Conexión hardware con servidor

Conexión a la red con el microntralador 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.

**Informe de avance de prototipo 1, características y porcentaje de cumplimiento de requisitos**

*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 registro de datos biométricos y personales** para que el administrador pueda registrar ordenadamente todas las huellas (simuladas) y datos que identifiquen a cada integrante 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.
* **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 verificación de huellas y la autenticación.

***Características y porcentaje de cumplimiento de requisitos del primer prototipo:***

Este primer prototipo tiene las siguientes características:

* Sa
* As
* Servidor con FastApi y Uvicorn

El servidor de autenticación del sistema de acceso biometrico debe ser bastante robusto debido a que este recibirá muchas solicitudes de acceso al tiempo y estará funcionando todo el tiempo. Además, teniendo en cuenta que se escogió Python como lenguaje de programación debido a su alta compatibilidad y practicidad con modelos de machine learning, entonces se ve la necesidad de que el servidor también este basado en Python.

Con esto en mente se realizó una búsqueda exhaustiva para definir que frameworks se podrían emplear para tener un servidor que tolere altas demandas de solicitudes de acceso y además tenga una disponibilidad muy alta. Flask, Django, Node.js Golang (Go)

**FastAPI** es un framework moderno y muy rápido para Python, diseñado para manejar muchas solicitudes simultáneas de manera más eficiente que Flask, gracias a su arquitectura asíncrona.

* Conectividad entre el microcontrolador y el servidor

Diagrama

Descripción generada automáticamente

**1. Inicio - Inicialización del Microcontrolador**

* **Acción:** Configura el ESP32 para la comunicación serial.
* **Código:**

cpp

Copiar código

Serial.begin(115200);

* **Propósito:** Permite la comunicación entre el ESP32 y el ordenador para mostrar mensajes en el monitor serial, facilitando la depuración.

**2. Configuración de Ethernet**

* **Acción:** Inicia la conexión Ethernet usando la función ETH.begin() y configura el manejador de eventos para monitorear cambios en la conexión.
* **Código:**

cpp

Copiar código

ETH.begin(); // Inicia la conexión Ethernet

WiFi.onEvent(onEvent); // Registra los eventos de Ethernet

* **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.

**3. Verificar Estado de Conexión**

* **Acción:** El microcontrolador espera a que se establezca la conexión Ethernet y obtiene una dirección IP mediante DHCP.
* **Evento ARDUINO\_EVENT\_ETH\_GOT\_IP:**
  + **Código:**

cpp

Copiar código

case ARDUINO\_EVENT\_ETH\_GOT\_IP:

Serial.println("Conexión establecida con DHCP");

Serial.print("Dirección IP: ");

Serial.println(ETH.localIP());

eth\_connected = true; // Marca que la conexión está activa

break;

* + **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 eth\_connected a true.

**4. 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.

**5. 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.

**6. Enviar Datos al Servidor**

* **Acción:** Si la conexión Ethernet está activa, se envían datos al servidor usando la función sendPostRequest().
* **Código:**

cpp

Copiar código

sendPostRequest("DispositivoESP", "conectado");

* **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.

**7. Esperar 10 Segundos**

* **Acción:** Después de enviar la solicitud al servidor, el microcontrolador espera 10 segundos antes de volver a enviar datos.
* **Código:**

cpp

Copiar código

delay(10000); // Espera de 10 segundos

* **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.

**8. 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.