Documento avance primer prototipo “Sistema de Ingreso por Reconocimiento Biométrico en la Universidad del Cauca”

Luis Miguel Ortiz Muñoz

Carlos Daniel Collazos Zambrano

Cristian Felipe Bolaños Ortega

Jefferson Danilo Noguera

Mary Cristina Carrascal Reyes

Hermes Fabian Vargas Rosero

Facultad de Ingeniería Electrónica y Telecomunicaciones

Ingeniería Electrónica y Telecomunicaciones

Laboratorio 4 de electrónica.

Popayán, septiembre de 2024

Índice

1. INTRODUCCIÓN

Este informe presenta el progreso en el desarrollo del primer prototipo del Sistema de Ingreso por Reconocimiento Biométrico para la Universidad del Cauca, un proyecto que busca mejorar los procesos de control de acceso a las instalaciones de la institución mediante la implementación de tecnologías biométricas. En esta fase inicial, los esfuerzos se han centrado en el diseño y creación de las interfaces web, la configuración de la base de datos y la implementación del servidor de autenticación, con el objetivo de proporcionar una plataforma funcional y escalable.

El desarrollo de las interfaces web se ha llevado a cabo utilizando Angular, un framework que permite construir aplicaciones web dinámicas y de alto rendimiento. Se han diseñado interfaces clave para el inicio de sesión, la gestión de usuarios y el registro de accesos, con un enfoque en la usabilidad y en garantizar una experiencia de usuario eficiente y simple para el personal administrativo encargado del control de accesos. Estas interfaces están preparadas para la futura integración con los dispositivos de autenticación biométrica, lo que facilitará una transición fluida hacia el uso de huellas dactilares como método de verificación.

Simultáneamente, se ha configurado una base de datos utilizando MongoDB, una solución NoSQL ideal para gestionar grandes volúmenes de datos no estructurados, como las huellas dactilares. La flexibilidad de MongoDB permite un almacenamiento eficiente de los datos biométricos, optimizado mediante algoritmos de compresión que reducen el espacio en disco sin comprometer la velocidad de acceso y verificación. Este enfoque garantiza que el sistema pueda escalar adecuadamente a medida que aumente el número de usuarios.

Otro componente clave de este primer prototipo es la implementación de un servidor de autenticación utilizando FastAPI, un framework que permite manejar solicitudes concurrentes de manera eficiente, esencial para un sistema que podría estar recibiendo múltiples peticiones simultáneas desde diferentes puntos de acceso en la universidad. Aunque el hardware de autenticación biométrica, como los lectores de huellas dactilares, aún no está disponible, se ha diseñado un servidor de pruebas que simula la autenticación mediante credenciales, sentando las bases para la futura integración de los dispositivos biométricos.

El presente informe no solo detalla los logros alcanzados hasta la fecha, sino que también describe los próximos pasos a seguir, que incluyen la integración del lector de huellas dactilares y el fortalecimiento de la seguridad de los datos biométricos

almacenados. sentando una base tecnológica sólida para las fases futuras del proyecto.

2. OBJETIVOS

- Diseñar e implementar interfaces web intuitivas y funcionales utilizando Angular, que permitan la gestión de usuarios, el control de acceso y la generación de reportes.
- Configurar una base de datos segura y eficiente en MongoDB para el almacenamiento y manejo de datos biométricos, incorporando algoritmos de compresión para optimizar el uso de espacio y mecanismos de cifrado para proteger la información.
- Desarrollar un servidor de autenticación con FastAPI que procese solicitudes de acceso y valide credenciales biométricas en tiempo real.
- Integrar un lector de huellas dactilares para la captura y verificación de huellas biométricas dentro del sistema.
- Implementar medidas de seguridad avanzadas para garantizar la privacidad y protección de los datos biométricos almacenados en el sistema.

3. FASES DESARROLLADAS

3.1 Creación de las interfaces de la aplicación Web

Diseño de la Interfaz

- **Mockups:**
  - Crear mockups o bocetos de cada interfaz para visualizar la disposición de elementos.

Estructura del Proyecto Angular

- **Creación del Proyecto:**
  - Usar Angular CLI para iniciar un nuevo proyecto Angular.
- **Configuración de Rutas:**
  - Definir las rutas necesarias para navegar entre las distintas interfaces.

Implementación de Funcionalidades

- **Interfaz de inicio de sesión:**
  - Crear un formulario con campos para cédula y contraseña.

    - Implementar la lógica para validar las credenciales.
- **Interfaz de administrador:**
    - Crear el menú de navegación que permita acceder a las distintas funcionalidades (registro de portería, gestión de usuarios, generación de reportes).

- **Interfaz de Gestión de Usuarios:**
    - Desarrollar la tabla de usuarios con las funciones de agregar, editar y eliminar usuarios.
    - Añadir un formulario para la entrada de datos de los usuarios.
- **Interfaz de Registro de Usuarios:**
    - Crear un formulario con campos para cédula, nombre y tipo de usuario.
    - Implementar la lógica para validar las credenciales.
    - Implementar un botón con la finalidad de registrar huella y representar la respuesta del servidor.

3.2 Conexión hardware con servidor

- **Conexión a la red con el microcontrolador WT32 ETH01**
    - Logros: Diseño previo del código que teóricamente tendría que establecer la conexión del microcontrolador con la red de la Universidad por medio de un cable Ethernet.
    - Desafíos: No poseer el microcontrolador para poder realizar las pruebas pertinentes de conexión.
- **Definir y probar los frameworks del servidor de autenticación.**
    - Logros: Definición e implementación de las tecnologías a usar para desarrollar un servidor robusto para el sistema a realizar
    - Desafíos: Limitación por la necesidad de la implementación de Python, ya que se necesita Python para los procesos de autenticación con Machine Learning.

3.3 Configuración de la base de datos

En esta fase, se configuró la base de datos del sistema de control de acceso de la Universidad del Cauca, utilizando MongoDB para almacenar las huellas dactilares capacitivas y los datos de los usuarios. MongoDB, al ser una base de datos NoSQL, fue elegida por su flexibilidad y capacidad para gestionar grandes volúmenes de datos de manera eficiente, especialmente cuando se trata de información biométrica como huellas dactilares.

- **Almacenamiento eficiente de datos**
    - Logros:

Estructura de Colecciones Definida: Se diseñaron las colecciones necesarias para el proyecto, como la colección de usuarios y la colección de huellas dactilares. La primera contiene información básica de los usuarios, como el nombre y el ID, mientras que la segunda almacena los datos biométricos, optimizando su acceso y almacenamiento.
Almacenamiento Eficiente de Huellas: Se implementaron estrategias para almacenar las huellas dactilares de forma eficiente en la base de datos. Las huellas fueron convertidas a un formato digital optimizado, como matrices de puntos o características clave que reducen el espacio en disco y mejoran la velocidad de acceso en comparación con almacenar imágenes completas.

    - Desafíos

Almacenamiento de Datos Biométricos: La conversión de las huellas dactilares a un formato eficiente y su almacenamiento en MongoDB presentó retos técnicos, especialmente en garantizar que el formato optimizado fuera lo suficientemente rápida sin comprometer la precisión en la comparación de la huella.

4. AVANCES PRIMER PROTOTIPO

4.1 Alcance y requisitos planteados

Según el primer documento de diseño del proyecto se tiene que el alcance del primer prototipo es el siguiente:

- **Interfaz de inicio de sesión** para el administrador del sistema que en este caso sería un funcionario administrativo de la división correspondiente dentro de la institución.
- **Interfaz de administrador** que permite la visualización de los accesos permitidos y denegados (simulados), así como el estado de los lectores de huella instalados.
- **Interfaz de registro de usuario:** permite agregar nuevos usuarios, especificando los campos de cedula, nombre y tipo, además de mandar una solicitud al servidor para la toma de huella.
- **Diseño de la base de datos** y configuración inicial para almacenar las plantillas de huellas y los datos de los usuarios.

Y los requisitos esperados para este primer prototipo son los siguientes:

- El aplicativo deberá funcionar sin el uso del lector de huellas por el momento, debido al tiempo que se necesita para obtener los dispositivos hardware del sistema, por lo que en el primer prototipo solamente se verán avances con funcionalidad solamente de software.
- Estructura de la base de datos para registrar usuarios y guardar las huellas dactilares (en forma de imágenes o templates).
- Conexión a un servidor de pruebas (local) para simular la autenticación de datos.

4.2 Características del primer prototipo

- I**nterfaz de inicio de sesión:**
  Permite a los usuarios autenticarse mediante su cédula y contraseña, garantizando que solo los usuarios autorizados puedan acceder al sistema. La interfaz está diseñada para ser intuitiva y fácil de usar, con validaciones para asegurar la correcta entrada de datos.
- I**nterfaz de Administrador:**
  Proporciona acceso a funcionalidades críticas como el registro de portería, la gestión de usuarios y la generación de reportes. Esta interfaz está diseñada para facilitar la administración de las diferentes secciones del sistema y mejorar la eficiencia del flujo de trabajo.
- **Interfaz de Gestión de Usuarios:**
  Permite la visualización, creación, actualización y eliminación de usuarios. Incluye una tabla que muestra los datos de los usuarios, así como un formulario para la entrada y edición de datos, facilitando así la gestión eficiente de la información de los usuarios.
- **Interfaz de Registro de Usuario**: Esta interfaz permite registrar nuevos usuarios en el sistema de manera rápida y sencilla. Incluye un formulario donde el administrador puede ingresar los datos básicos del usuario, como nombre, cédula y el registro de huella.
  Los datos ingresados son validados en tiempo real, y se proporcionan mensajes de error o advertencias en caso de que algún campo no cumpla con los requisitos establecidos (como el carácter numérico de la cedula o la falta de un nombre).
- **Servidor de prueba con FastAPI y Uvicorn:** Para el desarrollo del servidor de autenticación, donde se procesan las huellas para permitir el acceso, se ha optado por utilizar FastAPI como framework backend, junto con Uvicorn como servidor de aplicaciones ASGI. Esta decisión se ha tomado debido a las siguientes razones:

  - ***Capacidad de Manejo de Solicitudes Concurrentes:*** La Universidad del Cauca cuenta multiples puntos de acceso, lo que implica un alto volumen de

solicitudes simultáneas al servidor de autenticación. FastAPI, basado en la especificación ASGI, está diseñado para manejar solicitudes concurrentes de manera eficiente gracias a su arquitectura asíncrona, lo que permite que el sistema responda rápidamente y sin bloqueos ante múltiples peticiones. Esto asegura un acceso fluido y sin demoras en los puntos de control distribuidos por el campus.

- ***Desempeño y Eficiencia:*** FastAPI es conocido por su alto rendimiento, comparable con frameworks de otras tecnologías como Node.js y Go. Esto lo hace ideal para un sistema que requiere tiempos de respuesta rápidos y consistentes, garantizando una experiencia de usuario fluida en la autenticación de huellas dactilares. Además, al estar combinado con Uvicorn, un servidor ligero y eficiente, se maximiza la capacidad de respuesta del sistema, incluso bajo alta demanda.
- ***Integración con Python y Machine Learning:*** La implementación de un modelo de machine learning en el servidor es un componente crítico para la autenticación precisa de las huellas dactilares. Dado que Python es el lenguaje principal para el desarrollo de modelos de machine learning, el uso de FastAPI facilita la integración directa de estos modelos en el sistema de autenticación. Esto permite una verificación rápida y eficiente de las huellas dactilares, mejorando la precisión y la seguridad del sistema.
- ***Alta Disponibilidad y Escalabilidad***: La alta disponibilidad del servidor es esencial para garantizar el acceso continuo de los usuarios a las instalaciones. FastAPI, en combinación con Uvicorn y otras herramientas de despliegue como Docker o Kubernetes, permite crear entornos escalables y robustos. Esto significa que el sistema puede adaptarse a un número creciente de puntos de acceso o usuarios sin comprometer su rendimiento o estabilidad, asegurando una operación ininterrumpida y fiable.
- ***Documentación Automática y Facilidad de Desarrollo***: FastAPI genera automáticamente documentación interactiva para las API, lo que facilita el desarrollo, la prueba y la integración del sistema. Esto no solo acelera el proceso de desarrollo, sino que también simplifica la colaboración con otros miembros del equipo, al proporcionar una referencia clara y accesible para las funcionalidades del servidor.

- ***MongoDB (NoSQL):*** Se eligió MongoDB debido a su flexibilidad para almacenar datos no estructurados, su escalabilidad para manejar grandes volúmenes de datos biométricos, y su rapidez en consultas gracias a su modelo de base de datos basado en documentos. Esta decisión fue clave para

garantizar que el sistema pueda manejar eficientemente la verificación de huellas en tiempo real.

- **Colecciones de Huellas Dactilares:** Se creó una colección específica para almacenar las huellas dactilares en un formato optimizado. El objetivo fue garantizar que cada huella ocupe el menor espacio posible en el disco y que las consultas para verificar coincidencias sean rápidas.

4.3 Porcentaje de cumplimiento de requisitos del primer prototipo

- El aplicativo deberá funcionar sin el uso del lector de huellas por el momento, debido al tiempo que se necesita para obtener los dispositivos hardware del sistema, por lo que en el primer prototipo solamente se verán avances con funcionalidad solamente de software.

  Porcentaje de cumplimiento: 90%

  El aplicativo Web, funciona por el momento sin la necesidad del lector de huellas debido a que todavía no se ha implementado la función de registrar huellas, por ahora la aplicación puede realizar las demás tareas que no dependen del lector de huellas.

- Estructura de la base de datos para registrar usuarios y guardar las huellas dactilares (en forma de imágenes o templates).
  Porcentaje de cumplimiento: 90%
  Hasta ahora, se ha completado aproximadamente un 90% de los objetivos relacionados con la configuración de la base de datos. Los avances incluyen:
  - Configuración Inicial Exitosa: La base de datos MongoDB está completamente operativa, con las colecciones necesarias para almacenar usuarios y huellas dactilares.
  - Optimización del Almacenamiento: Se ha logrado almacenar las huellas en un formato eficiente, optimizando el uso del espacio y la velocidad de acceso a los datos. De esta manera las huellas están listas para ser almacenadas y verificadas en un futuro.

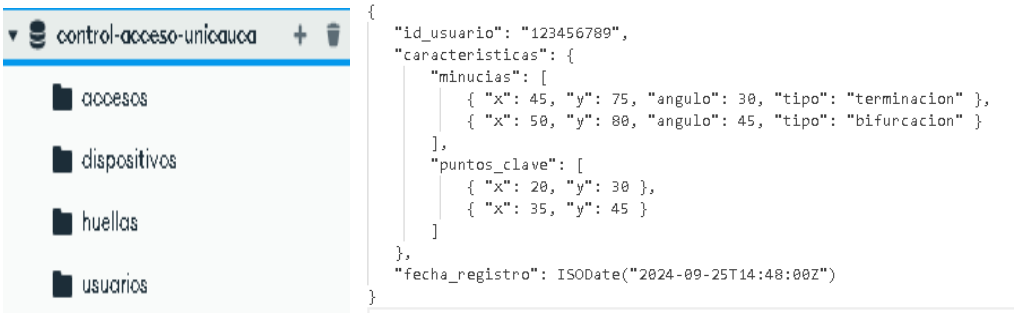

Figura 1 Colecciones creadas en la base de datos y ejemplo documento de colección huellas

No se logró el 100% de los requisitos para esta tarea debido a la falta de integración con la interfaz web para que se permitiera el registro de usuarios, por el momento se deja implementada la colección ya funcional en la base de datos.

- Interfaz de inicio de sesión:
  Porcentaje de cumplimiento: 80%
  La interfaz de inicio de sesión está funcional en cuanto a entrada de datos y verificación de credenciales, pero las validaciones de datos no se están realizando en apoyo de la base de datos.

- Interfaz de inicio de administrador:
  Porcentaje de cumplimiento: 100%
  La interfaz de administración ha sido completamente desarrollada, proporcionando acceso a las funcionalidades de registro de portería, gestión de usuarios, generación de reportes y un botón para regresar a la interfaz de inicio de sesión. Esta interfaz cumple con todos los requisitos de esta fase.

- Interfaz de gestión de usuarios:
  Porcentaje de cumplimiento: 90%
  La interfaz permite la creación, edición, eliminación y visualización de usuarios de manera eficiente. Sin embargo, falta la integración completa con el lector de huellas para el registro de datos biométricos, lo que impide que el flujo completo esté disponible, además de que también falta integrar la base de datos para la recepción de la lista de usuarios.

- Interfaz de registro de usuarios:
  Porcentaje de cumplimiento: 80%

La interfaz de registro de usuario permite la creación de nuevos usuarios con campos como nombre y cédula. Sin embargo, la funcionalidad para capturar y almacenar huellas dactilares aún no está habilitada debido a la falta del hardware necesario y la falta de integración con la base de datos para el envío de los datos.

- Conexión a un servidor de pruebas (local) para simular la autenticación de datos.

  Porcentaje de cumplimiento: 100%

  Para este primer prototipo se diseñó con éxito el primer servidor de prueba con FastApi, donde se realizaron las pruebas pertinentes de conexión y respuesta de solicitudes, de las cuales se obtuvieron resultados óptimos.

5. EJECUCIÓN DEL PROYECTO

A continuación, se explica de forma detallada como se elaboraron las fases desarrolladas de este primer prototipo. Cabe recalcar que la mayoría del código se representará por medio de diagramas con el fin de un mejor entendimiento. En caso de requerir la visualización completa de los códigos realizados, dirigirse al repositorio GitHub del proyecto en [1].

5.1 Creación de las Interfaces de la Aplicación Web

En esta fase, nos centramos en la creación de las diferentes interfaces que componen el sistema, asegurando que sean intuitivas y fáciles de usar. El proceso comenzó con la creación de mockups que sirvieron como guía visual para el desarrollo de cada interfaz. Utilizando **Angular CLI**, se creó la estructura base del proyecto, y se configuraron las rutas para la navegación entre las distintas pantallas. Obteniendo de esta manera los siguientes resultados:

- Diseño de mockups que definen la estructura visual de las interfaces.
- Configuración de rutas en Angular para facilitar la navegación entre pantallas como el **inicio de sesión**, **administración** y **gestión de usuarios (Agregar, eliminar y editar)**.

5.2 Configuración de la base de datos

Diagrama de colecciones de la base de datos

Este diagrama visualiza cómo están organizadas las colecciones en MongoDB y las relaciones entre ellas.

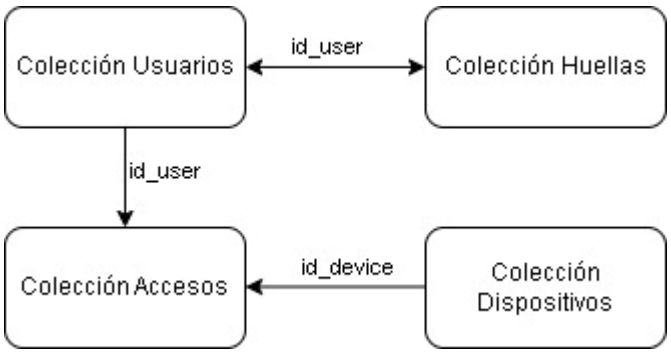

Figura 2 Colecciones en base de datos e interacción

Colección usuarios: Relacionada con las colecciones de huellas y accesos a través de id del usuario.

Colección huellas: Contiene la información característica de la huella y está asociada a un único usuario.

Colección accesos: Relacionada con usuarios y dispositivos a través del id del usuario e id del dispositivo. También contiene los intentos de registro de acceso (exitosos y fallidos), enlazada con el id usuario.

Colección dispositivos: Se relaciona directamente con los accesos al identificar el dispositivo desde el cuál se hizo el acceso gracias a su identificación en la red de dispositivos que habrá en cada punto de entrada.

5.3 Conexión hardware con servidor

Conexión a la red con el microcontrolador WT32 ETH01

Para establecer la conexión del hardware y el servidor se necesita de la implementación de un código para el microcontrolador WT32 ETH01, donde este debería conectarse a la red de forma automática, obteniendo una IP por medio del protocolo DHCP, cuando el microcontrolador sea encendido y conectado a algún switch o router de la universidad por medio de un cable Ethernet.

Teniendo en cuenta que el microcontrolador aún no se ha adquirido, no se han podido realizar pruebas para verificar su funcionalidad, pero la elaboración de este código está basada en la documentación adquirida y se asume que no debería poseer fallas al ser un código bastante sencillo. A continuación, se presenta un diagrama de flujo del código diseñado:

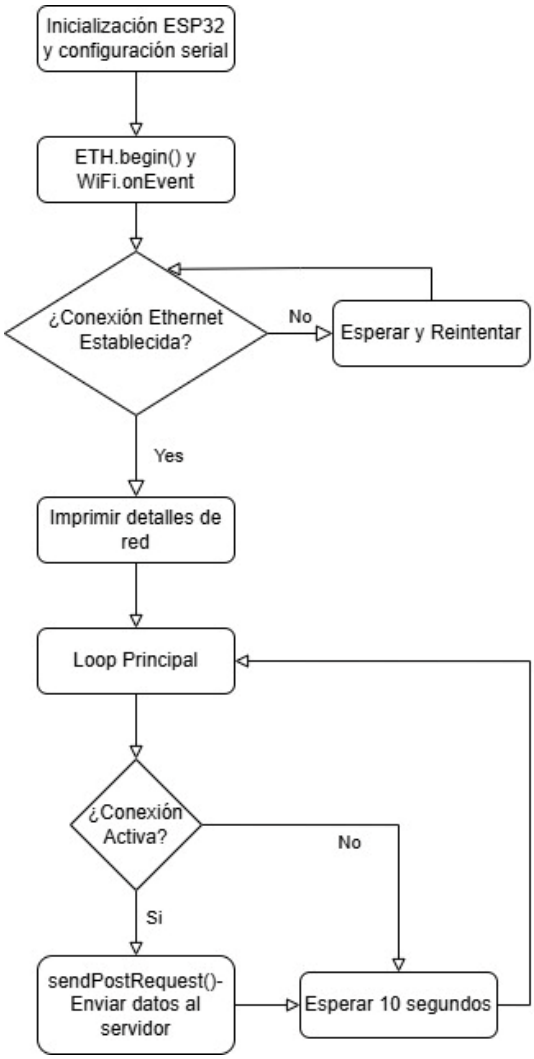

Figura 3 Diagrama de flujo de código de conexión del WT32-ETH01 a la red.

En la Figura 3 se observa el diagrama de flujo para el código empleado en el WT32-ETH01 para establecer la conexión a internet. Su explicación es la siguiente:

- Inicio - Inicialización del Microcontrolador
    - Acción: Configura el WT32-ETH01 para la comunicación serial.
    - Propósito: Permite la comunicación entre el WT32-ETH01 y el ordenador para mostrar mensajes en el monitor serial, facilitando la depuración.
- Configuración de Ethernet

    - Acción: Inicia la conexión Ethernet usando la función ETH.begin() y configura el manejador de eventos WiFi.onEvent) para monitorear cambios en la conexión.
    - Propósito: Configura el módulo Ethernet para que se conecte a la red y establece el manejador de eventos para detectar cambios en la conexión, como la obtención de una IP o la desconexión.
- Verificar Estado de Conexión
    - Acción: El microcontrolador espera a que se establezca la conexión Ethernet y obtiene una dirección IP mediante DHCP.
    - Propósito: Cuando se obtiene una dirección IP, el evento imprime detalles de la red, como IP, máscara de subred y puerta de enlace, y establece una variable 'eth_connected' a 'true'.
- Loop Principal
    - Acción: El microcontrolador entra en un bucle donde monitorea constantemente la conexión Ethernet.
    - Propósito: Verifica si la conexión está activa y realiza acciones en consecuencia, como enviar datos al servidor.
- Verificar Conexión Activa
    - Acción: Verifica si la variable eth_connected es verdadera.
    - Propósito: Determina si el microcontrolador está conectado a la red Ethernet y puede enviar datos.
- Enviar Datos al Servidor
    - Acción: Si la conexión Ethernet está activa, se envían datos al servidor usando la función sendPostRequest().
    - Propósito: Envía una solicitud POST al servidor remoto con los datos especificados (nombre y estado). Esto permite la comunicación con el servidor para informar sobre el estado del dispositivo.
- Esperar 10 Segundos
    - Acción: Después de enviar la solicitud al servidor, el microcontrolador espera 10 segundos antes de volver a enviar datos.
    - Propósito: Evita que el microcontrolador envíe solicitudes constantemente, dando un intervalo de tiempo entre cada envío de datos para no sobrecargar la red o el servidor.

- Fin del Ciclo y Repetición
    - Acción: El ciclo principal se repite, verificando nuevamente la conexión y enviando datos si la conexión sigue activa.
    - Propósito: Mantener una comunicación continua con el servidor para enviar información periódicamente.

Con este código se logra una conexión con el servidor a través de Ethernet y también se puede cambiar a que dirección IP se envía la solicitud según se necesite. Posteriormente, cuando se haya adquirido el microcontrolador, se realizarán las pruebas pertinentes para verificar su correcto funcionamiento.

Servidor de prueba para la autenticación de datos

Haciendo uso de Fast Api y Uvicorn se desplego un servidor el cual recibe una solicitud post que tiene consigo dos credenciales: nombre y clave. Donde internamente en el código hay un diccionario de usuarios que contiene nombres de usuario y contraseñas como pares clave-valor:

"Carlos": "clave123",

"Maria": "clave456",

"Juan": "clave789"

Por lo cual, este diccionario se vuelve una base de datos simulada. Entonces lo siguiente es hacer el proceso de autenticación de los datos y dar una respuesta, para esto se implementó un código Python que es representado por el diagrama de flujo de la Figura x.

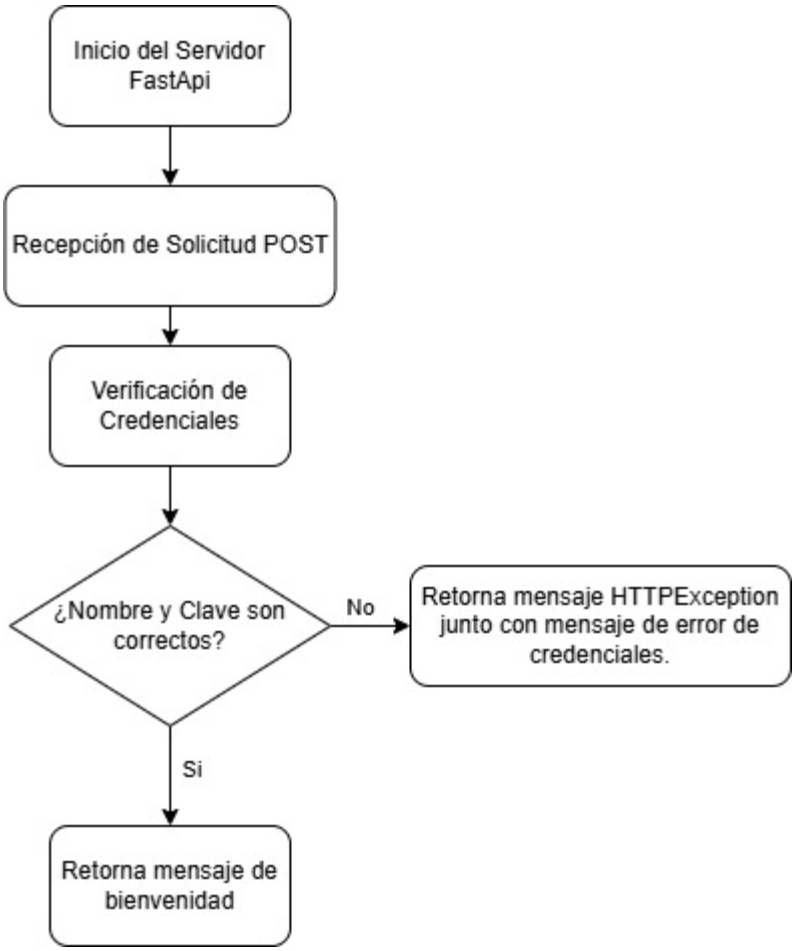

Figura 4 Diagrama de flujo de código de servidor de prueba para la autenticación de datos.

Con este código se obtiene una primera base para el desarrollo del servidor de autenticación, entonces para verificar su correcto funcionamiento se realizan los siguientes pasos:

I. Iniciar el servidor desde un terminal: Por medio del terminal se inicia el servidor desde su ubicación: C:\Users\colla\Desktop\Collazos\Universidad\Ing Elelctronica\Octavo Semestre\Lab 4 Electronica\Servidor\uvicorn main:app --workers 4 --host 0.0.0.0 --port 8000

   El comando “uvicorn main:app --workers 4 --host 0.0.0.0 --port 8000” se desglosa de la siguiente forma:

   - Uvicorn: Es el servidor ASGI (Asynchronous Server Gateway Interface) que se utiliza para ejecutar aplicaciones web asíncronas, que ene este caso es FastApi
   - main:app:

- main: Indica el nombre del archivo Python donde se encuentra la aplicación FastAPI. En este caso, main.py.
- app: Se refiere a la instancia de la aplicación FastAPI dentro del archivo main.py. Es decir, dentro del archivo main.py hay una línea que define app = FastAPI().

- --workers 4: Indica el número de trabajadores (workers) que se utilizarán para manejar las solicitudes. Donde 4 trabajadores significa que el servidor creará cuatro procesos separados para manejar las solicitudes de los clientes. Esto permite manejar múltiples solicitudes concurrentes de manera más eficiente, especialmente útil para aplicaciones con alta demanda.
- --host 0.0.0.0: Especifica la dirección IP en la que el servidor escuchará las solicitudes. Esto significa que el servidor estará disponible en todas las interfaces de red de la máquina, es decir, puede ser accedido desde cualquier dispositivo en la red local o externa, dependiendo de la configuración de red.
- --port 8000: Define el puerto en el que el servidor escuchará las solicitudes.8000 es el puerto donde se ejecutará la aplicación.

Después de ejecutar el comando se obtiene el siguiente resultado:

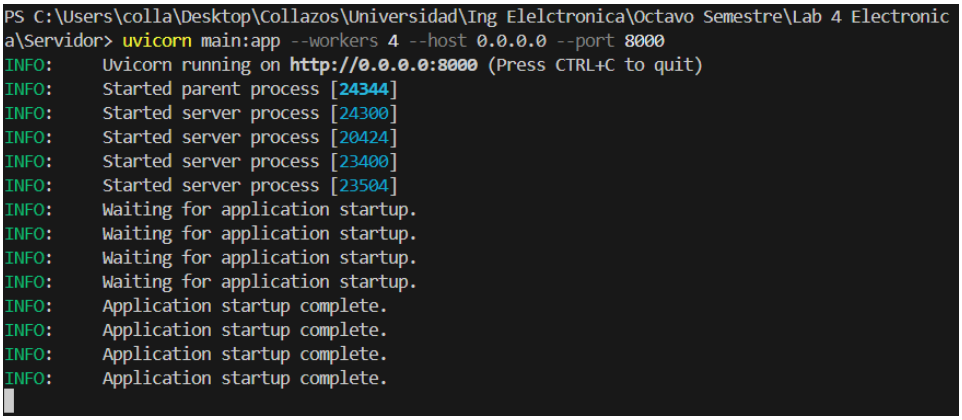

Figura 5 Resultado de ejecución del servidor en FastApi.

En la Figura 5 se puede observar como el servidor se inicia correctamente y esta disponible para recibir solicitudes POST de algún cliente.

II. Enviar solicitud POST al servidor: Para verificar el correcto funcionamiento del servidor se envía una solicitud POST la plataforma

API PostMan, donde está sirve para realizar pruebas entre APIs. La prueba del servidor se realizó de la siguiente manera:

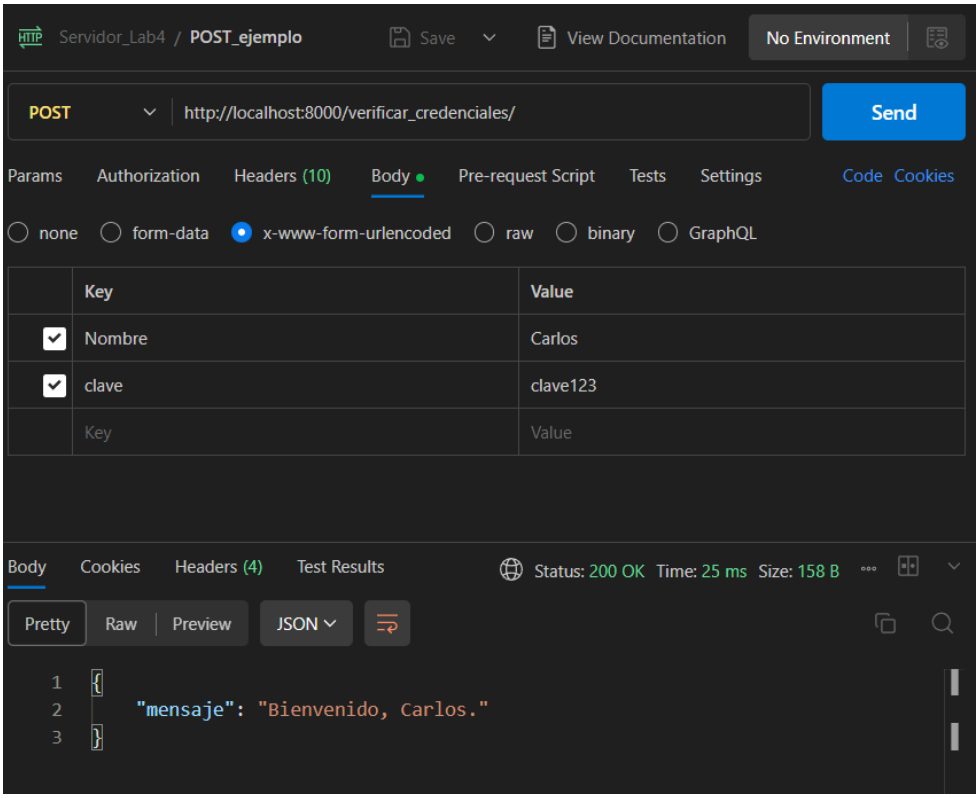

Figura 6 Prueba del servidor con PostMan.

En la Figura 6 se observa como por medio de PostMan se envió la solicitud POST al servidor, que en este caso su dirección es localhost o 127.0.0.1 porque se está haciendo la prueba desde el mismo host, y las credenciales a verificar son:

- Nombre: Carlos
- Clave: clave123

Como ya se vio anteriormente, estas credenciales están en la base de datos simulada, por lo cual la respuesta obtenida fue “Bienvenido, Carlos”.

Con esto se verifica el correcto funcionamiento de la primera prueba para el servidor de autenticación con FasApi, donde esta se vuelve una base fundamental para el desarrollo del proyecto, por lo que posteriormente se realizará la autenticación con las huellas dactilares almacenadas en una base de datos real con MongoDB, en vez de datos como nombre y clave en una base de datos simulada.

6. TROUBLESHOOTING

Problema: Optimización del almacenamiento de huellas dactilares

Descripción: El sistema estaba almacenando las huellas dactilares en un formato que ocupaba más espacio de lo necesario, lo que resultaba en un crecimiento rápido de la base de datos, especialmente con un gran número de usuarios registrados.

Diagnóstico y solución: Al revisar el esquema de almacenamiento, se observó que el formato utilizado para el hash de las huellas no estaba comprimido y algunos campos adicionales innecesarios estaban ocupando espacio.

Se implementó un algoritmo de compresión ligera para el hash de las huellas dactilares, reduciendo significativamente el tamaño de los datos sin perder precisión en la identificación biométrica.

Acción realizada y resultados: Implementación de una función de compresión para los datos biométricos antes de almacenarlos. Esta optimización redujo el uso de almacenamiento en aproximadamente un 30%, lo que permitió que el sistema escale de manera más eficiente a medida que aumenta el número de usuarios.

7. CONCLUSIONES Y PRÓXIMOS PASOS

- En el desarrollo de las interfaces se ha logrado una renderización de datos que están solo en la aplicación, un próximo paso es conectarla con otros puertos, es decir, con una base de datos y la esp32.
- En el desarrollo del primer prototipo del sistema de acceso biométrico para la Universidad del Cauca, se ha logrado una configuración efectiva de la base de datos noSQL en MongoDB, optimizando el almacenamiento de huellas dactilares a través de algoritmos de compresión. Además, las colecciones se diseñaron para garantizar un manejo eficiente de los registros de acceso y usuarios. Este enfoque ha permitido gestionar datos biométricos y registros de acceso de manera rápida y flexible. Un próximo paso será mejorar el cifrado de los datos biométricos almacenados para garantizar la privacidad y seguridad en la base de datos.

- La implementación hardware para este primer prototipo se ha quedado un poco corta debido a la ausencia de los dispositivos, se espera que para el próximo prototipo se puedan realizar pruebas con los dispositivos adquiridos y en particular probar la conexión a internet del WT32-ETH01.
- El uso de FastApi y Uvicorn para el servidor es un gran avance para el proyecto, debido que se podrán procesar múltiples solicitudes al tiempo y además no hay la necesidad de salirse del entorno de Python. El siguiente avance con el servidor deberá ser la autenticación de huellas dactilares y su conexión con la base de datos.

8. BIBLIOGRAFIA

[1] “GitHub - crisbol123/Control-de-acceso”. GitHub. [En línea]. Disponible: https://github.com/crisbol123/Control-de-acceso

[2] MongoDB, Inc., "MongoDB Python Drivers Documentation," MongoDB. [Online]. Available: https://www.mongodb.com/docs/drivers/python-drivers/. [Accessed: Sep. 25, 2024].

[3] MongoDB, Inc., "PyMongo Documentation," MongoDB. [Online]. Available: https://pymongo.readthedocs.io/en/stable/. [Accessed: Sep. 25, 2024].

[4] Python Software Foundation, "Python Documentation," Python.org. [Online]. Available: https://docs.python.org/3/. [Accessed: Sep. 25, 2024].

[5] “Popular python frameworks for building apis: A comprehensive guide”. Oscar Aguado Web. [En línea]. Disponible: https://oscaraguadoweb.com/python/popular-python-frameworks-for-building-apis-a-comprehensive-guide/

[6] “Crea una Aplicación con FastAPI para Python”. Kinsta®. [En línea]. Disponible: https://kinsta.com/es/blog/fastapi/

[7] Roy. “Building an API using FastAPI and Uvicorn”. DEV Community. [En línea]. Disponible: https://dev.to/blst-security/building-an-api-using-fastapi-and-uvicorn-3h79

[8] H. Lu. “FastAPI with uvicorn: A comprehensive tutorial | orchestra”. Orchestra | Modern Data Stack Orchestration + Observability. [En línea]. Disponible: https://www.getorchestra.io/guides/fastapi-with-uvicorn-a-comprehensive-tutorial

[9] “GitHub - khoih-prog/webserver_wt32_eth01: Simple ethernet webserver, HTTP/HTTPS client wrapper library for WT32_ETH01 boards using LAN8720 ethernet.

the webserver supports HTTP(S) GET and POST requests, provides argument parsing, handles one client at a time. it provides HTTP(S), MQTT(S) client and supports webserver serving from littlefs/spiffs. now supporting ESP32 core v2.0.0+". GitHub. [En línea]. Disponible: https://github.com/khoih-prog/WebServer_WT32_ETH01

[10] Angular, "Getting started with Angular: Your first app," 2023. [Online]. Available: https://angular.io/start. [Accessed: 20-Sep-2024].

[11] Angular Material, "Material Design components for Angular," 2023. [Online]. Available: https://material.angular.io/. [Accessed: 22-Sep-2024].