Taller de Proyecto II

Informe Final

09/12/2024

Generador de Números Aleatorios Basado en Patrones de Imagen y Temperatura

Grupo de Desarrollo

Pérez, Francisco – 02239/8

Cabrera, Merlina – 02240/0

Gallo, Francisco – 02008/3

Bosatta, Darío – 02338/0

1. Descripción General del Proyecto

1.1. Introducción

En el ámbito de la seguridad informática y cifrado de datos, la generación de números aleatorios es una tarea fundamental para garantizar la privacidad y protección en sistemas críticos, y se puede hacer uso de estos en variedad de aplicaciones. Sin embargo, los algoritmos tradicionales suelen ser predecibles, lo que compromete su seguridad. Este proyecto aborda este desafío desarrollando un generador de números aleatorios basado en datos impredecibles provenientes de una cámara web y un sensor de temperatura.

El sistema captura patrones visuales generados mediante una aplicación Android y mediciones ambientales tomadas con un sensor DHT11. Ambos datos se combinan para formar una semilla aleatoria, procesada mediante el algoritmo criptográfico SHA-256 para obtener claves seguras. El sistema también incluye una aplicación web para visualizar y gestionar estos datos mediante tecnologías modernas como Node.js, React y MySQL.

1.2. Objetivo

El objetivo principal de este proyecto es desarrollar un sistema capaz de generar claves aleatorias utilizando patrones visuales y datos de temperatura ambiental. Esto se logra mediante la integración de hardware y software especializados para capturar, procesar y almacenar datos, garantizando la máxima aleatoriedad posible y seguridad en los resultados generados. El proyecto busca ofrecer una solución robusta aplicable en entornos de seguridad digital y criptografía.

1.2.1. Descripción breve del sistema

El sistema utiliza una Raspberry Pi como módulo central, conectada a un sensor de temperatura DHT11 para obtener datos ambientales y a una cámara web para capturar imágenes en tiempo real. Estos datos son utilizados para generar una semilla aleatoria única, que pasará por un algoritmo para obtener un hash de 256 bits que sirve como clave aleatoria.

Los datos procesados se transmiten a un servidor web mediante el protocolo HTTP. La interfaz web, desarrollada en React, permite a los usuarios visualizar y gestionar las claves generadas. La información es almacenada en una base de datos SQL para poder verificar la aleatoriedad y reproducir simulaciones.

El sistema garantiza la privacidad de los datos mediante cifrado durante la transmisión y almacenamiento, y se implementan controles de acceso para restringir la generación y visualización de claves.

1.2.2. Requerimientos

Requerimientos de alimentación,

- Raspberry Pi 3B +: Se requiere una fuente de alimentación estable de 5V/2.5A.
- Sensor DHT11: Alimentado directamente desde los pines GPIO de la RPI, que proporciona tensión de 3.3V a 5V.

Requerimientos funcionales,

- El sensor DHT11 debe registrar datos de temperatura en tiempo real, mientras que la cámara web captura imágenes que contienen patrones visuales para generar la semilla.
- Los datos del sensor y de la cámara se concatenan para generar una clave aleatoria mediante SHA-256. Los usuarios deben poder generar estas claves a demanda desde la interfaz web.
- El sistema transmite los datos procesados desde la RPI a un servidor web, donde se gestionan y almacenan.
- Los usuarios deben poder visualizar las claves generadas y monitorear los datos utilizados para su creación a través de una interfaz web desarrollada en React.

Requerimientos no funcionales,

- Los datos transmitidos entre la RPI y el servidor web deben estar cifrados.
- El sistema debe ser capaz de manejar errores de captura de datos o fallos en la transmisión sin perder información crítica.
- El sistema debe procesar y transmitir los datos sin un retardo significativo.
- La Raspberry Pi y los dispositivos conectados deben operar de manera eficiente, optimizando el consumo de energía.

2. Presentación Esquemática del Proyecto

En principio se ilustra un diagrama general del sistema en la ***Figura 2.1***,

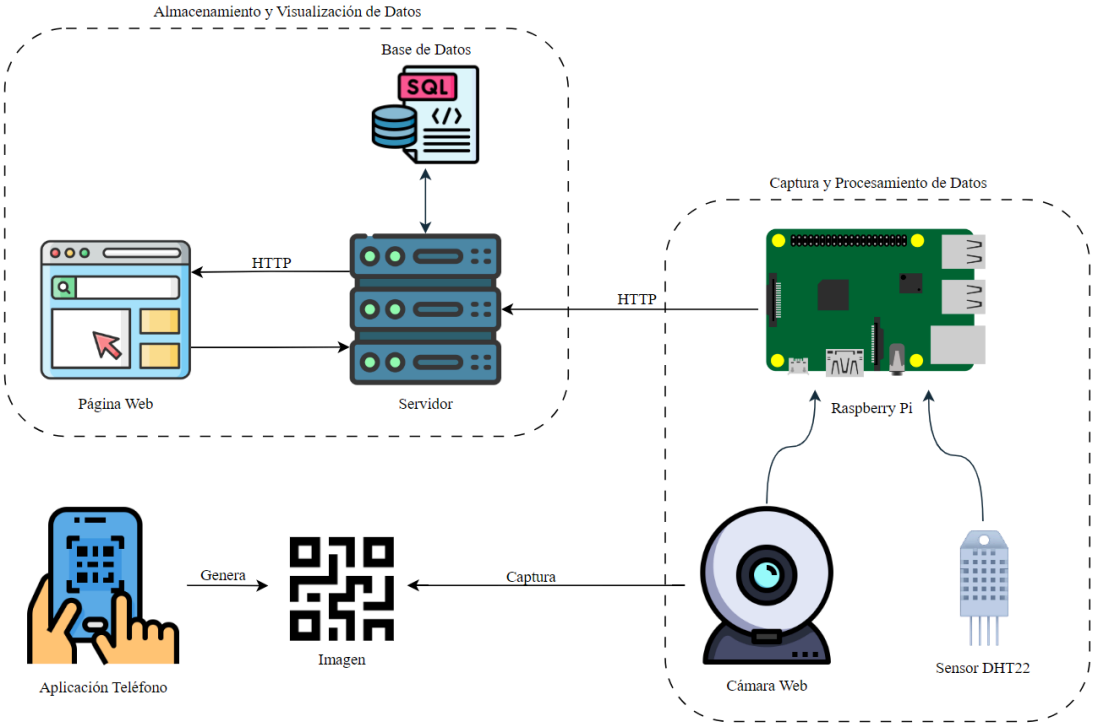

Figura 2.1. *Esquema general del sistema.*

A partir del esquema general del proyecto, se logran identificar dos grandes secciones a profundizar,

- **Captura y Procesamiento de Datos**

En ***Figura 2.2*** se muestra un esquema de conexión de la cámara web y el sensor DHT11 a la Raspberry Pi, que posibilita la captura de datos para su posterior procesamiento.

Por otro lado, en ***Figura 2.3*** se ilustra un diagrama de flujo del proceso de extracción de datos de los sensores para su procesamiento y generación de clave aleatoria.

- **Almacenamiento y Visualización de Datos**

Una vez procesados los datos capturados por la Raspberry Pi, se debe asegurar su persistencia y presentación al usuario. En ***Figura 2.4*** se visualiza un diagrama de flujo del sistema de almacenamiento y visualización de datos.

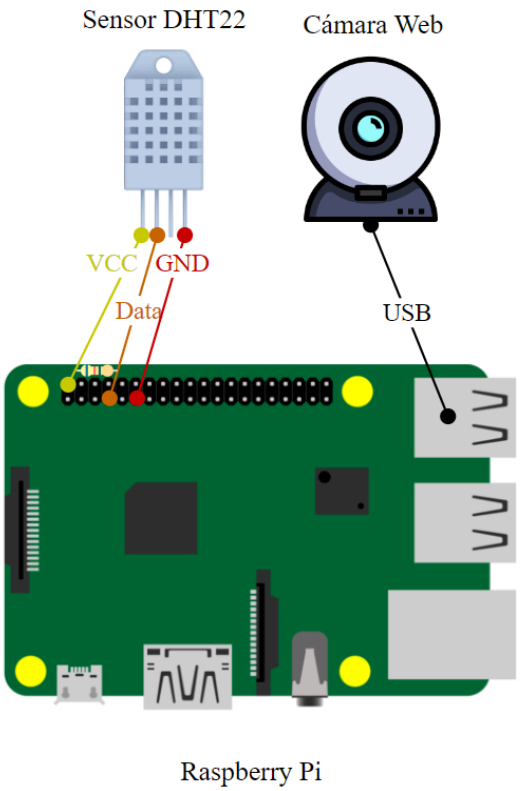

Figura 2.2. *Esquema de la conexión entre RPI, sensor y cámara.*

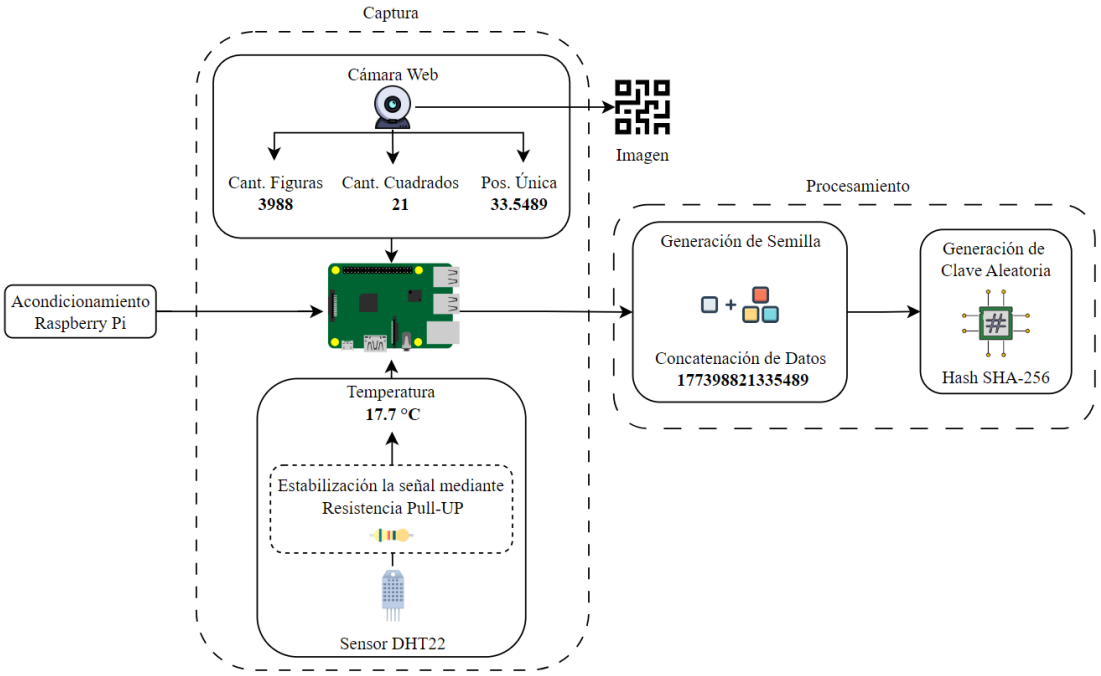

Figura 2.3. *Diagrama de flujo de captura y procesamiento de datos.*

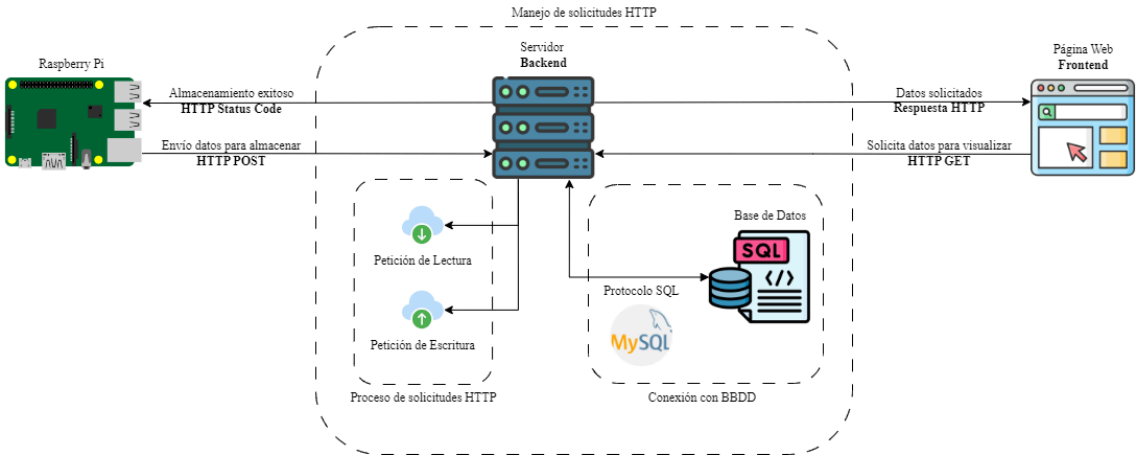

***Figura 2.4**. Diagrama de flujo del almacenamiento y visualización de datos.*

3. Documentación del Software del Proyecto

Como se observa en el esquema general del sistema, ***Figura 2.1***, este puede englobarse en dos grandes secciones, las cuales se describen en esta sección.

3.1. Captura y Procesamiento de Datos

Este apartado detalla el sistema de captura de datos desde los sensores y la generación de la clave aleatoria a partir de estos. Inicialmente, se acondicionó y programó la Raspberry Pi que actúa como base del sistema, para luego realizar las conexiones necesarias de hardware.

3.1.1. Configuración de Raspberry Pi y Conexiones

Se configuró desde cero la RPI 3 Model B+, cuyas características principales son las siguientes,

- **Procesador**: ARM Cortex-A53 de 64 bits @ 1,4 GHz.
- **Memoria RAM**: DDR2 de 1GB.
- **Conectividad**: Módulos integrados de Bluetooth y Wi-Fi.
- **Puertos**: Cuatro puertos USB 2.0.

Instalación de Sistema Operativo

Para instalar el sistema operativo, se conectaron un monitor mediante HDMI y un mouse y un teclado mediante los puertos USB de la Raspberry Pi. Además, se utilizó un dispositivo de almacenamiento, en este caso, una memoria, donde se grabó previamente el SO. Este fue conectado a la RPI mediante USB para facilitar la instalación.

El sistema operativo elegido fue “Raspbian”, que es una distribución oficial basada en Debian, optimizada para Raspberry Pi. Ofrece una interfaz gráfica ligera y herramientas para el desarrollo y control de proyectos, lo que facilita la programación y gestión de dispositivos conectados.

Una vez finalizada la instalación del SO y configurada la interfaz gráfica, se procedió a conectar los componentes necesarios para realizar pruebas funcionales del sistema. Los dispositivos conectados,

- Cámara web: Se conecta a uno de los puertos USB disponibles.
- Sensor DHT11: Se conecta a los pines GPIO mediante jumpers macho-macho,
    - Pin VCC: Conectado al pin 5V para la alimentación del sensor.
    - Pin de datos: Conectado al GPIO4 para la transmisión de datos.
    - Pin GND: Conectado a tierra para completar el circuito.

En la ***Figura 4.1*** se puede observar la Raspberry Pi con los componentes de hardware principales conectados. A su vez, se muestra gráficamente el esquema de conexiones de la ***Figura 2.2***.

3.1.2. Captura de Datos

3.1.2.1. Sensor DHT11

El DHT11 es un sensor que mide la temperatura ambiente, proporcionando datos precisos en tiempo real. Este es ampliamente usado en proyectos de esta índole debido a su precisión y bajo consumo energético,

- Es capaz de funcionar con tensiones de 3.3 y 5V, que le permite trabajar de manera estable cuando se conecta a los pines GPIO de la Raspberry Pi.
- En términos de corriente de operación, el sensor consume un máximo de 2.5mA, lo cual es significativamente bajo y puede ser fácilmente soportado por la RPI sin necesidad de fuentes de alimentación adicionales.

En el desarrollo del proyecto, se conectó el sensor a la Raspberry Pi y se configuró el entorno de desarrollo en Python, instalando las bibliotecas necesarias para capturar datos. Inicialmente, se consideró utilizar la biblioteca “Adafruit DHT”, ampliamente reconocida y utilizada para trabajar con sensores DHT y manejar los puertos GPIO de manera eficiente. Sin embargo, durante su instalación surgieron problemas relacionados con dependencias y restricciones de paquetes en el sistema operativo utilizado, lo que en un primer momento impidió su uso.

Dado este inconveniente, se optó por manejar manualmente la lectura de datos del sensor utilizando la biblioteca “RPi.GPIO”, junto con el paquete “dht11" para gestionar el sensor. Este enfoque permitió avanzar con las pruebas iniciales, logrando lecturas consistentes de temperatura y humedad.

No obstante, tras resolver las limitaciones iniciales y configurar adecuadamente el sistema operativo, se logró instalar con éxito la biblioteca previamente mencionada, la cual ofrece una interfaz más sencilla y optimizada para trabajar con el sensor. Debido a su robustez y facilidad de uso, se decidió finalmente reemplazar las configuraciones manuales por el uso de esta biblioteca, simplificando significativamente el desarrollo del código y mejorando la estabilidad en las lecturas.

Los resultados de las lecturas fueron satisfactorios en cuanto a consistencia y precisión, pero se observó un porcentaje recurrente de errores de lectura durante las mediciones. Estos errores ocurren cuando el sensor no responde adecuadamente a la solicitud de los datos, lo que puede deberse a factores como,

- Interferencias en la señal de los pines GPIO.
- Condiciones de alimentación insuficientes que afectan la estabilidad del sensor.
- Frecuencia de consulta excesiva, ya que el DHT11 necesita un tiempo de espera mínimo entre cada solicitud de datos.

Para abordar esta limitación y estabilizar las mediciones de temperatura se optó por estabilizar la señal mediante la integración de una resistencia pull-up al circuito eléctrico. La razón de esta elección, consiste en que el sensor DHT11 utiliza un bus de datos unidireccional, donde tanto el microcontrolador como el sensor comparten una única línea para enviar y recibir datos. Este tipo de comunicación es eficiente, pero presenta ciertos desafíos cuando no se implementan medidas adecuadas de estabilización. Sin una resistencia pull-up en la línea de datos, se puede obtener resultados no deseados como,

- **Estado Flotante**: Cuando ninguno de los dispositivos está transmitiendo datos, la línea de datos puede quedar en un estado “flotante” o indeterminado, lo que genera señales falsas o interferencias en la comunicación.
- **Inestabilidad en las Lecturas**: Esto puede llevar a lecturas erróneas o fallos recurrentes en la transmisión de datos entre el sensor y la Raspberry Pi.

La resistencia pull-up conecta la línea de datos al voltaje de alimentación, en ***Figura 2.2*** se identifica como cable “VCC”. Esto asegura que la línea de datos permanezca en un estado alto o “1” lógico cuando no hay transmisión activa. Al añadir esta resistencia en el circuito, se estabiliza la señal en la línea de datos reduciendo significativamente los errores en la lectura y también se minimiza la sensibilidad a interferencias externas, lo que mejoró la confiabilidad general del sistema.

Sin embargo, la resistencia pull-up no fue la primera opción para contrarrestar los errores de lectura, inicialmente se implementó un filtro basado en promedio móvil en el código. Este enfoque suavizaba las fluctuaciones en los datos al calcular el promedio de las últimas 10 lecturas, descartando valores atípicos. Aunque esta solución mejoraba la estabilidad de los datos, presentaba algunas limitaciones,

- **Latencia**: Las lecturas de temperatura no eran inmediatas, ya que el promedio se basaba en un conjunto acumulado de valores previos.
- **Dependencia del Software**: Se añadía complejidad al código, lo que aumentaba el tiempo de procesamiento y mantenibilidad al sistema.

Con la incorporación de la resistencia pull-up al circuito eléctrico, los errores de lectura disminuyeron drásticamente, haciendo innecesaria la implementación del promedio móvil en el código. Esto permitió obtener lecturas inmediatas y sin latencia, mejorando la capacidad de respuesta del sistema, y simplificar el código y reducir el tiempo de procesamiento en la Raspberry Pi.

3.1.2.2. Cámara Web

La cámara utilizada es Logitech C270, cuyo consumo coincide con las características estándar de un puerto USB. Funciona con 5V de entrada, y no supera los 500mA. Esta se encarga de capturar imágenes en tiempo real.

Se desarrollaron dos algoritmos utilizando Python: El primero se encarga de generar una imagen con un modelo visual específico que la cámara web puede reconocer fácilmente, y el segundo está diseñado para identificar y analizar dicho modelo. Con este enfoque, se busca facilitar la extracción de datos relevantes para su uso en la generación de claves aleatorias, asegurando que las imágenes contengan patrones claros y consistentes, optimizando la precisión del sistema.

Programa Generador de Imágenes

En cuanto al primer algoritmo mencionado, se trata de una aplicación desarrollada para la plataforma Android. El objetivo de este algoritmo es crear imágenes con un formato que sea fácilmente reconocido por el sistema de detección vinculado a la cámara web, optimizando así la captura de datos para la generación de claves aleatorias.

- Algoritmo de Generación de Patrón – Versión 1

En esta primera versión del algoritmo, se optó por generar imágenes con un patrón aleatorio, variando los colores que lo componen. Una vez iniciada la aplicación, el usuario puede ingresar manualmente las dimensiones del patrón, así como los colores que compondrán la imagen. Además, la plantilla generada incluye recuadros de color verde en sus esquinas, los cuales sirven como puntos de referencia para ayudar al algoritmo de detección a identificar y delimitar el área de interés de manera eficiente. La ***Figura 4.2*** ilustra cómo se presentan estos recuadros en las esquinas.

Sin embargo, la variabilidad en las condiciones lumínicas causó dificultades para la correcta detección de los colores, incluso con los filtros aplicados durante el análisis. Además, la imposibilidad de ajustar el nivel de exposición automática de la cámara web contribuyó a estos problemas. Debido a estas limitaciones, se procedió a modificar el algoritmo para generar patrones específicos.

- Algoritmo de Generación de Patrón – Versión 2

La segunda versión del algoritmo mantiene las mismas bases que la primera, pero introduce nuevas características para mejorar la flexibilidad y reconocimiento de las imágenes generadas. Al igual que la versión anterior, esta permite al usuario definir tanto el alto como el ancho de la imagen (en píxeles), ahora mediante controles deslizantes, también incluye un botón que inicia el proceso de creación de la imágen con los valores elegidos, ***Figura 3.1***.

Al presionar el botón, se crea un lienzo del tamaño especificado y de color blanco, añadiendo márgenes calculados dinámicamente para garantizar el espacio suficiente para las figuras, cuya cantidad se determina mediante un valor pseudoaleatorio relativo al tamaño de la imagen y su posición en el lienzo se define a partir de las figuras "vecinas" ya puestas en el mismo. Esta operación se define pixel por pixel. Adicionalmente se agrega los recuadros verdes en las esquinas de los márgenes. A medida que los mismos son "pintados", también se marca el área cuadrada que ocupa la figura con un margen adicional para asegurar la no superposición de las mismas. Esto es de vital importancia ya que las figuras deben ser claramente reconocibles por el escáner. ***Figura 3.2.***

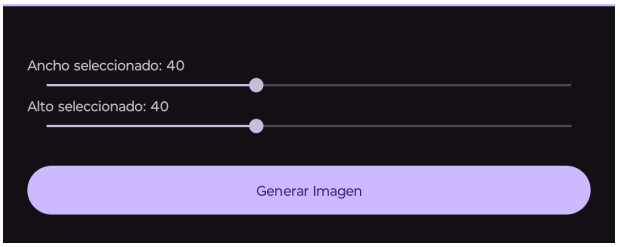

Figura 3.1. *Interfaz de la aplicación.*

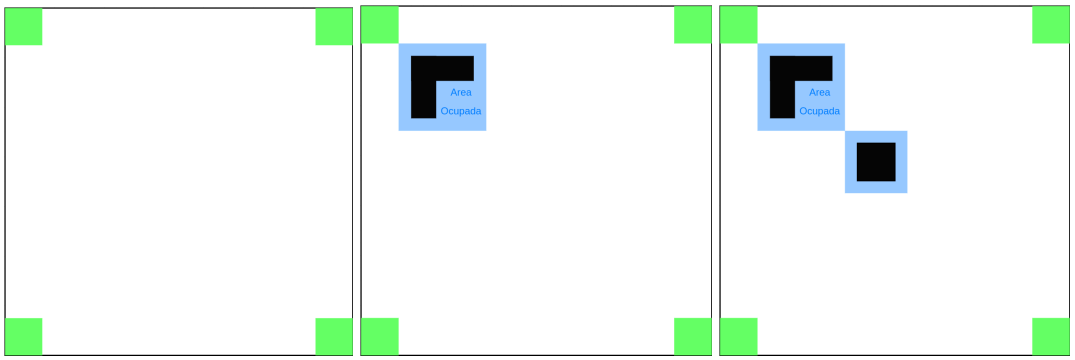

Figura 3.2. *Ejemplo de generación progresiva de la imagen a escanear*

Una vez ubicadas todas las formas, la imagen se reescala para ocupar el ancho del teléfono (1080px). Además, en esta versión, se añaden aleatoriamente patrones en forma de "L" y cuadrados, lo que incrementa la diversidad de las figuras generadas. Los resultados de esta versión se pueden ver en la ***Figura 4.3***.

Programa Capturador de Imágenes

En lo que respecta al segundo algoritmo mencionado, una característica fundamental es que se ejecuta directamente sobre la Raspberry Pi. Implementa una aplicación multi-hilo que interactúa con hardware y utiliza procesamiento de imágenes en tiempo real. Su funcionalidad principal incluye captura de video, detección de formas, monitoreo de la cámara en una interfaz gráfica, y envío de datos procesados al servidor.

- Hilo de Interfaz de Usuario

Esta parte del programa se encarga del manejo de comandos a través de la terminal del SO de la Raspberry Pi. Desde la misma, es posible elegir si se desea utilizar un monitor o no, lo cual resulta ideal en caso de querer conectarse a la RPI a través de SSH o si se cuenta con el modelo del sistema operativo Raspbian sin interfaz gráfica de usuario, como se muestra en la ***Figura 3.3***.

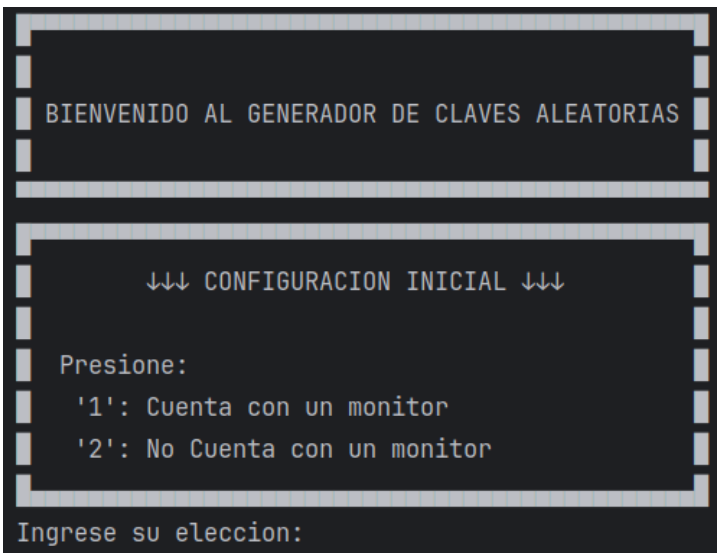

Figura 3.3. *Interfaz inicial.*

Una vez seleccionada la opción deseada, el sistema estará en condiciones de capturar la imagen generada por la aplicación Android que se describió antes. A partir de este punto, también permite habilitar o deshabilitar el envío de datos al servidor para ser almacenados, como se muestra en la ***Figura 3.4***.

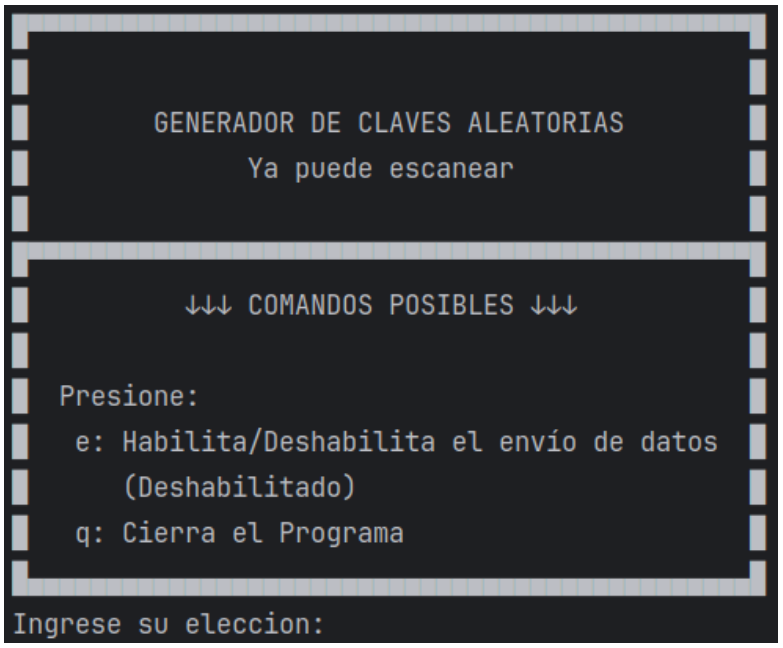

Figura 3.4. *Interfaz de control.*

- Hilo de Detección

El algoritmo que detecta la imagen es capaz de reconocer un área particular y aislarlo del resto de la imagen, consiguiendo así delimitar un perímetro cuyo contenido será el origen de los datos para generar parte de la semilla.

Como se indicó previamente, para facilitar la detección de la zona de interés que puede corresponder a un cuadrado o rectángulo, se deben detectar recuadros verdes en sus esquinas, que no formarán parte del análisis del algoritmo. Al unir estos mediante líneas rectas, se delimita el área que contiene la información deseada.

Para poder reconocer los mismos se utiliza el espacio de color HSV (Hue, Saturation, Value), que es una representación más intuitiva para identificar y manipular colores que el modelo RGB, porque separa la información de matiz (color) de la intensidad y la saturación, permiotiendo identificar el color puro independientemente de la intensidad o brillo, por lo que es menos sensible a variaciones de luz, donde:

- **Hue (H):** Representa el tipo de color (por ejemplo, rojo, verde, azul) y se mide en grados (0° a 360°). El verde típicamente tiene valores de 60° a 180°.
- **Saturation (S):** Indica la intensidad del color, desde un gris puro (0%) hasta el color más vibrante (100%).
- **Value (V):** Representa el brillo del color, desde negro (0%) hasta el máximo brillo del color (100%).

** Aclaración: se optó por este color para aprovechar los principios de un "chroma key" o pantalla verde, que facilita su reconocimiento debido a la alta luminancia de este color en comparación con los canales rojo y azul. Adicionalmente, la mayoría de sensores de cámaras funcionan bajo el patrón Bayer, que corresponde a un filtro de colores RGB (rojo, verde, azul) distribuidos al 25%, 50% y 25% respectivamente, basados en la sensibilidad del ojo humano.*

Una vez aislado este área, se interpreta su interior, lo que consiste en reconocer y procesar información correspondiente a la cantidad de figuras presentes, distinción de las mismas e información relativa a su posicionamiento. El algoritmo es capaz de detectar las figuras de tipo "L" y tipo "cuadrado" contando la cantidad de vértices que tiene cada figura negra detectada. Esto se realiza mediante el uso de una función de la biblioteca *ComputerVision* llamada *findCountours*, cuya salida es una lista de matrices donde se almacenan las coordenadas que forman los límites del objeto.

Para la correcta detección de las figuras se aplican diversos filtros para convertir la imagen a escala de grises, disminuir el brillo y aumentar el contraste. Estos ajustes reducen al mínimo aquellos "grises" que se generan alrededor de las figuras (difuminado) al no lograr un enfoque perfecto por parte de la cámara, Figura 5.6.

Una vez detectadas las figuras, las contabiliza y guarda sus coordenadas relativas al área delimitada, normalizándolas entre un valor mínimo (x,y) = (-50, -50) para la esquina inferior izquierda y máximo (x,y)=(50, 50) para la esquina superior derecha de la imagen. Esto Mostrado en la ***Figura 4.5*.**

Finalmente, se contabilizan las figuras de tipo "cuadrado" $\left(N_C\right)$ y de tipo "L" $\left(N_L\right)$ encontradas y se realiza una suma de las coordenadas correspondientes a estas últimas y se calcula la distancia desde la esquina superior izquierda de la imagen $(0,\ 0)$ hasta ese punto,

$$PosL = \sqrt{\left(\sum_{i=1}^{N_L} x_{L_i}\right)^2 + \left(\sum_{i=1}^{N_L} y_{L_i}\right)^2}$$

Donde,

- $\left(x_{L_i}, y_{L_i}\right)$ es la coordenada de posición de la “L” .

Una vez calculado este valor, se tiene disponible el total de la información a extraer de la imagen, obteniendo,

- $distancia\_l$ es la distancia de la suma ponderada de las posiciones de todas las “L”.
- $cant\_c$ es la cantidad total de “Cuadrados” detectados $\left(N_C\right)$.
- $CantTotal$ es la cantidad total de figuras detectadas $\left(N_L + N_C\right)$.

A modo de depuración, se hace una prueba y se muestran los valores obtenidos en ***Figura 4.6***.

3.1.3. Generación de Semilla y Clave

La generación de una clave aleatoria a partir de una semilla es un proceso esencial para asegurar la integridad y la aleatoriedad de las claves utilizadas en sistemas criptográficos. En este caso, la semilla es una combinación de los datos extraídos de las fuentes mencionadas previamente: imágenes capturadas por una cámara web y datos de temperatura proporcionados por el sensor DHT11. A continuación, se describe en detalle el proceso que se sigue para generar una clave aleatoria utilizando estos datos.

- **Concatenación de Datos**

El primer paso en la generación de la clave aleatoria es la concatenación de los datos que componen la semilla. Los datos utilizados incluyen,

- Cantidad de figuras “L” detectadas en la imagen procesada.
- Cantidad de figuras “cuadradas” identificadas en la imagen.
- Valor único, que consiste en la distancia de la suma ponderada de las posiciones de todas las “L”.
- Valor de temperatura captado por el sensor DHT11.

Al concatenar estos datos en un formato de texto, se obtiene una cadena que representa la semilla de la clave aleatoria. La longitud de esta cadena variará dependiendo de los datos extraídos, pero la longitud de la clave generada siempre será fija debido al algoritmo de hash utilizado. En este caso, se ha optado por un valor de 256 bits (32 bytes), lo que corresponde a una longitud estándar para claves seguras.

- **Aplicación del Algoritmo SHA-256**

Una vez concatenados los datos de entrada, se aplica el algoritmo SHA-256, una función hash criptográfica. Este algoritmo toma la cadena de datos concatenados y genera un valor hash único. El proceso de SHA-256 involucra varios pasos, los cuales se describen a continuación, además de ilustrarse en ***Figura 3.5***,

1. **Conversión a bytes**: Para que el algoritmo pueda procesar correctamente la entrada, la cadena concatenada se convierte a una secuencia de bytes, ya que SHA-256 trabaja con datos binarios.

Este paso se visualiza en ***Figura 2.3***.

2. **Relleno de Datos**: Debido a que SHA-256 requiere que los datos tengan una longitud múltiplo de 512 bits, el algoritmo realiza un proceso de relleno de semilla. Este relleno agrega un bit de valor 1 seguido de ceros, hasta alcanzar la longitud deseada. Además, se agrega la longitud original de la semilla en el último bloque de datos.

3. **División en Bloques**: Después de rellenar la semilla, esta se divide en bloques de 512 bits. Cada bloque de datos es procesado en las siguientes etapas.

4. **Rondas de Transformación**: Cada bloque pasa por múltiples rondas de transformación, donde se realizan operaciones complejas como rotaciones, desplazamientos y sumas modulares. Estas operaciones garantizan que el valor final generado sea único y difícil de predecir, incluso si la entrada cambia mínimamente (como al alterar un solo bit de la temperatura).

5. **Salida de la Clave**: Finalmente, después de las rondas de transformación, el algoritmo genera un valor de hash de 256 bits. Este valor es la clave aleatoria que representa la semilla procesada, y se muestra en formato hexadecimal.

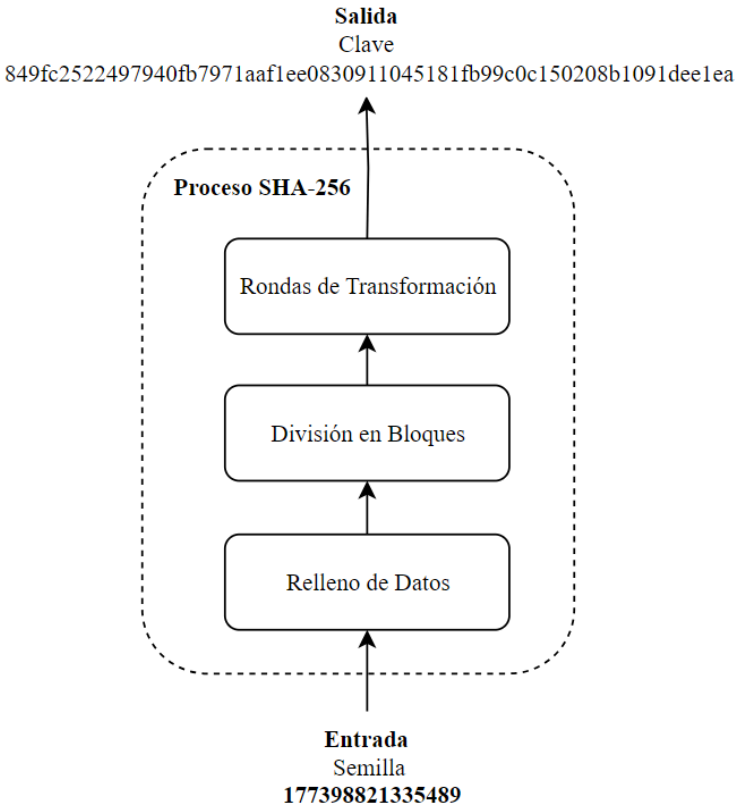

***Figura 3.5**. Aplicación de algoritmo SHA-256 a la semilla para generar clave.*

El valor generado por SHA-256 es prácticamente único, lo que significa que es extremadamente improbable que dos entradas diferentes generen el mismo valor de hash. Esta propiedad, conocida como **sensibilidad a cambios**, es crucial en la seguridad criptográfica, ya que garantiza que cualquier alteración mínima en los datos de entrada (por ejemplo, cambiar un solo bit de la temperatura) dará como resultado un valor de hash completamente diferente.

Calidad de Aleatoriedad y Propiedades de la Clave Generada

La calidad de la aleatoriedad es un aspecto esencial en la generación de claves criptográficas. El sistema implementado genera claves aleatorias a partir de datos no predecibles y únicos, obtenidos de diversas fuentes como la temperatura medida por el sensor DHT11, así como imágenes capturadas por una cámara web. Estos datos se combinan de forma tal que la clave generada es difícil de predecir, lo que refuerza su seguridad y fiabilidad para aplicaciones criptográficas sensibles.

Propiedades de la Clave Generada,

- **Unidireccionalidad**: El proceso de hashing utilizado es unidireccional, lo que significa que es prácticamente imposible revertir el valor hash para obtener los datos de entrada originales. Esto asegura que las claves generadas no puedan ser reconstruidas, aumentando la seguridad.
- **Dificultad de Predicción**: La clave generada es difícil de predecir, ya que depende de una combinación de datos sensibles y no predecibles, como la temperatura ambiente, la humedad y las características específicas de las imágenes procesadas. Esta dependencia de datos dinámicos garantiza que cada clave sea única y esté protegida frente a ataques de predicción.
- **Resistencia a Colisiones**: Aunque, teóricamente, podría existir una pequeña probabilidad de que diferentes entradas generen el mismo hash (colisión), esta posibilidad es extremadamente rara y se reduce significativamente al emplear SHA-256, que tiene una alta resistencia a este tipo de vulnerabilidad.
- **Seguridad**: Dado que el proceso de hashing es computacionalmente costoso de revertir y unidireccional, el sistema de generación de claves proporciona un alto nivel de seguridad. Esto lo hace adecuado para su uso en aplicaciones criptográficas en las que la integridad y la confidencialidad son fundamentales.

3.2. Almacenamiento y Visualización de Datos

En esta sección se describe cómo se gestionará el almacenamiento de las claves generadas y cómo se visualizarán los datos a través de la interfaz web. El sistema se apoya en un servidor que procesa peticiones desde la Raspberry Pi y desde el frontend de la página web.

3.2.1. Almacenamiento de datos

El almacenamiento de datos es crucial para registrar las semillas generadas y permitir su consulta o reproducción posterior. Para esto, se ha diseñado un sistema de almacenamiento que es descrito a continuación.

Para gestionar la comunicación entre la página web y la Raspberry Pi, se implementará un servidor encargado de procesar peticiones tanto desde la web como desde la RPI. Este servidor estará vinculado a una base de datos que almacenará información clave. De esta forma, no solo se podrán registrar las semillas generadas, sino también reproducirlas posteriormente para verificar el algoritmo.

La base de datos elegida es de tipo SQL, lo que ofrece un modelo estructurado y robusto para almacenar los datos de manera persistente. Esto garantiza que los registros históricos puedan ser consultados sin problemas según las necesidades del sistema. Además, las capacidades transaccionales de una base de datos SQL aseguran la consistencia, integridad y seguridad de los datos esenciales en un entorno donde la aleatoriedad y precisión de la información son fundamentales para la correcta operación del generador.

En cuanto a los datos que se guardan en la base de datos son aquellos que utiliza el algoritmo de detección detallado en la sección 3.1.3, la semilla generada, el tiempo en el que fue generado y el id único para ese dato, es decir, la tabla queda de la siguiente manera:

- Cantidad de figuras de tipo entero.
- Cantidad de cuadrados de tipo entero.
- Temperatura de tipo decimal.
- Semilla de tipo varchar.
- id de tipo entero.
- Tiempo de creación de tipo timestamp.

El servidor maneja las peticiones HTTP, conectándose a la base de datos para realizar operaciones de lectura y escritura. Este esquema permitirá la consulta eficiente de los datos por parte de los usuarios desde el frontend, así como el registro seguro de nuevas semillas desde la RPI.

La base de datos se encuentra desplegada en la nube, permitiendo registrar las semillas generadas y almacenar los parámetros asociados para su posterior consulta desde cualquier interfaz gráfica de base de datos, como por ejemplo MySQL Workbench. Esto es esencial para evaluar y verificar el rendimiento del generador de números aleatorios.

3.2.2. Visualización de Datos

La visualización de datos es esencial para que los usuarios puedan observar las claves generadas. Se ha decidido usar React para construir una interfaz web dinámica, que permitirá ver y gestionar los datos de manera eficiente.

3.2.2.1. Backend

El Backend del sistema es una pieza fundamental para la gestión y visualización de los datos. Se llevará a cabo un servidor utilizando Node.js con el framework Express, que permite manejar peticiones HTTP y comunicarse con la base de datos SQL. Este servidor actúa como intermediario entre la Raspberry Pi y la aplicación web desarrollada en React.

El Backend se encarga de guardar los datos relevantes en la base de datos SQL. Permite realizar operaciones de lectura y escritura, asegurando la persistencia y permitiendo la consulta de los registros históricos para análisis o verificación.

A través de rutas definidas en Express, el servidor recibe peticiones del frontend y envía las respuestas con los datos solicitados. Por ejemplo, cuando el usuario consulta las semillas generadas, el servidor extrae esta información de la base de datos y la envía en formato JSON, para que React pueda renderizarla en la interfaz de usuario.

Se implementaron tres endpoints esenciales en el backend: uno para almacenar los datos que envía la Raspberry Pi, otro para consultar las semillas generadas junto con sus parámetros, y uno auxiliar para conocer el estado del servidor, es decir si está operativo o no. Estos permiten la interacción fluida entre el servidor y los componentes del sistema.

Previamente se realizaron pruebas simulando la comunicación con la Raspberry Pi para verificar su funcionamiento, de este modo, se comprobó la capacidad del servidor para recibir y procesar datos de entrada, así como para enviar la información solicitada en respuesta a las consultas desde el frontend.

Este servidor se encuentra desplegado en la nube mediante la plataforma Render, donde se pueden observar métricas, logs y salud en general de la aplicación.

3.2.2.2. Frontend

Para la visualización de los datos, se decidió implementar el frontend utilizando React, una biblioteca de JavaScript ampliamente utilizada para el desarrollo de interfaces de usuario. React permite crear una aplicación web dinámica y modular, facilitando el desarrollo de componentes reutilizables para distintas partes de la interfaz.

La aplicación web permite visualizar las semillas generadas, junto con los parámetros utilizados. El uso de React garantiza una experiencia de usuario fluida, ya que el DOM virtual optimiza las actualizaciones en la interfaz, permitiendo reflejar los cambios casi instantáneamente a medida que se generan nuevos números aleatorios o se consultan registros históricos almacenados en la base de datos.

Para manejar la comunicación con el servidor, se utiliza *fetch* para realizar peticiones HTTP de manera eficiente, conectado el frontend con el Backend para recuperar y visualizar la información de la base de datos SQL.

El código central que hace que el sitio web muestre estos datos con esos estilos (colores, espacios, posición, etc.) se encuentra en dos archivos. En uno se especifica la estructura y funcionalidad del sitio y en el otro los estilos. A modo de ejemplo se muestra la Figura 5.10 y la Figura 5.11.

El frontend muestra datos extraídos de la base de datos a través del servidor, lo que permite visualizar las semillas generadas y sus parámetros asociados. Además, la interfaz de usuario tiene un diseño óptimo, que asegura una experiencia intuitiva y atractiva, con ajustes en la disposición de elementos y estilos que permiten una visualización clara de la información obtenida desde la base de datos.

Al igual que el servidor, el frontend se encuentra desplegado en la plataforma Render. Esto es fundamental ya que permite el acceso a la aplicación web desde cualquier dispositivo con internet mediante una URL.

3.3. Conclusiones

El desarrollo del sistema para la generación de claves aleatorias ha sido completado con éxito, cumpliento con los objetivos planteados al inicio del proyecto. A lo largo de la implementación, se utilizó un enfoque eficiente para garantizar la aleatoriedad y la seguridad en la generación de claves, utilizando métodos criptográficos que aseguran un alto nivel de imprevisibilidad en las claves generadas.

El sistema ha demostrado ser capaz de generar claves de longitud variable de manera rápida y confiable. A través de la implementación de algoritmos de generación de números aleatorios de calidad criptográfica, se ha logrado una distribución uniforme en el rango de valores posibles, minimizando el riesgo de patrones predecibles que podrían comprometer la seguridad.

Una parte clave del proyecto fue la implementación de un sistema de interfaz de usuario, que facilita la interacción con el programa, permitiendo al usuario generar claves de manera sencilla, con la opción de personalizar parámetros como la longitud de la clave y el tipo de caracteres a incluir.

Durante el proceso, se verificó que el sistema no solo cumpliera con los requerimientos de seguridad, sino que también fuera eficiente y fácil de utilizar, garantizando la generación de claves en un tiempo adecuado para su implementación en sistemas que requieren alta seguridad.

En conclusión, el proyecto ha sido una experiencia enriquecedora que no solo ha logrado su propósito principal, sino que también ha proporcionado una base sólida sobre la cual se pueden seguir incorporando nuevas características, como la integración con otros sistemas de gestión de seguridad, la personalización avanzada de las claves generadas, o la implementación de un sistema de almacenamiento seguro para las claves generadas. Este sistema puede ser utilizado en una variedad de aplicaciones que requieran la creación de claves seguras, brindando una herramienta útil en el ámbito de la seguridad informática.

4. Documentación Relacionada

En principio, se adjunta en ***Video 1*** una demostración visual del sistema de generación de números aleatorios desarrollado de forma integral.

Video 1. *Demostración del sistema completo*.

Por otro lado, en ***Figura 4.1*** se puede apreciar la conexión requerida de los componentes en los puertos o pines de la Raspberry Pi, 3.1.1.

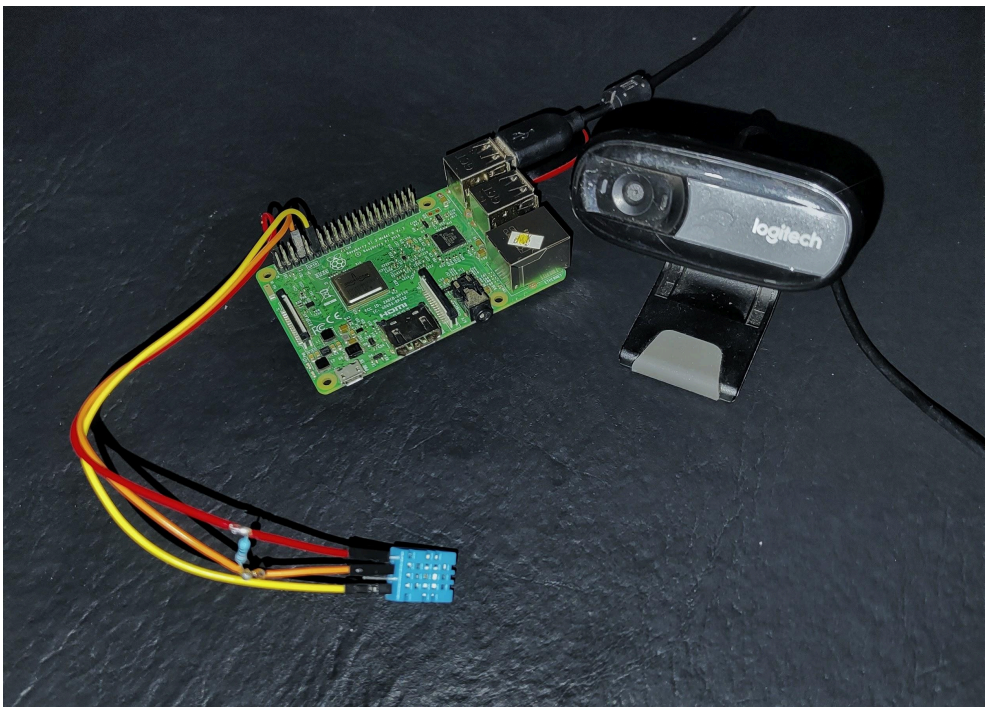

Figura 4.1. *Conexiones de hardware.*

La ***Figura 4.2*** y ***Figura 4.3*** incluye ejemplos de los patrones generados por las respectivas versiones desarrolladas de la aplicación Android, 3.1.2.2.

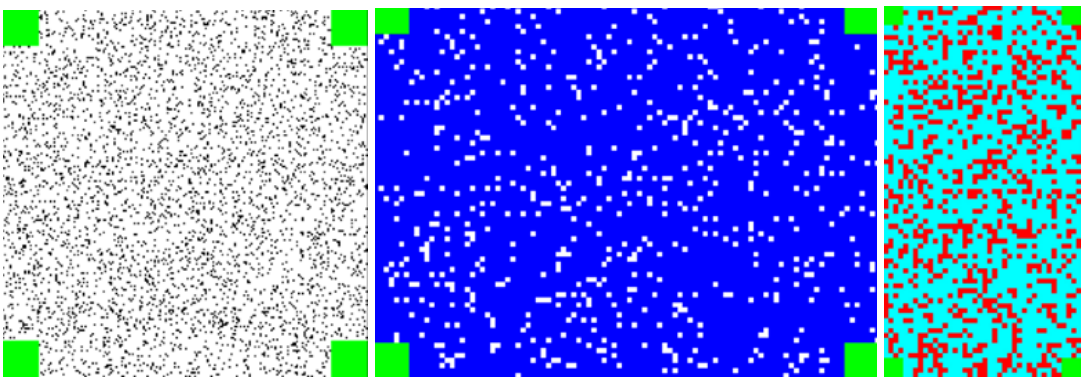

Figura 4.2. *Ejemplos de patrones generados por el algoritmo V1.*

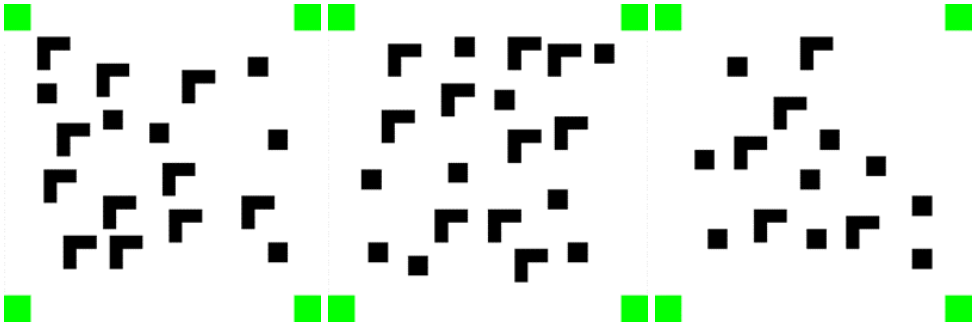

Figura 4.3. *Ejemplos de patrones generados por el algoritmo V2.*

La ***Figura 4.4*** es una captura de pantalla de las opciones de configuración de la aplicación descrita en 3.1.2.2.

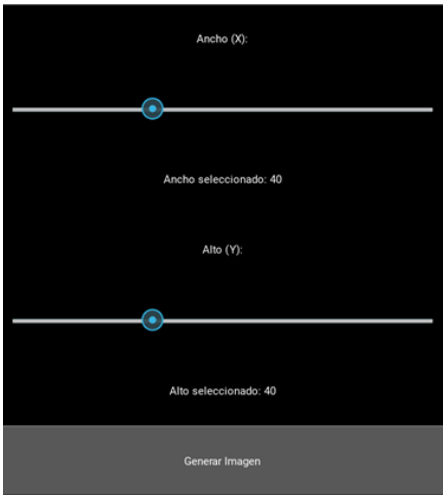

Figura 4.4. *UI de la aplicación.*

La ***Figura 4.5*** muestra la aplicación de filtros en la última versión del algoritmo generador de patrones, 3.1.2.2.

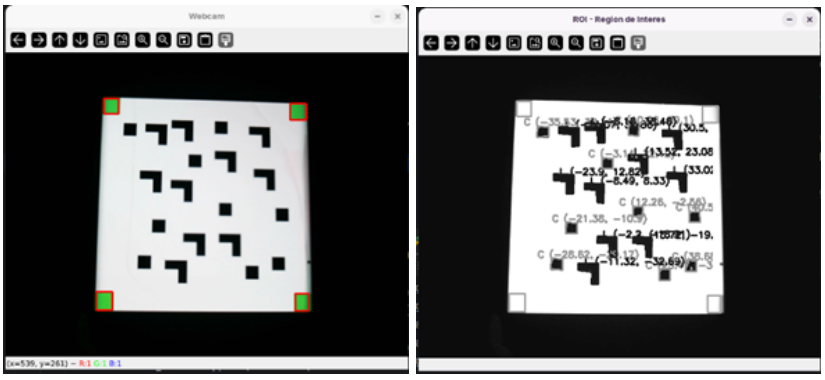

Figura 4.5. *Aplicación de filtros.*

La ***Figura 4.6*** muestra la detección de coordenadas de las figuras que capta el algoritmo generador de patrones, 3.1.2.2.

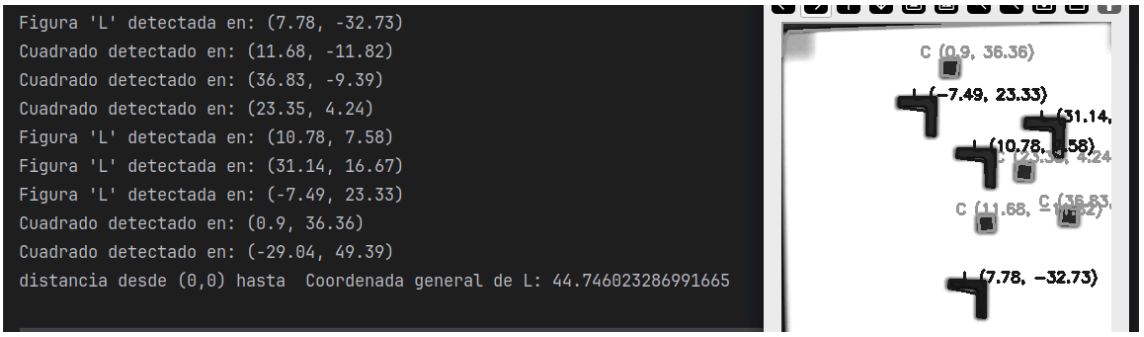

Figura 4.6. *Detección de coordenadas.*

La ***Figura 4.7*** es una captura de pantalla de la interfaz gráfica desarrollada, 3.2.2.2.

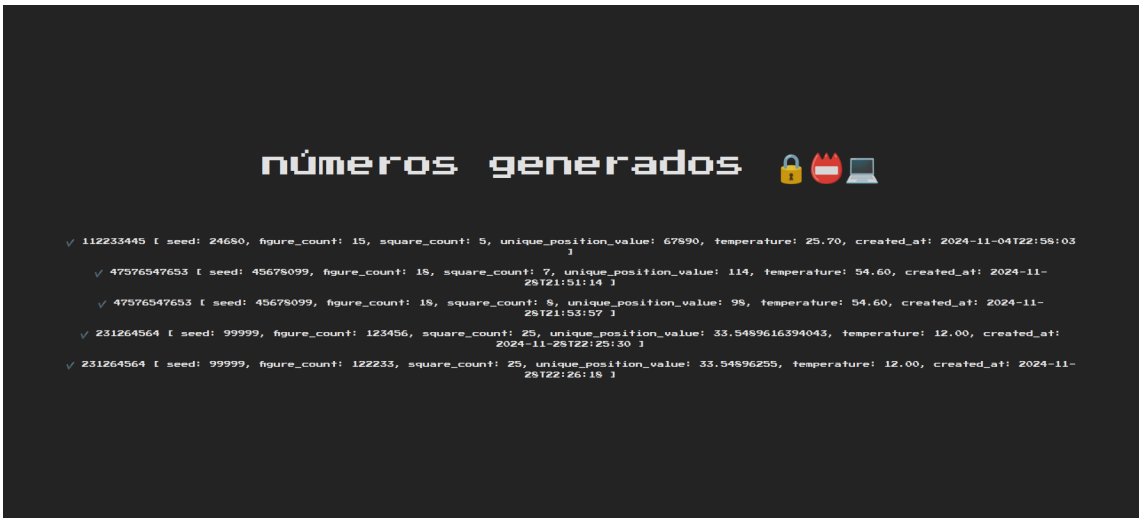

Figura 4.7*. Captura de pantalla del frontend.*

4.1. Organización de tareas

Respecto a la organización de las tareas del equipo de trabajo, se adjunta la Bitácora y el Repositorio de Código utilizado.

Apéndice A

Materiales y Presupuesto

Para la consolidación de este proyecto, se requieren dispositivos de hardware que se detallan a continuación.

Dispositivo	**Descripción**	**Precio ($)**	**Referencias**	**Brindado por la cátedra**
Raspberry Pi 3	Computadora de placa reducida	79900	Raspberry Pi	Si
Fuente de alimentación	Fuente de alimentación para la RPI (5V/2.5ª)	21900	Fuente	No
Sensor DHT11	Sensor de temperatura y humedad	8410	Sensor DHT11	Si
Resistencia	Resistencia Pull-Up	1997	Resistencia 10KΩ ¼W	No
Cámara Web	Cámara Web compatible con RPI	37622	Cámara Web Logitech	Si