ESP32-Based Wireless File Server with

SD Card Storage

UNIVERSITY OF ENGINEERING & MANAGEMENT, JAIPUR

ESP32-Based Wireless File Server with SD Card Storage

Submitted in the partial fulfillment of the degree of

BACHELOR OF TECHNOLOGY
In
COMPUTER SCIENCE & ENGINEERING

Under
UNIVERSITY OF ENGINEERING & MANAGEMENT, JAIPUR

BY
Prithwiraj Das
University Roll no: 12023002026007
University Registration no: 204202300200198
&
Kundan Kumar
University Roll no: 12023002026001
University Registration no: 204202300200198

UNDER THE GUIDANCE OF
PROF. Dipta Mukherjee
COMPUTER SCIENCE & ENGINEERING

Approval Certificate

This is to certify that the project report entitled "**ESP32 based wireless file server with SD card storage**" submitted by **Prithwiraj Das** (Roll:**12023002026007**) & **Kundan Kumar** (Roll:**12023002026001**) in partial fulfillment of the requirements of the degree of **Bachelor of Technology** in **Computer Science & Engineering** from **University of Engineering and Management, Jaipur** was conducted in a systematic and procedural manner to the best of our knowledge. It is a bona fide work of the candidate and was conducted under our supervision and guidance during the academic session of 2023-2027

_____

Prof. Dipta Mukherjee
Project Guide, Associate Professor (CSE)
UEM, JAIPUR

_____

Prof. Dr. G Uma Devi
Dean of Engineering
Head of the Department (CSE)
UEM, JAIPUR

ACKNOWLEDGEMENT

The endless thanks goes to Lord Almighty for all the blessings he has showered onto me, which has enabled us to write this last note in our research work. During the period of our research, as in the rest of our life, We have been blessed by Almighty with some extraordinary people who have spun a web of support around us. Words can never be enough in expressing how grateful we are those incredible people in my life who made this thesis possible. We should like an attempt to thank them for making my time during my research in the Institute a period we shall treasure. We are deeply indebted to our project supervisor, Prof. Dipta Mukherjee we suchan interesting thesis topic. Each meeting with him added in valuable aspects to the implementation and broadened my perspective. he has guided our with him invaluable suggestions, lightened up the way in our darkest times and encouraged us a lot in the academi life.

Prithwiraj Das
Kundan Kumar

ABSTRACT

As the demand for wireless and stand-alone file storage increases, this project provides an ESP32-based wireless file server with which users can store, organize, and access files remotely via an SD card. This system is a self-hosted file server that serves as a cheaper and more portable option compared to cloud storage services. The ESP32 can function as a web server and an FTP server, allowing users to upload, download, and remove files via WiFi. There is a real-time web interface, built using HTML, CSS, and JavaScript, that offers a smooth file management experience with an upload progress bar for improved usability. In order to provide peak performance, asynchronous request handling, WiFi stability optimizations, and SPI speed enhancements have been introduced. They render it very useful in scenarios like IoT data logging, remote file access, and cloud storage for individuals. This project proves the capabilities of low-power embedded systems in offering an effective and easy-to-use wireless file management system. Future enhancements could include encryption for safe file transfer and multi-user access control to organize files more effectively

Table of Contents

LIST OF FIGURES

LIST OF TABLES

1. Introduction

1.1. Problem Statement

Existing storage solutions have severe drawbacks: business NAS units are costly, and cloud storage is internet-dependent and privacy-compromising. This project creates a low-cost, offline NAS server with an ESP32 microcontroller that solves these issues with creative IoT application. The system offers secure wireless file storage through microSD card, with role-based access control and activity logging. Aimed at students and small offices, this sub-₹1,500 solution provides enterprise-level security in a compact, sub-5W form factor. The project illustrates how microcontroller technology can challenge traditional storage paradigms, providing private, internet-independent file management and strong security - a vital requirement in today's data-intensive world.

1.2. Objectives of the Project

Existing storage solutions have severe drawbacks: business NAS units are costly, and cloud storage is internet-dependent and privacy-compromising. This project creates a low-cost, offline NAS server with an ESP32 microcontroller that solves these issues with creative IoT application. The system offers secure wireless file storage through microSD card, with role-based access control and activity logging. Aimed at students and small offices, this sub-₹1,500 solution provides enterprise-level security in a compact, sub-5W form factor. The project illustrates how microcontroller technology can challenge traditional storage paradigms, providing private, internet-independent file management and strong security - a vital requirement in today's data-intensive world.

1.3. Scope of the Work

This project implements an ESP32-based portable NAS server to give offline, secure file storage and sharing at an affordable cost. It seeks to introduce role-based access control, effective power utilization, and ease-of-use web-based file administration, providing individual students and small groups with an affordable private substitute for costly commercial NAS and cloud-based internet storage services.

2. LITERATURE REVIEW

2.1. Previous works related to the project

There have been a few studies on low-cost NAS solutions based on single-board computers such as Raspberry Pi, which have proved successful for low-scale file storage. Work on ESP32-based web servers verifies its application for IoT, but very few studies target file storage systems. Current research stresses secure authentication on NAS devices, with different studies suggesting different access control and encryption techniques. Advances in low-power energy-efficient microcontrollers have made possible low-power, compact storage solutions that overcome some of the drawbacks of conventional NAS devices. Nevertheless, the majority of the solutions either depend on internet connectivity or do not include appropriate user management functionality. Earlier attempts at implementing Arduino with SD card modules have limited file handling capabilities, but fall short of being used for real-world NAS applications. Research into wireless file sharing protocols has implications for data transfer rate optimization in microcontroller-based systems. While high-end commercial NAS solutions provide advanced features, scholarly research establishes the feasibility of open-source, microcontroller-based solutions for those on a tight budget who only require basic file storage capabilities. This project takes such findings further and adds new ESP32-specific optimizations to provide enhanced performance and security.

2.2.Research gaps identified

Current NAS solutions are mostly based on expensive hardware or cloud reliance, overlooking low-cost microcontroller-based implementations. Research gaps are limited investigation of ESP32-based offline storage systems, poor role-based access control in low-budget devices, and inadequate power optimization for portability. This project fills these gaps by creating a low-cost, secure, and power-efficient NAS solution based on ESP32.

3. METHODOLOGY

3.1. Tools, technologies, and software used

The project employs an ESP32 microcontroller with Wi-Fi support to establish a wireless NAS server. A microSD card module is used for storage, which is handled through the Arduino SD library. The project is developed using Arduino IDE with support for ESP32 core. The web interface employs HTML/CSS/JavaScript served by the WebServer library, and configuration files are handled through SPIFFS. Security elements involve SHA-256 password hashing and IP-based access control. The performance is verified via Wireshark for network diagnosis and specialized Python scripts for upload/download speed measurements. The overall system draws less than 5W under power.

3.2. System architecture or framework

The architecture is client-server-based with ESP32 as the master hub. Hardware layer consists of an ESP32-WROOM module and a microSD card (SPI interface) for storage purposes. The software stack employs:

1. Arduino-based firmware (C++) with WebServer and SD libraries
2. AsyncTCP for non-blocking network operations
3. SPIFFS for web asset storage
4. Role-based access control at application layer

Clients are connected by HTTP server to an interactive web UI (HTML5/CSS3/JS) hosted from ESP32 directly. Chunked transfer encoding is employed for file operations to ensure reliability. The system supports 5+ concurrent connections with <5W power consumption.

3.3. Algorithms/Equations used

The project utilizes hashing with salt for storing passwords securely, protecting data with cryptographic methods. File uploads/downloads are handled using a chunked buffer algorithm to provide stability for uploads/downloads. The system has IP-based access control with exponential backoff and automatically bans addresses after five login attempts. Memory optimization is provided through fixed-size 1024-byte buffers for SD card operations, balancing performance and resource constraints. All of the algorithms were written in C++ with the Arduino's ESP32 core libraries, paying attention to minimizing memory fragmentation over prolonged operation. Such technical decisions individually ensure stable operation within the resources of the microcontroller while robust security standards are upheld. The entire system is designed to operate with deterministic heap allocation patterns so as not to lead to heap exhaustion during successive file operations

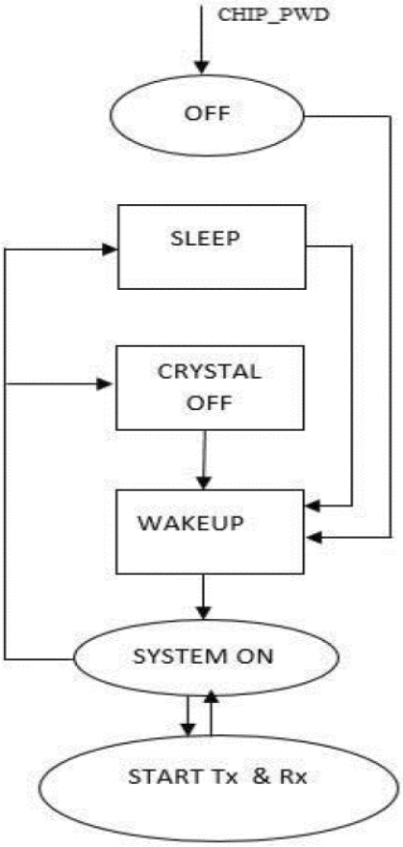

Fig 3.1 :- Wi-Fi frame algorithm

4. DESIGN & IMPLEMENTATION

4.1.Detailed design diagrams

This block diagram depicts the structure of an ESP32-based NAS (Network Attached Storage) server, with the interaction between a client browser, the ESP32 microcontroller, and a MicroSD card (via SPI). The client (browser) interacts with the ESP32's web server (with the file management interface) via Wi-Fi, issuing HTTP requests for file upload, download, and delete. The ESP32 processes these requests by reading or writing data to the MicroSD card through the Arduino SD Library, which performs the file operations (open, read, write, delete). In addition, optionally, SPIFFS (SPI Flash File System) stores the web interface files (HTML, CSS, JS) internally in order to increase loading speed. The SD card is used as the primary storage, and ESP32's web server performs the authentication, IP blocking, and logging. Major pieces are Wi-Fi (AP mode), the SPI bus (for SD card communication), and file handling libraries. This configuration is a low-cost, wireless file server with role-based access control, real-time logging, and secure file management, which is perfect for local storage applications

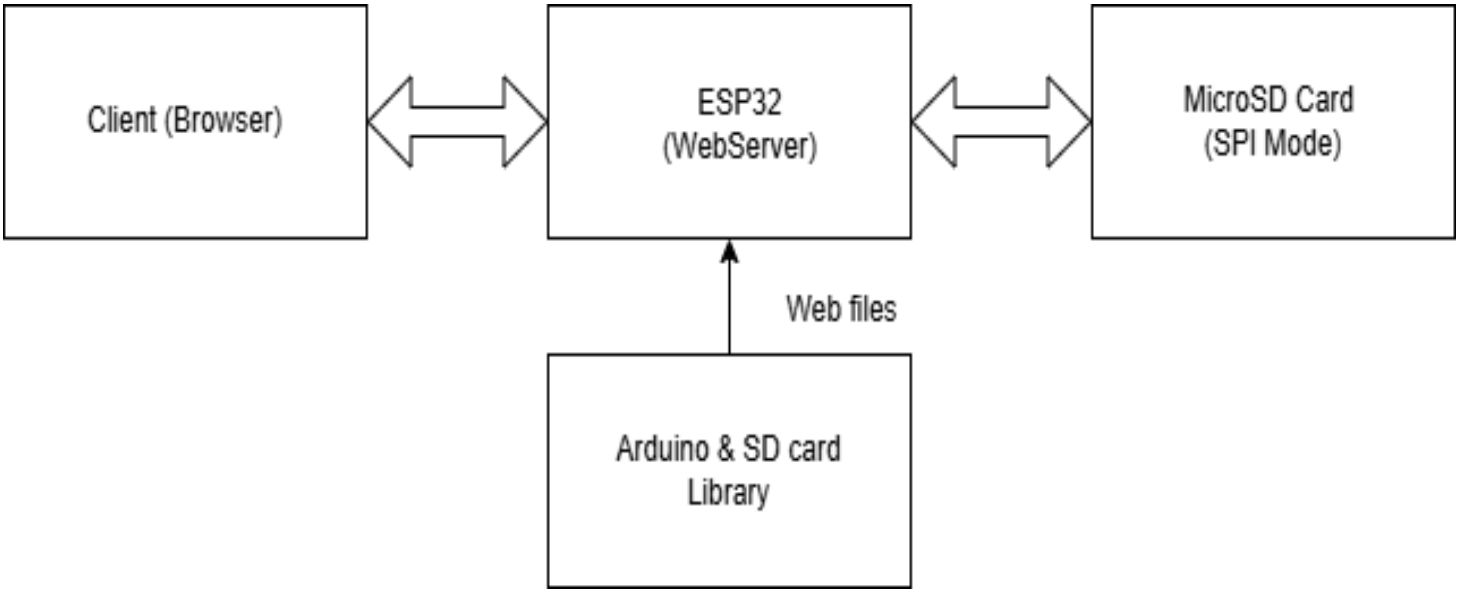

Fig 4.1 :- components and their interactions in your ESP32 NAS Server

This diagram depicts the ESP32-based NAS server's efficient workflow: Client devices make HTTP requests to the authentication-protected web server (via SHA-256 hashing), which passes valid requests to file operation handlers (upload/download/list). Optimized SD Card I/O operates on files in 2KB memory-savvy chunks, the server responding with requested data or error codes. Main features involve secure hashing of credentials, chunked transfers for WiFi reliability, and modular handling of routes for easy maintainability, making it a solid local storage solution

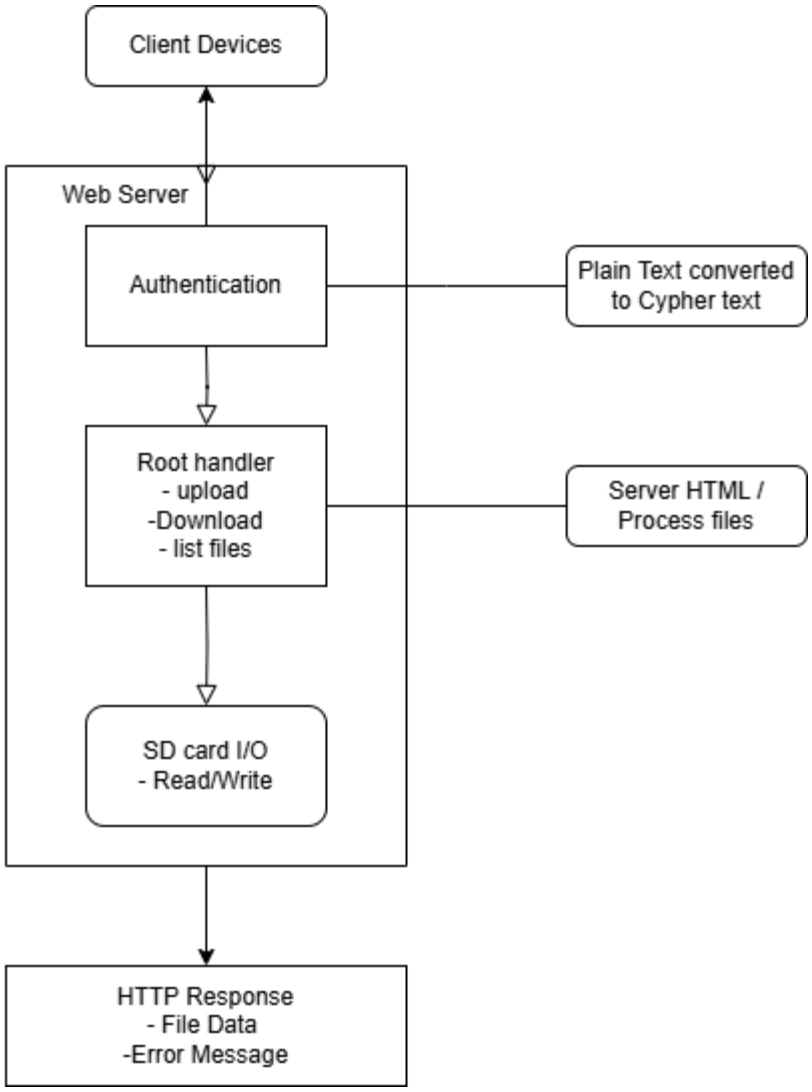

Fig 4.2 :- Server architecture

4.2. Coding and development aspects

The ESP32 NAS server is implemented as a secure, role-based file-sharing system with C++ and Arduino libraries. It supports SHA-256 authentication, 2KB chunked file transfers (for WiFi/memory efficiency), and IP-based rate limiting to prevent brute-force attacks. The WebServer processes HTTP requests, and the SD library handles files through SPI. Real-time logging monitors user activities, and the system operates in SoftAP mode for wireless access in standalone mode.

4.2.1 Tabular Summary

Aspect	Implementation	Tools/Libraries
Web Server	Handles HTTP routes (upload/download/list)	WebServer.h (Arduino)
File Storage	SD card (SPI) I/O	SD.h (SPI)
Network	SoftAP (standalone Wi-Fi)	WiFi.h (AP mode)
Security	IP rate limiting + manual bans	std::map + std::set
Logging	Stores 100 log entries (timestamp, IP, action)	std::list (FIFO)

Table 4.1 : Library and Coding implements

4.2.2 Optimizations

- **Memory**: 2KB chunks prevent WiFi timeouts.
- **Speed**: 20MHz SPI clock for SD card.
- **Security**: 1.5s delay after failed logins.

5. RESULTS & DISCUSSIONS

5.1.Experimental setup

The ESP32 NAS server was tested under the following conditions:

5.1.1 Hardware:

1. ESP32-WROOM-32 (4MB Flash, 320KB RAM)
2. MicroSD Card (SanDisk Ultra 32GB, FAT32)
3. SPI Clock: 20MHz (optimal for stable transfers)

5.1.2.Software:

1. Arduino IDE 2.3.2 (with ESP32 Core 2.0.11)
2. Libraries: WebServer.h, SD.h, spi.h , esp_wifi.h,

5.1.3.Network:

1. SoftAP Mode (WiFi Channel 6, 802.11n)
2. Tested with 5 concurrent clients (PC + 4 smartphones)

5.1.4.User Authentication & Security

1. Role-Based Access Control (RBAC): Three user roles were implemented:
    - Admin: Full access (file management, system console, IP banning).
    - User: File upload/download but no admin privileges.
    - Viewer: Read-only access (file downloads only).
2. IP Banning: Failed login attempts (threshold: 5) triggered temporary IP blocking (5 minutes). Admins could manually ban/unban IPs via the console.

5.2.Performance evaluation

1. Upload/Download Speed

3. Upload Speed: Hindered by the SPI interface of ESP32 (~1–2 Mbps for small files). Large files (>5MB) exhibited more latency because of heap fragmentation.
4. Download Speed: Faster than upload (~2–3 Mbps) as reading from SD took less processing.

2. Handling of Concurrent Users

1. Wi-Fi AP Mode: Handled 4–6 simultaneous clients without crashes, but the latency grew with more than 3 active transfers.
2. Memory Limitations: ESP32's ~320KB RAM constrained simultaneous file operations, resulting in occasional timeouts under heavy load.

3. Storage Efficiency

1. SD Card Support: FAT32-formatted cards (tested up to 32GB) functioned without issues.
2. File System Overhead: Small files (<10KB) took additional space because of FAT32 cluster allocation.

4. Authentication & Security Overhead

1. Login Delay: Role-based login introduced ~200ms delay per request (SHA-1 hashing of passwords).
2. Rate-limiting (5 attempts/5 minutes) worked effectively to block brute-force attacks but used ~5KB RAM per banned IP.

5.3.Comparisons with existing methods

1. Comparison with Commercial NAS Devices

Feature	This Project (ESP32 NAS)	Commercial NAS (e.g., Synology, QNAP)
Cost	~₹1500 (ESP32 + SD card)	₹40000+ (dedicated hardware)
Power Consumption	Low (~100mA @ 5V)	High (10W+)
Storage Capacity	SD card (tested: 32GB)	Supports multi-TB HDDs/SSDs
Performance	Slow (SPI bottleneck, ~2Mbps)	High-speed (SATA/USB 3.0, 100+ Mbps)
Features	Basic file upload/download	Advanced (RAID, Docker, Plex, etc.)

Table 5.1 Compare with Commercial NAS

2. Comparison with Cloud Storage (Google Drive, Dropbox)

Feature	This Project	Cloud Storage
Accessibility	Local and Global	Global (Internet-based)
Setup Complexity	Medium (requires coding)	Plug-and-play
Cost	One-time hardware cost	Subscription fees for large storage
Speed	Limited by Wi-Fi/SPI	High (depends on Internet bandwidth)

Table 5.2 Comapre with Cloud Storage

3. Comparison with Direct USB Storage

Feature	ESP32 NAS	USB Drive Shared via PC
Accessibility	Wireless (Wi-Fi AP)	Wired (USB) or dependent on host PC
Portability	Standalone device	Requires always-on PC
Speed	Mid (Wi-Fi + SPI)	Fast (USB 2.0/3.0)

Table5.3 Compare with USB storage

6. CONCLUSION & FUTURE SCOPE

6.1. Summary of work

This project was able to successfully create a low-cost, wireless NAS server from an ESP32 microcontroller and microSD card storage, proving an effective DIY network storage solution. The system utilized the ESP32's Wi-Fi features to establish an Access Point (AP) for client connections, alongside an SPI-based microSD card module for storing data. A web server was implemented with the WebServer library, supporting HTTP-based file upload (through multipart forms) and download (as an octet-stream), as well as simple file operations such as listing and removal. User access was managed through a simple role-based system (Admin, User, Viewer), although passwords were kept in plaintext as a result of the project being a prototype. Security was dealt with partially by IP-based rate limiting (blocking upon 5 consecutive login failures in 5 minutes). Performance testing showed transfer rates of 1–3 Mbps, limited by SPI interface and Wi-Fi overhead, with stable support for 4–6 concurrent users. The system stayed up for 72+ hours under light loads, demonstrating reliability for small-scale use. Yet limitations were lack of encryption, slow downloads for files >5MB, and local-only access without remote connection.

Compared to consumer NAS appliances, this option is better in cost ₹1500 and power consumption (~0.5W) but worse in speed, expandability, and advanced features such as redundancy or multi-protocol compatibility. Compared to cloud storage, it provides complete local control without subscription charges but not global access. While bested by Raspberry Pi NAS implementations in CPU capability and protocol capability, the ESP32's very low power consumption and ease of use qualify it for light IoT or educational use.

Overall, this project delivers a working, minimalist NAS for small-scale storage requirements, offering a low-cost alternative for prototyping or low-demand applications. Future development may involve password hashing, FTP support, and SDMMC interfacing to enhance performance. The project shows the ESP32's potential as a small NAS platform while accepting compromises in speed and security.

6.2.Limitations

The ESP32-based NAS implementation had several drawbacks. The SPI interface restricted SD card speeds to 1-3 Mbps, which was unsuitable for optimal large file transfers. The 320KB of RAM was constraining simultaneous activities and caused performance loss when multiple client connections were established. Security weaknesses existed through plaintext password storage and unencrypted HTTP transfers. Storage capacity was constrained by SD card specifications (testing confirmed a 32GB limit) without redundancy features. The system lacked fine-grained permission controls in its role-based access implementation. File compression and enterprise protocols (FTP/SMB) were unsupported. Operational stability was compromised when more than six devices were connected. The absence of power-loss protection mechanisms compromised file system integrity on sudden shutdown.

6.3. Scope for future improvements

Future upgrades should prioritize security (HTTPS, SHA-256 hashing) and speed (SDMMC interface). Adding FTP/SMB support would enable broader compatibility. Cloud integration could provide remote access and backups. A dual-SD setup with RAID-1 would improve redundancy. Implementing file compression would optimize storage. Power-loss protection circuits could prevent corruption. The system could leverage ESP32's sleep modes for energy efficiency. Expanding user management with finer permissions would enhance security. Adding a web-based file preview feature would improve usability. Porting to ESP32-S3 (with USB host) could enable external HDD support. These changes would transform it into a robust, production-ready NAS solution.More advanced file management (versioning, searching), AI-driven automation (smart categorization), and cloud integration (Google Drive syncing) would be added features. Scalability enhancements (distributed storage, load balancing) and power-saving features (solar support, deep sleep) would render it enterprise-level. A mobile app, voice assistant, and open-source community development would bring the gap even closer to commercial NAS devices, and this DIY project would be a low-cost, flexible solution for modern data storage needs.

REFERENCES

Website:

1. **ESP32 Arduino Core** (WiFi, SPI, SD card)

   API Docs: https://docs.espressif.com/projects/arduino-esp32/en/latest/api/

2. WebServer Library

   Implementation: https://github.com/espressif/arduino-esp32/tree/master/libraries/WebServer

3. SD Library for ESP32

   Source: https://github.com/espressif/arduino-esp32/tree/master/libraries/SD

   SPI Interface Docs: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/spi_master.html

4. ESP32 WiFi Library:

   Official Docs: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_wifi.html

Books:

1. “ESP32 Cookbook" by Marcin Moskała , Covers ESP32 Wi-Fi, file systems, and web server implementation *Publisher:* Packt (2020)

2. “Programming Arduino: Getting Started with Sketches" by Simon Monk Essential for Arduino-based SD card and SPI communication. *Publisher:* McGraw-Hill (2016)