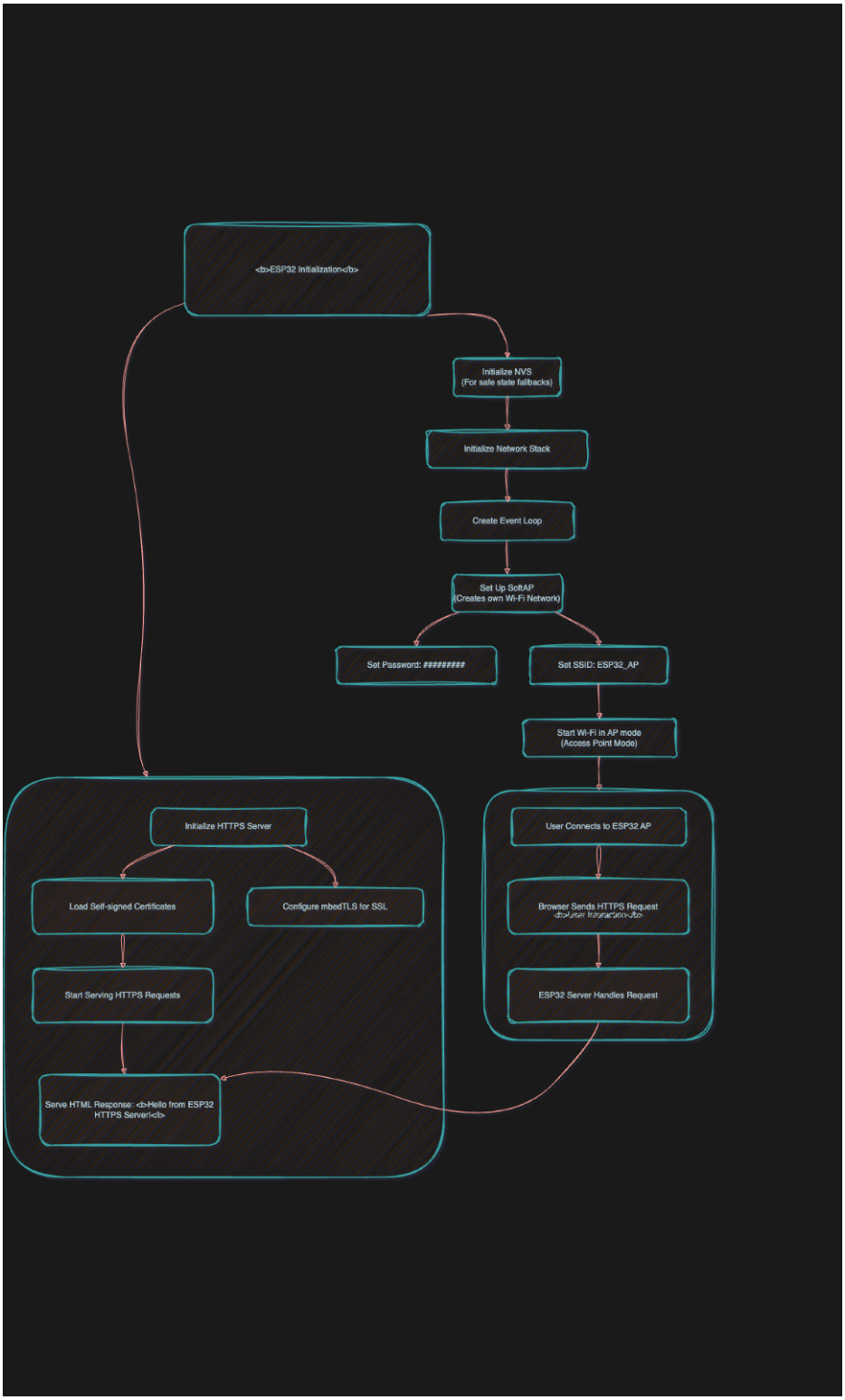
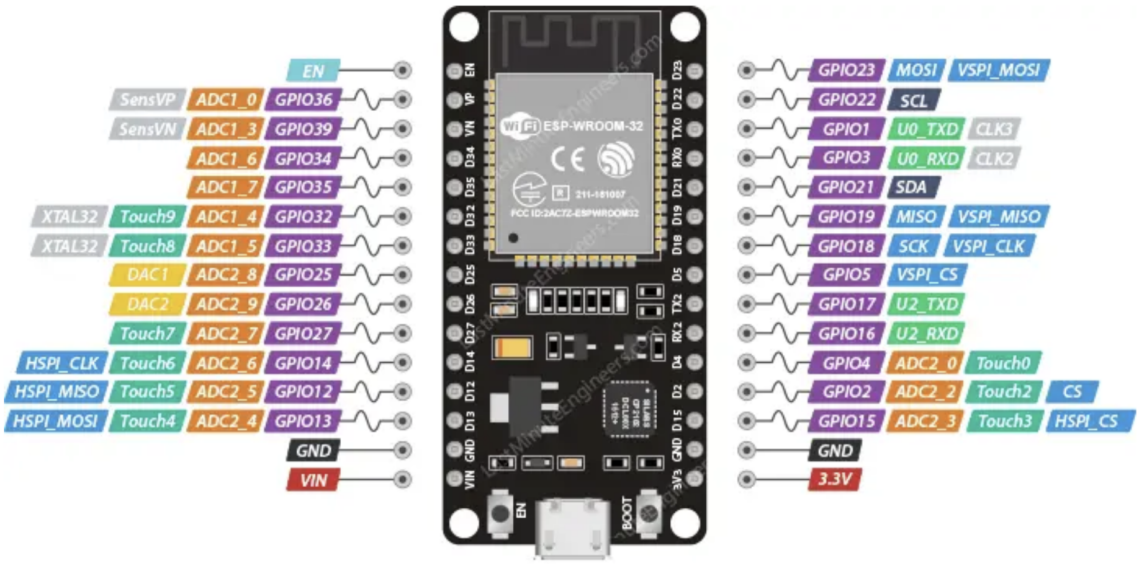
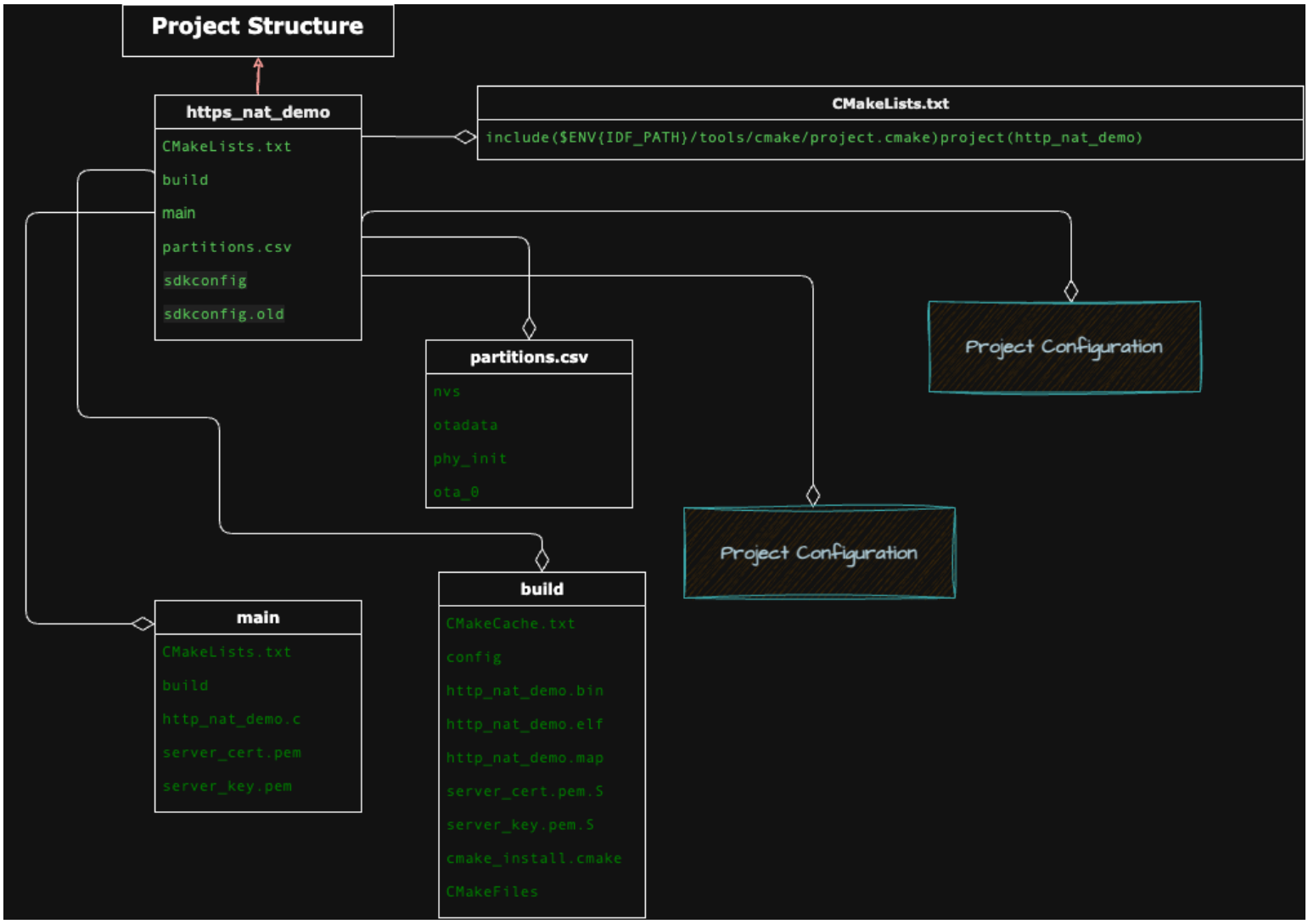
# ESP32 HTTPS Server with SoftAP Mode






## Project Structure:



## 1. Introduction

This project demonstrates the implementation of a secure HTTPS server on an ESP32 microcontroller operating in SoftAP mode. The ESP32 acts as a Wi-Fi access point, allowing other devices to connect directly and access a secure web interface without the need for an external router. This setup is especially useful for offline IoT deployments and secure communication over local networks.

## 2. System Architecture Overview

The architecture consists of the following components:
- ESP32 configured in SoftAP mode with SSID 'ESP32_AP'
- DHCP server enabled by default for IP allocation
- Static IP support if needed
- Embedded HTTPS server using self-signed certificates
- Secure endpoint serving HTML content via '/get' path

## 3. Functional Breakdown of APIs and Implementation

### 3.1 NVS Initialization

API: nvs_flash_init()
Initializes the Non-Volatile Storage used by the system. In case of no free pages or version issues, the storage is erased and re-initialized to maintain system integrity.

Here NVS is implemented as for safe state fallbacks without any error getting caused by any function.

```
esp_err_t wifi_init_softap(void) {
   esp_err_t ret=nvs_flash_init();
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret ==
ESP_ERR_NVS_NEW_VERSION_FOUND|| ret ==  ESP_ERR_NOT_FOUND ) {
       ret=nvs_flash_erase();
       if(ret!=ESP_OK)
        ESP_LOGE(TAG, "Failed to erasef, error code: %s", esp_err_to_name(ret));
       ret = nvs_flash_init(); // Reinitializing the  NVS
   };
   if(ret!=ESP_OK){
    ESP_LOGE(TAG, "Failed to initialize NVS, error code: %s", esp_err_to_name(ret));
       return ret;
   }
```

### 3.2 TCP/IP Stack Initialization

API: esp_netif_init()
Initializes the TCP/IP stack for networking. This is essential for setting up any network interface on the ESP32.

Initializes ESP-IDF's network interface abstraction, which is the layer that manages:

- IP assignment (DHCP/static)
- TCP/UDP communication setup
- Wi-Fi interface linkage
- Event handling for networking
- DNS, gateway, and routing logic

### 3.3 Event Loop Initialization

API: esp_event_loop_create_default()
Creates the default event loop for the system, allowing asynchronous event management such as Wi-Fi connection events.

### 3.4 Wi-Fi Configuration in SoftAP Mode

API: esp_netif_create_default_wifi_ap()
Sets up the ESP32 as a Wi-Fi Access Point. DHCP server is enabled by default for assigning IPs to connecting clients.
API: esp_wifi_set_mode(WIFI_MODE_AP)
Configures the ESP32 in AP mode.
API: esp_wifi_set_config(WIFI_IF_AP, &wifi_config)
Applies Wi-Fi credentials such as SSID and password.

### 3.5 HTTPS Server Configuration

API: httpd_ssl_start(&server, &conf)
Initializes and starts an HTTPS server with the given SSL certificate and private key.
API: httpd_register_uri_handler(server, &root_uri)
Registers an endpoint at '/get' which returns a static HTML response.

**Here SSL certificate is self-signed and created by openSSL**

**CMD:** openssl req -x509 -newkey rsa:2048 -keyout server_key.pem -out server_cert.pem -days 365 –nodes

extern const uint8_t server_cert_pem[] asm("_binary_server_cert_pem_start");

extern const uint8_t server_cert_pem_end[] asm("_binary_server_cert_pem_end");

extern const uint8_t server_key_pem[] asm("_binary_server_key_pem_start");

extern const uint8_t server_key_pem_end[] asm("_binary_server_key_pem_end");

PEM stands for Privacy Enhanced Mail, but in modern usage, it's a base64-encoded format used to store cryptographic data, such as:

- SSL/TLS certificates
- Private keys
- Public keys
- Certificate chains

A .pem file is a plain text file that contains encoded binary data, typically enclosed between specific headers and footers.Including the above statements we can access it through pointers.

### 3.6 DHCP and Static IP Configuration

By default, the ESP32 DHCP server provides IP addresses to connected clients. Static IP can also be configured by modifying the network interface configuration before initializing Wi-Fi.

### 3.7 HTTP Response Handling

Function: root_get_handler(httpd_req_t *req)
This function handles GET requests to '/get' and responds with a simple HTML page.

## 4. Partition Table:

Here I have customized the memory space by partitions.csv file:

Previously occupied memory address:

| Region | Start Address | Typical Size | Purpose |
|---|---|---|---|
| Bootloader | 1000 | 7000 | Loads the application |
| Partition Table | 8000 | 1000 | Defines flash layout |
| NVS | 9000 | 10000 | Used by ESP-IDF for NVS APIs |

For the reason, that the above bootloader and partition table start address NVS (Non-volatile storage) starts from 0x9000.

# Name, Type, SubType, Offset, Size

nvs, data, nvs, 0x9000, 0x10000

otadata, data, ota, 0xf0000, 0x2000

phy_init, data, phy, 0x120000, 0x1000

ota_0, app, ota_0, 0x130000, 0x180000

Now here i have increased the size of ota_0 since it will having the firmware code and if updates happens that  it will be through OTA(Over The Air).

## 5. Conclusion

In this project I have implemented  private wi-fi network making ESP32 as an accesss point where the other devices can connect and share the informations detected by ESP32 end devices(Sensors).This project provides a solid foundation for developing secure and self-contained IoT applications. By combining SoftAP mode with an HTTPS server, the ESP32 can host encrypted services locally, enabling secure device-to-device communication without relying on the internet.