecidos) y el que podría generar la mayor complejidad de C++. El entorno de programación que se ha elegido es Android Studio, principalmente por la capacidad de crear dispositivos Android virtuales de manera nativa, ya que facilita enormemente la depuración y la realización de pruebas.

5.2.1. Herramientas necesarias

- **Reconocimiento de figuras**

  Se necesita poder dividir el diagrama en las distintas figuras que lo componen para tratarlas individualmente. De esta manera será más sencillo distinguir la forma de la figura y analizar el contenido de la misma.

- **Reconocimiento de relaciones**

  También es necesario que podamos encontrar y reconocer las líneas que unen los elementos del diagrama, es importante que no solo se capten las líneas como tal, también hemos de ser capaces de distinguir qué elementos son los que unen.

- **Reconocimiento de texto**

  Necesitamos algún tipo de motor de reconocimiento óptico para extraer el texto escrito en cada figura del diagrama,

- **Tratamiento de imagen**

  Para facilitar todo lo anterior hace falta poder tratar la imagen de la que se tomará el diagrama para minimizar el error del reconocimiento. Uno de los objetivos más importantes del tratamiento de la imagen es disminuir ruido, para hacer el color de fondo más uniforme y fácil de diferenciar del contenido escrito o dibujado a mano (texto, figuras y relaciones).

Opciones

- Tesseract OCR[14]:

  Motor de reconocimiento óptico de caracteres considerado de los más precisos entre los motores de código abierto de este tipo en 2006.

- ABBYY FineReader[15]:

  Programa de pago dedicado a extraer texto de archivos PDF e imágenes, no soporta caracteres escritos a mano, tiene API para Java entre otros lenguajes.

- Google Cloud Vision[16]:

  También de pago, herramienta en la nube de Google que permite analizar imágenes en busca de texto, reconocimiento de caras y de patrones.

- TensorFlow[17]:

  Biblioteca de código abierto para aprendizaje automático desarrollado por Google para la construcción y entrenamiento de redes neuronales para detección y análisis de patrones.

- OpenCV[18]:

  Biblioteca libre de visión artificial de mucha popularidad desarrollada por Intel. Puede aplicarse en diversas áreas como edición y tratamiento de imágenes, características 2D y 3D, reconocimiento de objetos y segmentación, entre otros más.

- BoofCV[19]:

  Biblioteca de código abierto desarrollada para visión por ordenador a tiempo real, muy alto rendimiento para tareas de alto nivel.

Selección y justificación

Para cubrir todas las necesidades de funcionalidad de la aplicación, se han seleccionado OpenCV y Tesseract OCR. OpenCV para el tratamiento de la imagen y el reconocimiento de las figuras y relaciones, y Tesseract para el reconocimiento de los textos y caracteres.

Se ha elegido Tesseract ante sus posibles alternativas principalmente por ser gratuito, TensorFlow también lo es pero que sea de aprendizaje automático le quita la simplicidad que preferimos con Tesseract. El resto de opciones se han desechado por

ser de pago, aunque tienen más razones, por ejemplo ABBYY FineReader no soporta letra escrita y Google Cloud Vision nos haría dependiente de la nube.

El principal motivo por el que se ha preferido OpenCV es la mayor cantidad de documentación respecto al resto de opciones, debido a su popularidad también es mucho más fácil encontrar ayuda incluso para cuestiones muy específicas. BoofCV sigue siendo muy buena opción, más rápido para rutinas de alto nivel pero peor para algoritmos más sencillos ([20]).

5.3. Implementación

El proceso de desarrollo puede ser segmentado en varias partes para mayor claridad y organización. Estas partes surgen de la secuencia de desarrollo que se planeó preliminarmente y con la que se comenzó a trabajar, cuyos pasos brevemente descritos son:

1. Crear aplicación de móvil que pueda tomar fotos desde la galería o la cámara, solicitando los permisos que requiera cuando sea necesario.
2. Primero detectar las figuras, para obtener el número de figuras y sus posiciones.
3. Una vez se pueden diferenciar entre cada figura, reconocer el tipo de figura.
4. Extraer de alguna manera las figuras de la imagen para poder reconocer las líneas que las conectan.
5. Identificar qué figuras conecta cada línea.
6. Aislar el interior de la figura para obtener mejores resultados con el reconocimiento de caracteres.
7. Crear y guardar el archivo que contendrá el diagrama con todos los datos obtenidos.

Este orden no requiere ser seguido a la perfección ni es totalmente descriptivo del ritmo que se llevó realmente, pues algunos pasos pueden intercambiarse sin ningún efecto y puede incluso trabajarse en distintas partes simultáneamente. Además se incluirán algunos pasos intermedios que contengan información relevante pero sin ser considerados parte de ninguna sección de las anteriores.

5.3.1. Aplicación base

Aunque la gran mayoría del tiempo de la implementación se ha dedicado a la extracción y el reconocimiento, se empezó por crear una versión de la aplicación que solo contendría la interfaz usable pero sin funcionalidad, de manera que sea lo único que quede por añadir.

Al crear el proyecto, primero se crea la distribución de los elementos de la interfaz, tenemos dos botones y una imagen. Después se les asignan funciones a los botones, el primero será para extraer el diagrama de una foto tomada con la cámara, por lo que la aplicación intentará abrir la cámara y pedirá permiso en caso de no tenerlo; el segundo botón hará lo mismo pero para la galería de fotos, pedirá permiso para acceder al almacenamiento del dispositivo y abrirá la galería para seleccionar la imagen.

Las imágenes a través de estas peticiones se obtienen en formato URI. Una vez la foto sea tomada o seleccionada, se mostrará en la aplicación sustituyendo la imagen (que estará vacía en caso de ser la primera vez que se utiliza) situada encima de los botones.

5.3.2. Pruebas OCR

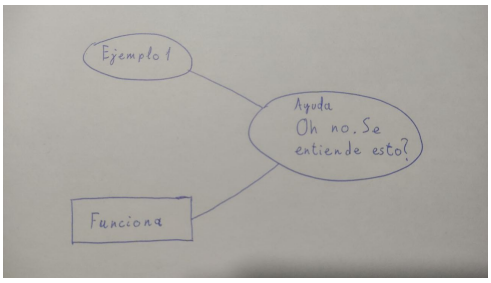
Antes de empezar con la extracción del diagrama, realizo pruebas con Tesseract, aprendiendo a instalarlo y a ver resultados con pequeños textos escritos a mano.

Haciendo esto aprendemos sus limitaciones y que para que funcione bien necesitamos solo texto, si existe algún contorno alrededor del texto arruina toda la lectura y no detecta el texto del interior. Gracias a estas pruebas tenemos más información para saber el tratamiento que debo darle a los fragmentos que recibirá Tesseract más adelante.

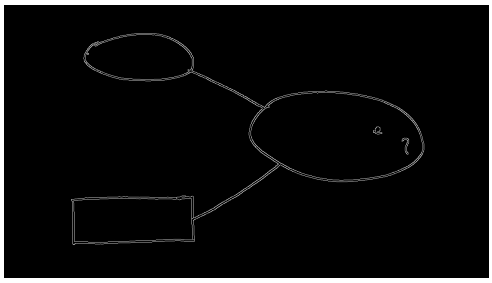
5.3.3. Detección de figuras

Para encontrar las figuras y poder distinguirlas, se utilizará el método `findContours` de OpenCV, el cual requiere (además de varios argumentos) que las figuras se vean de color blanco sobre un fondo negro, que es básicamente lo opuesto a lo que obtendríamos al escribir sobre una hoja de papel blanco, por lo que antes tratamos la imagen para realizar esta transformación.

Para ello utilizaremos operaciones de OpenCV como la erosión para eliminar ruido (así evitamos que manchas o arrugas de la hoja sean confundidos con elementos del diagrama) y el algoritmo `Canny` para detección de bordes que nos dará el resultado que queremos resaltando los contornos que detecte en blanco sobre un fondo negro tal y como se ve en la figura 5.1b.



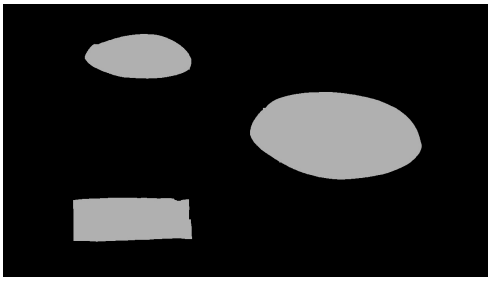
(a) Imagen sin editar



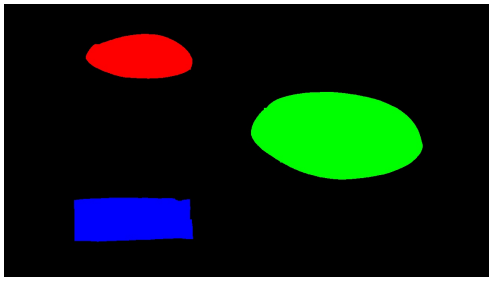
(b) Imagen tras aplicar el algoritmo detector de bordes Canny

Figura 5.1: Comparación después de haber realizado las primeras transformaciones a la imagen

Ahora que tenemos la imagen[2], realizamos una operación de erosión más para difuminar las líneas conectoras y podemos usar `findContours`, que tal y como explica el nombre, encontrará los contornos y almacenará cada uno en su propia matriz. Utilizando el método `drawContours` con la estructura que contiene estas matrices podemos ver los contornos que se han obtenido previamente, resultando en las figuras 5.2a y 5.2b.



(a) Contornos obtenidos



(b) Contornos de diferentes colores para mostrar que pueden dibujarse de manera independiente

Figura 5.2: Distinción de contornos

[2]En este punto y durante todo el desarrollo se trabaja con `Mat`, que son las matrices que utiliza OpenCV, pero seguiremos hablando de «fotos» e «imágenes» mientras no dificulte la comprensión

5.3.4. Reconocimiento y distinción de figuras

Podemos distinguir una figura de otra, pero necesitamos ser capaces de identificar la forma de cada una.

Esta aplicación solo distinguirá entre figuras elípticas y rectangulares, y en la aplicación esto lo realizamos creando un rectángulo del menor área posible que contenga a cada contorno (véase la figura 5.3) y calculando la relación entre el área del propio contorno y su rectángulo.

Si el resultado fuese 1 sería un rectángulo perfecto, pero teniendo en cuenta el error que pueda darse escribiendo en papel suponemos que tenemos un rectángulo cuando el resultado es superior a 0.9 (el área de la figura ocupa más del 90 % de su rectángulo contenedor) y una elipse cuando es inferior.

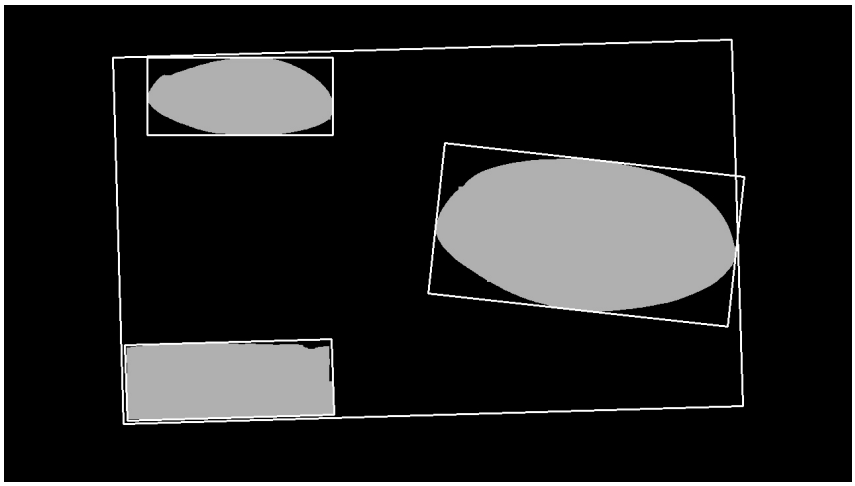


Figura 5.3: Rectángulos conteniendo cada contorno

5.3.5. Detección de líneas

Una vez tenemos las figuras, para obtener las líneas se utilizó `drawContours` de nuevo para dibujar los contornos sobre sí mismos de color negro, para así conseguir una imagen con únicamente las líneas que conectan los elementos, como en la figura 5.4a.

Hecho esto aplicamos el operador de dilatación de OpenCV, que sería el inverso de la erosión, para aumentar ligeramente el grosor y el tamaño de las líneas para la detección con `findContours` de nuevo, resultando en la figura 5.4b en este caso.

Con ello solo obtendremos el número de líneas, para identificar qué figuras conecta cada una, utilizaremos `connectedComponents`, otro método de OpenCV que dada una imagen con varios contornos devuelve otra con los píxeles siendo coloreados con el identificador del propio contorno (esencialmente pintando cada figura con un color distinto).

Ahora que tenemos las líneas en una imagen y las figuras identificadas en otra, lo que hacemos es buscar los píxeles en los que se solapan. Para cada una, se toman las coordenadas de los píxeles pertenecientes a la línea y se leen en la imagen con los identificadores para contar cuántos píxeles conectan con cada figura, lo normal será que solo se encuentren píxeles correspondientes a dos figuras por línea, pero en el caso de encontrar de más (algo que no se ha dado en las pruebas pero que podría darse si las figuras se dibujan muy cerca entre ellas) simplemente se tomarán las dos que más píxeles coincidan con la línea.

Al terminar habremos conseguido identificar todas las parejas de elementos que una cada relación.

(a) Imagen tras eliminar las figuras (b) Dilatación de las líneas

Figura 5.4: Obtención y tratamiento de relaciones

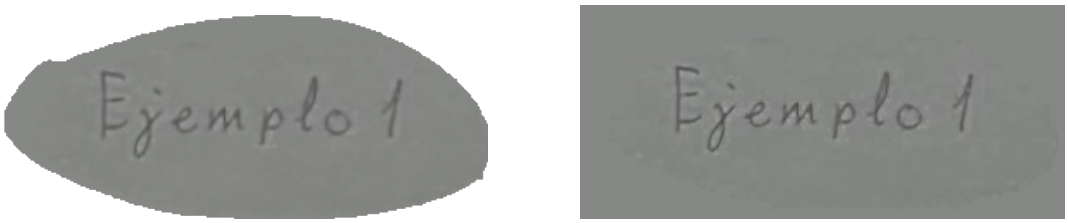
5.3.6. Reconocimiento de texto

Ya tenemos las figuras y sus conexiones, solo nos falta extraer el texto del interior de las figuras para obtener toda la información que necesitamos para crear el diagrama.

Recordemos que para que Tesseract extraiga el texto con más precisión necesitamos darle una imagen en la que solo se distingan letras sobre un fondo lo más uniforme posible. Para ello, teniendo almacenadas las coordenadas de cada píxel de cada contorno, podemos ”recortar” esos píxeles de la imagen original, para tener solo el interior de las figuras con el texto escrito.

Acto seguido realizamos otra erosión para eliminar los bordes de la figura, en este punto tendremos cada figura aislada en su propia imagen y el siguiente problema que nos encontramos es que los píxeles vacíos de estas imágenes son tomados como píxeles negros por Tesseract y al intentar reconocer texto arruinan la lectura.

Esto se soluciona binarizando la imagen (utilizando el operador de `threshold` de OpenCV) para hacer el fondo más uniforme y rellenando estos píxeles vacíos con el color de fondo resultante (véase la figura 5.5b como ejemplo), hecho esto se utiliza Tesseract para reconocer los caracteres de cada figura y se almacenan.



(a) Figura recortada (b) Imagen tras rellenar el fondo

Figura 5.5: Relleno de píxeles vacíos para Tesseract

5.3.7. Creación del diagrama

En este punto hemos conseguido extraer toda la información que necesitamos para el diagrama (posición, tamaño y forma de las figuras; número de líneas y las figuras que conectan y el texto contenido en cada figura) y solo falta crearlo.

El objetivo es que pueda abrirse y editarse desde la página diagrams.net, que acepta los formatos drawio, SVG, PNG, HTML y XML.

Realizando pruebas para ver la manera en la que se organizan los elementos en estos archivos descubrimos que el diagrama como tal se encuentra encriptado y encapsulado en un XML[22], buscando más información damos con una herramienta online[3] de los creadores de draw.io y la propia página que pretendemos usar para abrir los diagramas, utilizando esta herramienta podemos decodificar diagramas de prueba para aprender a crear cada tipo de elemento de los que contendrán los que crearemos nosotros.

Ahora que sabemos cómo construir los diagramas, lo que hará la aplicación es recorrer las figuras y las líneas almacenadas y crear los elementos del diagrama utilizando los datos que tome, en la figura 5.6 se muestra un ejemplo de esta conversión.

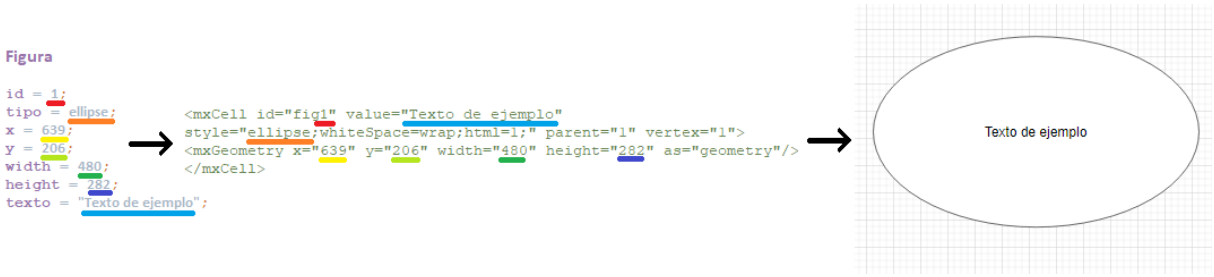


Figura 5.6: Creación de una elipse

No tenemos que preocuparnos de encriptar el diagrama porque diagrams.net los acepta también desencriptados (mientras no tenga errores el archivo) y de hecho se encargará de encriptarlos una vez se guarden. El archivo se guarda en formato drawio en el almacenamiento interno del terminal.

[3] https://jgraph.github.io/drawio-tools/tools/convert.html

Capítulo 6

Pruebas y resultados

Las imágenes que se han mostrado en el capítulo anterior surgían todas del proceso de digitalización del mismo diagrama hecho a mano, pues es sobre el que más se estuvo experimentando y realizando pruebas, pero no significa que sea el único ejemplo. En esta parte se incluirán todas las imágenes que se han probado con la aplicación así como capturas de pantalla de los archivos resultantes abiertos desde app.diagrams.net.

6.1. Ejemplo 1

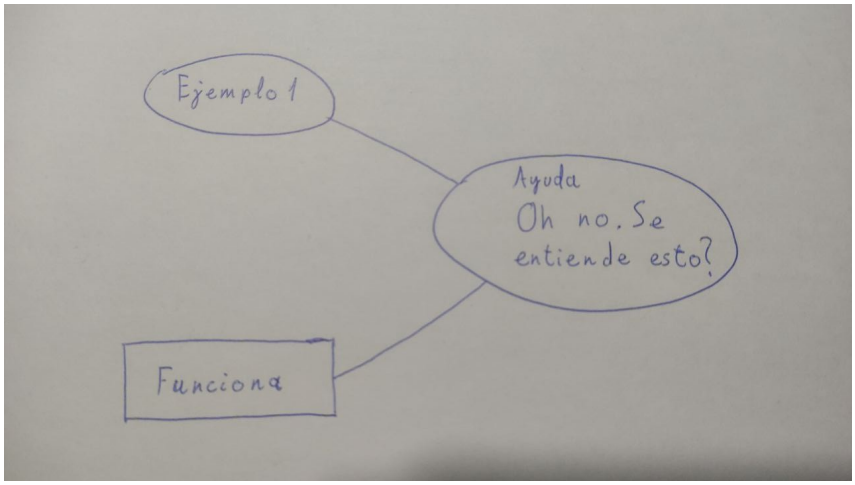


Figura 6.1: Prueba 1

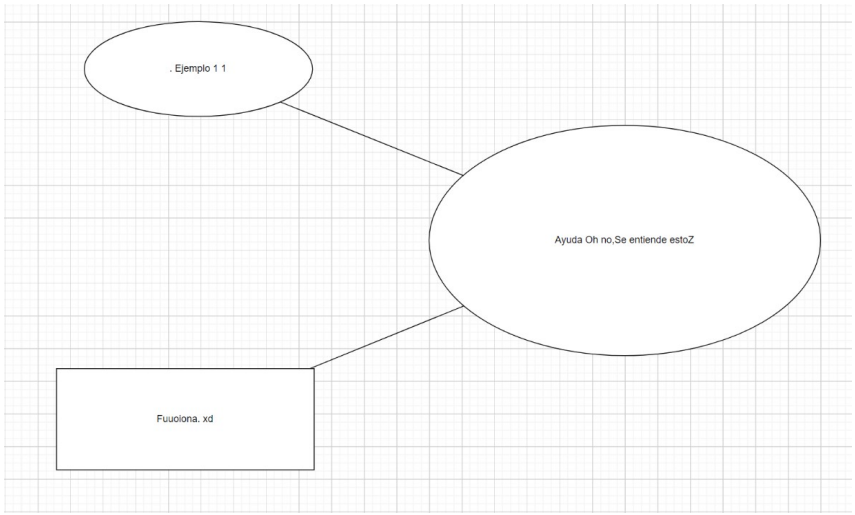


Figura 6.2: Resultado 1

El resultado que se puede ver en la figura 6.2 es bastante bueno, las figuras y conexiones se han extraído a la perfección, el texto se ha reconocido aceptablemente, con muy pocos fallos teniendo en cuenta mi mejorable caligrafía. Tiene sentido que este sea un buen resultado pues la figura 6.1 es la imagen que más se utilizó en la realización de pruebas a lo largo del desarrollo.

6.2. Ejemplo 2

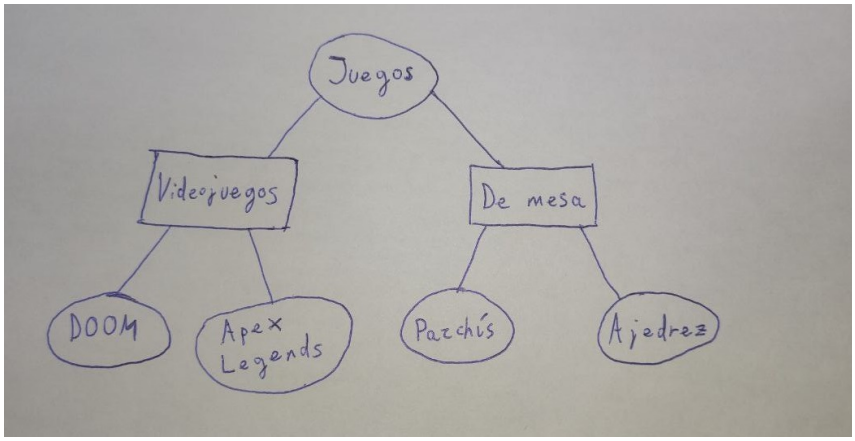


Figura 6.3: Prueba 2

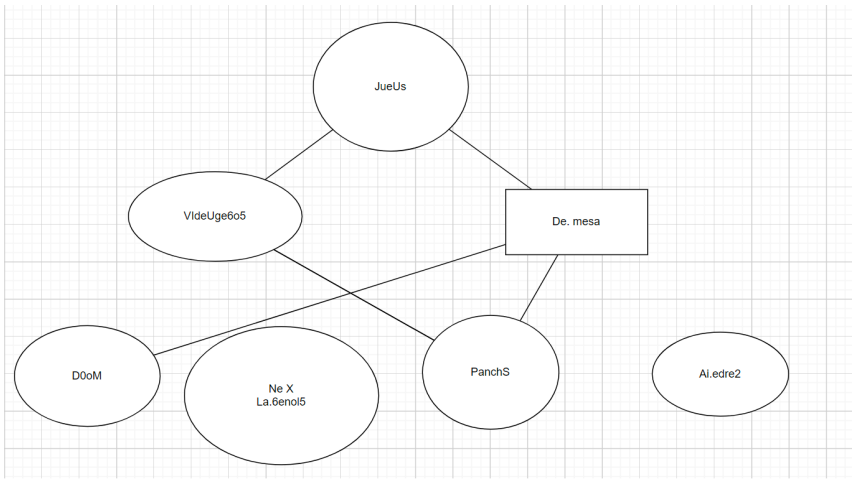


Figura 6.4: Resultado 2

En esta segunda descubrimos que la precisión disminuye conforme aumenta la complejidad del diagrama, como podemos ver en la figura 6.4, se obtienen todas las figuras, con el único error siendo una elipse que debería ser un rectángulo. Algunas líneas aparecen en sitios equivocados e incluso falta una, además el texto contiene muchos más errores. La caligrafía es la principal culpable de esto último y la incorrecta elipse, sin embargo, los fallos en las conexiones se dan cuando varias líneas se detectan como una sola, o cuando se segmentan debido a una erosión excesiva.

6.3. Ejemplo 3

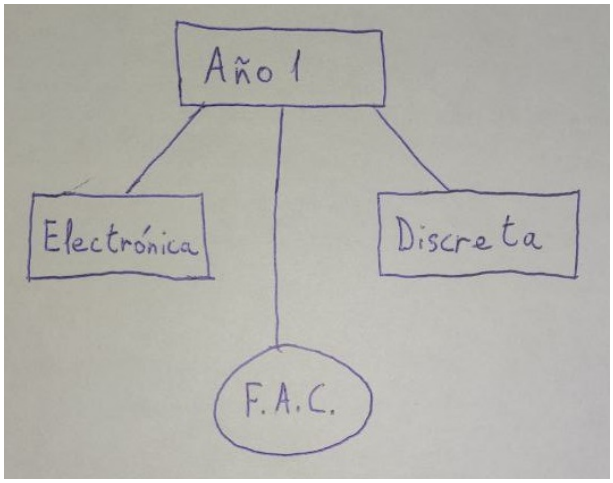


Figura 6.5: Prueba 3

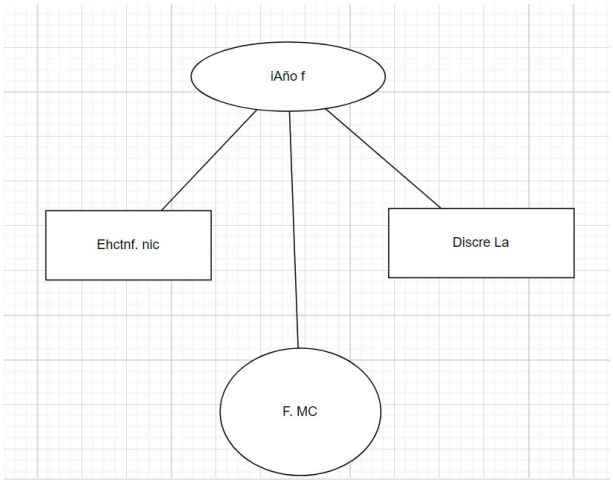


Figura 6.6: Resultado 3

En la figura 6.6 se observan todas las figuras y conexiones extraídas, las formas se han reconocido bien a excepción de una. No obstante, el texto no se ha reconocido muy bien, se captan todos los caracteres, pero algunos son confundidos debido al tipo de letra. En general es un resultado aceptable.

6.4. Ejemplo 4

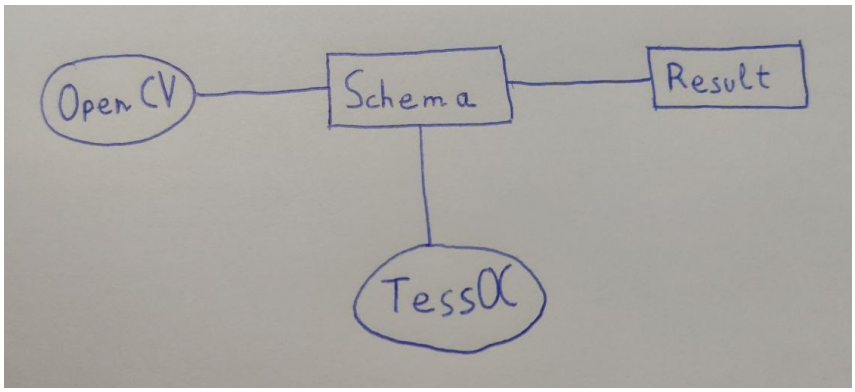


Figura 6.7: Prueba 4

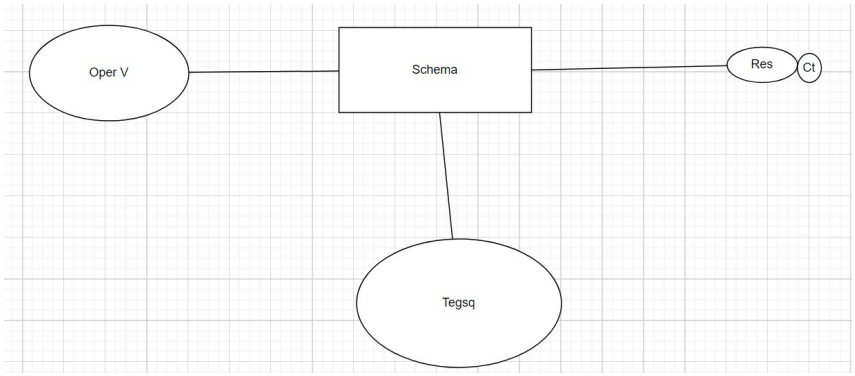


Figura 6.8: Resultado 4

En esta última prueba, el resultado es bastante parecido al anterior. Casi todas las figuras identificadas correctamente, a excepción de una que aparece segmentada como podemos observar en la figura 6.8. Aún así, las líneas se reconocen bien y también obtenemos un mejor resultado del texto, que se ha intentado hacer con mejor letra.

Capítulo 7

Conclusión y observaciones finales

Se ha terminado el desarrollo de la aplicación, gracias al cual se ha aprendido mucho. En este capítulo se expondrán las conclusiones a las que se ha llegado a lo largo del proyecto, así como las cosas que podrían mejorarse teniendo en cuenta todo lo que se ha ido descubriendo.

7.1. Conclusiones

Como conclusiones finales para el proyecto destacaría que subestimaba tanto la dificultad de llevar a cabo un proyecto completo como la utilidad de la organización y la planificación previa al desarrollo. A pesar de ya saber que es beneficioso elaborar un buen diseño, uno no es consciente de la magnitud de ese beneficio hasta que lo vive, pues conocer la complejidad estimada de cada tarea ha resultado ser mucho más útil de lo que parecía mientras se calculaban.

Por otro lado, no esperaba la poca disponibilidad de herramientas para el reconocimiento de letra escrita, lo cual supuso una pérdida importante de precisión, que junto con la limitación de tiempo me ha hecho imposible dar una aplicación perfecta. Sin embargo, estoy contento y satisfecho con el resultado final, aprender a usar las herramientas e ir completando las tareas que se iban proponiendo fueron experiencias bastante disfrutables.

7.2. Aspectos a mejorar

Como se ha mencionado antes, no existen muchas herramientas para el reconocimiento de caracteres escritos a mano, porque la mayoría están dedicadas para el reconocimiento de texto tipográfico (con fuentes como Arial o Times New Roman). Este es el motivo principal por el que la aplicación detecta mejor la buena caligrafía que se asemeja al texto escrito a máquina, y la manera en la que podría cambiarse esto y conseguir mejores resultados para cualquier tipo de caligrafía sería entrenando Tesseract con el tipo de letra del usuario en cuestión, pues simplemente habría que indicar a Tesseract que utilice el nuevo archivo de datos de entrenamiento que se habría generado.

Además, podría ampliarse la capacidad de generar diagramas más complejos y diversos con estas futuras mejoras:

- Reconocimiento y posterior creación de más formas, como por ejemplo rombos y triángulos.
- Reconocimiento de más tipos de conexiones como líneas con puntas de flecha, tanto uni como bidireccionales.
- Reconocimiento de relaciones más complejas, para que no solo resulten en líneas rectas.
- Añadir reconocimiento de colores para mantenerlos en el diagrama final.
- Ajustar el tamaño del texto dependiendo de la cantidad del mismo y del área de la figura que lo contenga.

Otra posible mejora para mayor comodidad sería que se preguntase al usuario si quiere abrir el diagrama tras crearlo, y en caso afirmativo que se redirigiese a `app.diagrams.net` con el nuevo archivo abierto.

Capítulo 8

Apéndices

A continuación se incluyen los manuales de instalación y uso de la aplicación.

8.1. Manual de instalación

Para instalar la aplicación tendremos que obtener el archivo APK (Android Application Package, formato que se usa para distribuir e instalar componentes empaquetados) del proyecto en el teléfono móvil en el que queramos utilizar la aplicación. Este archivo se genera desde el entorno de programación Android Studio, seleccionando la opción de ”Build APK” del proyecto.

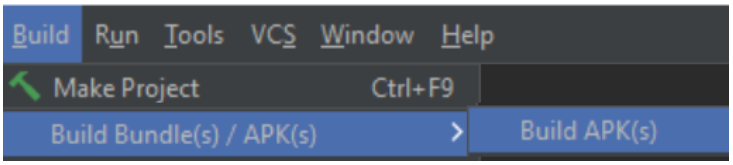


Figura 8.1: Localización de la opción de generación del archivo APK en Android Studio

Ahora que tenemos el archivo necesitamos enviarlo al teléfono, esto puede realizarse conectándolo por USB al ordenador y copiando el archivo al almacenamiento del móvil en cuestión o a través de plataformas como Google Drive, pudiendo acceder al archivo necesitando únicamente un enlace al mismo. Teniendo el archivo en el teléfono, solo habría que buscarlo en el almacenamiento (probablemente en la carpeta de ”Descargas” en el caso de usar Google Drive o en la que se haya guardado la copia mediante USB) y seleccionarlo para que se ejecute, tras confirmar la acción comenzará el proceso de instalación.

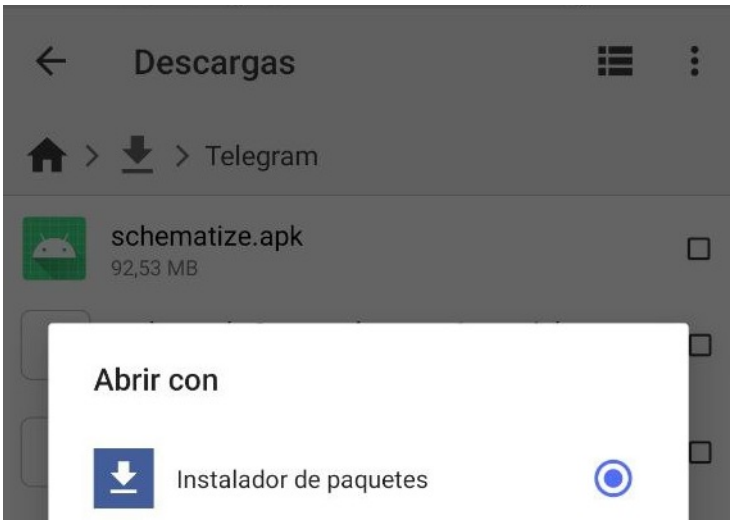


Figura 8.2: Ejecución del archivo APK desde el almacenamiento del teléfono

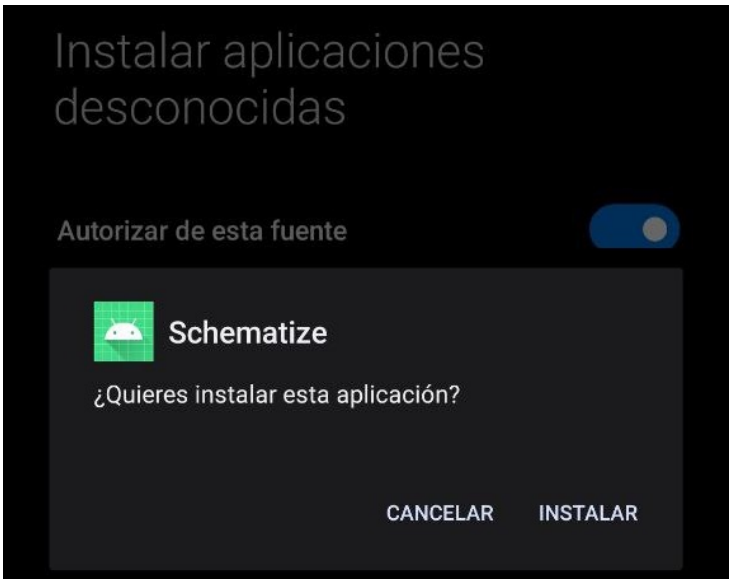


Figura 8.3: Confirmación de la instalación

Una vez instalada, la aplicación será accesible de la misma manera que el resto de programas del móvil, seleccionando su icono desde el menú de aplicaciones.

8.2. Manual de uso

La aplicación es tan fácil de usar como pulsar el botón para seleccionar la opción que queramos, si tomar la foto con la cámara (Figura 8.4) o seleccionarla de la galería (Figura 8.5), una vez se haya elegido la imagen se comenzará a procesar y digitalizar.

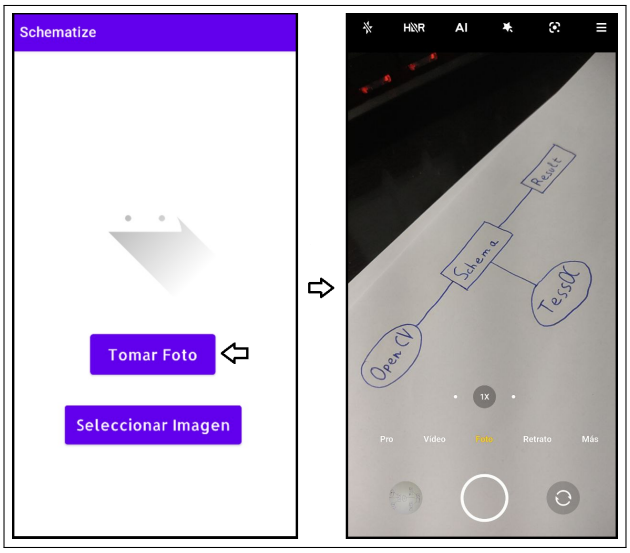


Figura 8.4: Primera opción

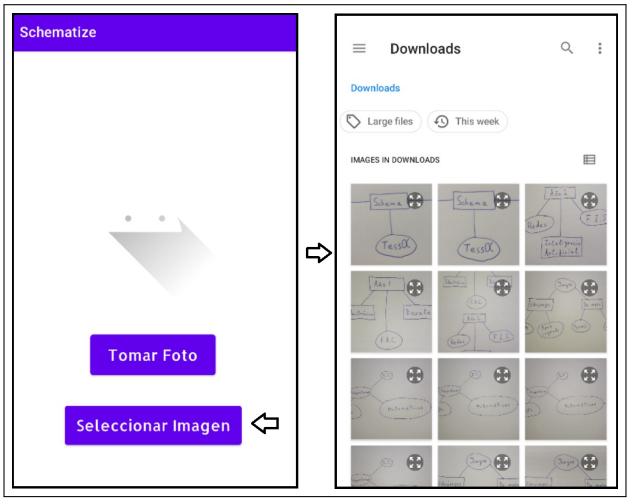


Figura 8.5: Segunda opción

Al terminarse este proceso, aparecerá un mensaje con la ruta del archivo creado, y para ver y poder editar el resultado obtenido solo hay que abrirlo desde `app.diagrams.net`.

Figura 8.6: Pantalla tras terminar la creación del diagrama

El diagrama habrá de estar hecho a lápiz o bolígrafo de color oscuro sobre una hoja blanca para que pueda reconocerse y extraerse. Para obtener resultados más precisos, se recomienda utilizar fotos con una buena iluminación que sea uniforme (que no haya partes que brillen más que las otras) y que el texto tenga una caligrafía lo más parecida a fuentes como Arial o tipografías similares.

Bibliografía

[1] Kent Beck. Manifiesto por el desarrollo Ágil de software, 2001. URL https://agilemanifesto.org/iso/es/manifesto.html.

[2] J. Schwaber, K. y Sutherland. La guía scrum. la guía definitiva de scrum: Las reglas del juego, 1991. URL https://www.scrum.org/resources/what-is-scrum.

[3] Oxagile. Waterfall software development model, 2014. URL https://www.oxagile.com/article/the-waterfall-model/.

[4] Todd Grimm. The human condition: A justification for rapid prototyping, 1998. URL http://www.tagrimm.com/publications/art-human-1998.html.

[5] Matthew Martin. Incremental model in sdlc: Use, advantage and disadvantage, 2022. URL https://www.guru99.com/what-is-incremental-model-in-sdlc-advantages-disadvantages.html.

[6] Barry Boehm. Spiral development: Experience, principles, and refinements, 2000. URL https://resources.sei.cmu.edu/asset_files/SpecialReport/2000_003_001_13655.pdf.

[7] Emmanuel Egeonu. The advantages and disadvantages of the rad model, 2022. URL https://distantjob.com/blog/rad-model-advantages-and-disadvantages/.

[8] Ankur Mistry. Agile story point estimation techniques - t-shirt sizing, 2017. URL https://www.c-sharpcorner.com/article/agile-story-point-estimation-techniques-t-shirt-sizing/.

[9] Julee Everett. Practical fibonacci: A beginner's guide to relative sizing, 2021. URL https://www.scrum.org/resources/blog/practical-fibonacci-beginners-guide-relative-sizing.

[10] Mike Cohn. Estimating with use case points, 2020. URL https://www.cs.cmu.edu/~jhm/DMS%202011/Presentations/Cohn%20-%20Estimating%20with%20Use%20Case%20Points_v2.pdf.

[11] Barry Boehm. *Software Engineering Economics*. Prentice-Hall, 1981. ISBN 978-0138221225.

[12] BOE. XVII Convenio colectivo estatal de empresas de consultoría, y estudios de mercados y de la opinión pública, 2018. URL https://www.boe.es/boe/dias/2018/03/06/pdfs/BOE-A-2018-3156.pdf. comprobado en 2021-08-08.

[13] Junta de Andalucía. Guía para la redacción de casos de uso. URL https://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/416.

[14] Izertis. Ocr en android, 2016. URL https://ahorasomos.izertis.com/solidgear/ocr-en-android/.

[15] Milosz Kmieciak y Tomasz Parkola Marcin Helinski. Report on the comparison of tesseract and abbyy finereader ocr engines, 2012. URL https://www.digitisation.eu/fileadmin/Tool_Training_Materials/Abbyy/PSNC_Tesseract-FineReader-report.pdf.

[16] Volodymyr Holomb. Side-by-side ocr in python with google vision and tesseract, 2021. URL https://towardsdatascience.com/side-by-side-ocr-in-python-with-google-vision-and-tesseract-66021d5702a0.

[17] Por qué tensorflow, 2015. URL https://www.tensorflow.org.

[18] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. .º'Reilly Media, Inc.", 2008.

[19] BoofCV. Página principal de boofcv. URL http://boofcv.org/.

[20] BoofCV. Comparativa de rendimiento entre opencv y boofcv, 2019. URL https://boofcv.org/index.php?title=Performance:OpenCV:BoofCV. comprobado en 2022-06-05.

[21] Wikipedia. Shape factor (image analysis and microscopy). URL https://en.wikipedia.org/wiki/Shape_factor_(image_analysis_and_microscopy).

[22] FileFormat. Drawio file format. URL https://docs.fileformat.com/web/drawio.