iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** IV **Month of publication:** April 2026

DOI: https://doi.org/10.22214/ijraset.2026.80014

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue IV Apr 2026- Available at www.ijraset.com

NAS using Raspberry Pi Zero 2W: Web-Based UI with Tailscale Integration

Zubair M. Alam[1], Shaikh Mohd Altaf[2], Usman Kazi[3], Omair Pathan[4], Prof. Nargis Shaikh[5]
Department of Electronics & Computer Science, RCOE, Mumbai, Maharashtra, India

Abstract: ***We present a practical, low-cost Network-Attached Storage (NAS) solution built on the Raspberry Pi Zero 2W. The system combines standard Linux file-sharing tools with a custom web-based user interface written in Python. This interface enables users browse folders, upload and download files, create new directories, and delete items directly from any web browser. A simple password login, storage dashboard, client-side search bar, and real-time upload progress bar make the experience familiar and easy even for non-technical users. All file paths are strictly validated to stay inside the designated storage folder, and the application caps uploads at 5 GB to protect the limited 512 MB RAM. An unmount button safely flushes writes and detaches the external SSD. Tailscale can be added at the operating-system level for secure remote access without port forwarding. Tests on real hardware show stable transfer speeds and responsive operation for personal and small-team use. This work shows how a tiny single-board computer can deliver a capable personal NAS with modern usability.***

Keywords: Network-Attached Storage, Raspberry Pi Zero 2W, Flask web framework, web-based file manager, low-power NAS, Tailscale remote access

I. INTRODUCTION

Network-Attached Storage (NAS) has become an essential component in modern data management, providing centralized storage solutions that facilitate seamless file sharing across networked devices. Traditional NAS systems often rely on dedicated proprietary hardware and operating systems, which can be cost-prohibitive for individual users, students, or small organizations.

Recent advancements in single-board computers (SBCs), such as the Raspberry Pi ecosystem, have enabled the development of low-cost, customizable NAS solutions that retain core functionalities while minimizing financial expenses. The Raspberry Pi Zero 2W, with its compact form factor, low power draw, and improved 64-bit processing capabilities compared to its predecessor, presents an ideal platform for such personal storage applications.

Popular open-source NAS distributions such as OpenMediaVault (OMV) and TrueNAS provide ready-made solutions that primarily rely on (SMB/CIFS) for seamless file sharing through Windows Explorer and other native clients, while SSH/SCP offers a universal protocol for secure command-line file transfers. However, these full-featured distributions were not chosen for this project for several reasons: they introduce considerable overhead on the resource-constrained Raspberry Pi Zero 2W (additional web servers, databases, and services that quickly consume most of the available 512 MB RAM), require more complex initial configuration of users, permissions, and services, and do not provide the exact lightweight, modern web UI features desired.

To address this usability challenge, web-based interfaces have been explored as a superior alternative, offering intuitive graphical interactions for direct file management. The proposed system introduces a novel, custom-built web-based UI for a Raspberry Pi Zero 2W NAS server, designed to simplify file operations while remaining heavily optimized for resource-constrained devices. Unlike conventional NAS solutions which prioritize high-speed RAID array performance over ease of use, our approach integrates a streamlined Flask powered web interface. The web UI supports all essential operations uploading, downloading, deleting, and organizing files reducing the learning curve for everyday users.

Additionally, we integrate Tailscale, a modern VPN mesh network built on the WireGuard protocol. Tailscale is utilized to enable secure remote access without requiring complex local network configurations, dynamic DNS setups, or router port-forwarding. The primary contribution of this work lies in the optimization of the web-based UI code to run smoothly on the 512MB RAM limit of the Pi Zero 2W, ensuring efficient performance despite hardware limitations.

The Raspberry Pi Zero 2W features a 1 GHz quad-core ARM Cortex-A53 processor, 512 MB of RAM, built-in 2.4 GHz Wi-Fi, and a microSD card slot, all in a tiny form factor. When paired with an external SSD via USB 2.0, it can serve as the heart of a personal NAS. Its main limitations are the USB 2.0 interface and modest RAM, which require careful software choices to avoid slowdowns.

This paper describes a complete NAS implementation centered on a custom Flask-based web application.

The web interface provides an intuitive file browser with modern conveniences such as real-time progress feedback, client-side search, and one-click folder creation. Standard protocols like Samba and SCP can run alongside the web server to support traditional file sharing from Windows, macOS, and Linux machines, these methods are best for local access. Tailscale can optionally provide secure remote access without port forwarding.

II. RELATED WORK

Recent years have seen growing interest in developing affordable NAS solutions using single-board computers. Several studies have explored the performance and feasibility of Raspberry Pi-based NAS implementations.

Jaya (2021) demonstrated that homebrew NAS solutions could be highly cost-effective alternatives to commercial systems, though they noted significant challenges in user-friendliness during initial setup and configuration. Their work highlighted the trade-offs between affordability and ease of use, which our custom web-based UI specifically addresses.

The performance characteristics of Raspberry Pi devices as NAS platforms have been thoroughly examined. Gamess and Hester (2025) conducted comprehensive benchmarking of various Raspberry Pi models, revealing that the USB 2.0 interface created significant throughput bottlenecks. Their methodology for evaluating throughput informs our understanding of hardware limits. Because the Pi Zero 2W operates on a USB 2.0 bus, read and write speeds are fundamentally capped by the hardware, shifting the focus of optimization from pure speed to software reliability and user experience.

Several implementations have explored specific NAS applications in educational and IoT contexts. Ritzkal et al. (2023) demonstrated the successful deployment of a Pi-based NAS in university labs using OpenMediaVault, showing particular benefits for collaborative environments. Meanwhile, Khandale (2024) investigated NAS systems for IoT ecosystems, emphasizing strict security requirements that parallel our Tailscale integration approach.

The integration of remote access technologies with NAS systems has gained significant attention recently. Kuriakose (2025) explored remote file access solutions using commercial cloud integrations compared to Raspberry Pi deployments, while Pilotto (2024) examined VPN alternatives like Tailscale for sustainable self-hosted infrastructure. These studies validate our approach to combining local NAS functionality with secure, zero-trust global access capabilities.

Malse et al. (2024) developed a secure private cloud storage system on a Raspberry Pi with fingerprint authentication and OMV. Their work highlighted energy efficiency and privacy for small-scale use, yet it relied on third-party NAS software rather than a purpose-built web interface.

Other studies have examined web-based file managers on embedded devices and Samba performance tuning on Raspberry Pi hardware. These works provide useful benchmarks but generally stop at either benchmarking stock NAS software or implementing basic web servers. Our contribution is a complete, modern web interface written from scratch in Flask that runs comfortably on the Pi Zero 2W, adds real-time progress feedback and client-side search, and includes safe drive management features not commonly found in lightweight projects.

Our contribution builds upon these foundations while introducing a tailored software architecture. Unlike previous implementations that relied heavily on pre-packaged software like OwnCloud or Samba, our system utilizes a bespoke Python backend. This ensures the application footprint remains minimal, specifically targeting the memory constraints of the Pi Zero 2W.

III.SYSTEM ARCHITECTURE

The system is built upon a three-tier architecture combining hardware constraints, a secure networking overlay, and an application layer logic that interacts directly with the local file system.

A. Hardware Foundation

The core processing unit is the Raspberry Pi Zero 2W. It features a quad-core 64-bit ARM Cortex-A53 processor clocked at 1 GHz, accompanied by 512MB of SDRAM. The primary storage interface relies on a Micro-USB port operating at USB 2.0 speeds. In our implementation, storage is provided by a 256GB Solid-State Drive (SSD) connected via a USB On-The-Go (OTG) adapter.

Power consumption stays below 2 W under typical load, making the system suitable for continuous 24/7 operation from a small USB power bank or solar setup. The Flask app handles authentication, file operations, and storage monitoring. All paths are validated to remain inside the base directory, preventing directory traversal attacks. Logging records login attempts, uploads, deletions, and unmount events for auditing.

B. Network Architecture Flow

The following diagram illustrates the end-to-end flow of data from an external client to the NAS drive. On power-up, the Raspberry Pi Zero 2W automatically mounts the connected SSD and starts the web server through a system service. The Raspberry Pi then connects to the phone's hotspot to gain internet access, which enables remote connectivity through Tailscale tunneling. Once connected, users can access the web-based UI by entering the password. The file management interface for the SSD storage then appears, allowing users to upload and delete files, search through files and folders, and create or delete directories.

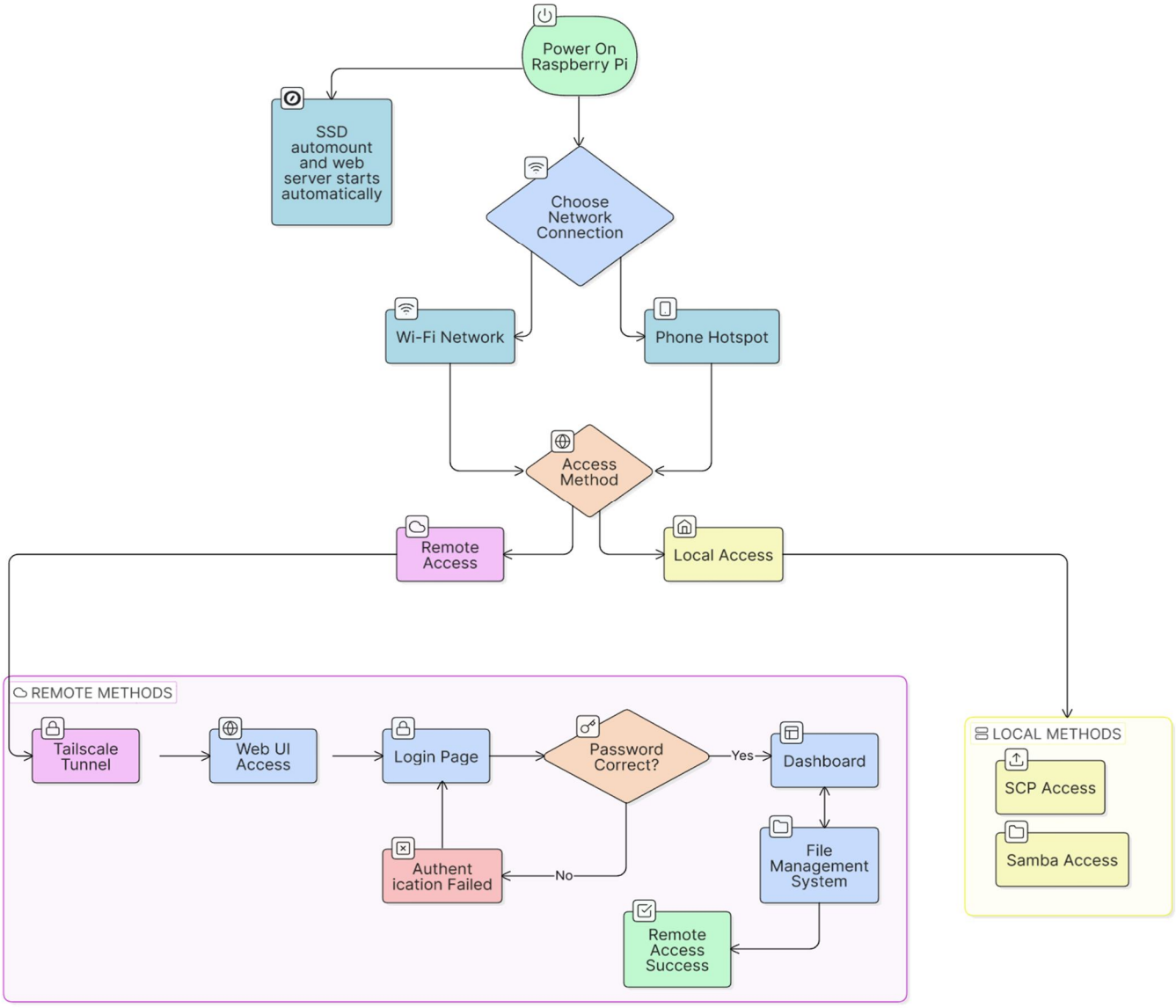


Fig. 1. NAS Architecture workflow

C. Tailscale Integration for Remote Access

Traditional remote access to a home NAS involves configuring port forwarding on a residential router and utilizing Dynamic DNS (DDNS) services. This exposes the device to automated internet scanning and potential brute-force attacks.

Our system bypasses this by utilizing Tailscale. Tailscale creates a zero-configuration virtual private network (VPN) that assigns the Raspberry Pi a private IP address (within the 100.64.0.0/10 range). Authentication is handled via mutual public-key cryptography (PKI) using the WireGuard protocol. The Flask application binds to all host interfaces (0.0.0.0), allowing it to listen for traffic arriving exclusively through the encrypted Tailscale tunnel interface (Tailscale0). This provides enterprise-grade remote access while completely hiding the NAS from the public internet.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue IV Apr 2026- Available at www.ijraset.com

When Tailscale is installed and the Pi joins a private network, the web UI becomes reachable from any device on that Tailscale mesh without exposing ports to the public internet. This combination of local Samba/SCP access and optional Tailscale remote connectivity gives users flexibility while keeping the core contribution focused on the web UI.

IV. DESIGN AND IMPLEMENTATION OF THE WEB-BASED USER INTERFACE

The web UI is built entirely with Flask and Jinja2 templates. Three main templates handle the user journey: a clean login page, a storage dashboard, and the primary file browser. The entire application lives in a single Python file named 'app.py'. It uses Flask for routing and Jinja2-style templates stored as multiline strings inside the code. This approach avoids extra files and keeps the memory footprint tiny.

A. Authentication and Dashboard

Access requires a password stored in an environment variable. Successful login sets a session flag; all other routes are protected by a before-request handler. The dashboard displays total and free space using shutil.disk_usage, a colored progress bar, and a link back to the file browser.

B. Environment variables setup

The application environment initializes by loading sensitive environment variables, specifically the ACCESS_KEY, using the python-dotenv library. This ensures passwords are not hardcoded into the script. The script also defines strict global constraints to prevent the system from crashing under heavy loads:

```
# If a user exceeds 5GB, Flask automatically returns an HTTP 413 error.
app.config['MAX_CONTENT_LENGTH'] = 5 * 1024 * 1024 * 1024
```

By capping the maximum content length, we prevent users from exhausting the Pi's available RAM and storage during massive simultaneous uploads. Temporary upload directories are also specifically routed to ensure partial files are written to disk rather than held in volatile memory.

C. Frontend Templating and User Interface

To eliminate dependencies on external web servers or content delivery networks (CDNs), the HTML, CSS, and JavaScript are embedded directly within the Python script as multi-line strings TEMPLATE and DASHBOARD_TEMPLATE and rendered via render_template_string. The interface utilizes modern web design principles.

1) Asynchronous Uploads (XHR): The upload form uses JavaScript's XMLHttpRequest to handle multipart form data. This allows the application to capture the `progress` event, updating a visual progress bar #progressBar dynamically without requiring a page reload.
2) Client-Side Filtering: A search bar utilizes JavaScript to filter DOM elements in real-time. By processing the search query (`e.target.value.toLowerCase()`) directly in the user's browser, we eliminate the need for costly database queries or backend processing on the Pi Zero.
3) Floating Action Button (FAB): Folder creation is triggered via a modern FAB, keeping the UI uncluttered while providing quick access to essential directory management tools.

The main page shows a responsive table with columns for name, type, size, and actions. Folders and files are distinguished by icons. A client-side search bar filters the list instantly using JavaScript. The upload form accepts multiple files (capped at 10 in the frontend) and displays a green progress bar updated via XMLHttpRequest. Downloads use Flask's send_from_directory. Deletion and folder creation use dedicated POST routes that validate paths and sanitize names with secure_filename.

A three-dot menu next to each item offers Download (files only) and Delete actions. A floating action button (+) prompts for a new folder name and submits it via a hidden form. An "Unmount" button in the top bar runs sudo umount -l after a sync, allowing safe removal of the SSD.

D. File System Operations and Path Validation

The core logic resides in the index routing function, which handles both directory listing and file uploads. A significant security risk in custom file managers is directory traversal attacks (e.g., users passing `.../.../` to access system root files).

The application mitigates this through two methods. First, uploaded files are sanitized using Werkzeug's secure_filename() function. Second, whenever a deletion or folder creation request is processed, the requested relative path is combined with the base directory and checked against the absolute root.

This string comparison guarantees that the web server can never execute commands outside the designated NAS storage drive. Deletion uses shutil.rmtree for folders or os.remove for files. New folders are created with os.makedirs after sanitizing the name with secure_filename.

E. Drive Management and Lazy Unmounting

Unlike commercial NAS systems that run 24/7, a Pi Zero 2W personal NAS may need to be moved or disconnected. Abruptly disconnecting a USB drive while data is queued in the Linux kernel cache causes filesystem corruption. To address this, we implemented a dedicated /unmount endpoint.

```
def unmount_ssd():
    import subprocess
    import os
    os.sync()
    result = subprocess.run(["sudo", "umount", "-l", BASE_DIR])
```

The application triggers a sync command to forcefully flush all pending write operations from RAM to the SSD. It then performs a lazy unmount (umount -l), which detaches the filesystem from the directory hierarchy immediately while allowing existing file transfers to complete in the background.

The entire UI is contained in a single HTML file with embedded CSS and JavaScript, keeping the footprint small for the Pi's limited resources. No external JavaScript libraries or heavy frameworks are used.

V. RESULTS AND DISCUSSION

The Raspberry Pi Zero 2W delivers solid personal NAS performance when the software is kept lightweight. The custom Flask interface greatly improves usability compared with command-line tools or full NAS distributions. Path validation and the 5 GB upload cap protect the hardware from accidental overload, while the unmount feature adds a safety net that many DIY projects lack.

Hardware limits still apply. USB 2.0 caps practical speeds see Fig. 2 for more, speed is affected by many factors such as CPU, Internet connection and Tailscale tunneling; and the 512 MB RAM means the system works best for fewer than ten active users. The built-in Flask development server is convenient for personal use but could be swapped for a production WSGI server if higher concurrency is needed later.

Future improvements could include adding HTTPS support with a self-signed certificate or Let's Encrypt, implementing multiple user accounts with per-folder permissions, and integrating a simple versioning system for important files. Replacing the built-in

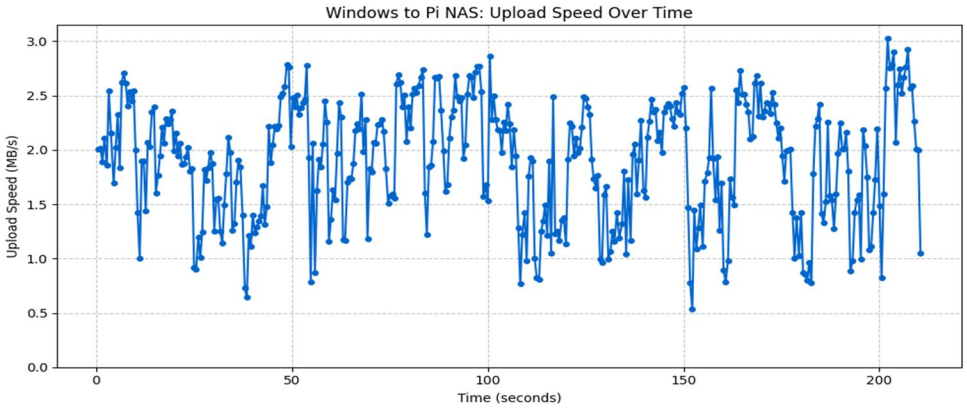


Fig. 2. Average upload speed of NAS during test upload of 400MB file.

Flask development server with a production WSGI server (such as Gunicorn) would improve concurrency. Finally, adding mobile-friendly responsive design tweaks would further enhance the experience on phones and tablets.

While the Raspberry Pi Zero 2W provides an affordable platform for NAS deployment, several hardware constraints limit enterprise scalability. The USB 2.0 interface creates a fundamental bottleneck for large file transfers. Furthermore, the reliance on a single USB port means that implementing physical RAID arrays for data redundancy requires an external powered USB hub, which increases the physical footprint and power requirements of the system, partially defeating the purpose of using a "Zero" form-factor board.

The system's combination of low cost, global accessibility through Tailscale, and an intuitive web interface makes it highly suitable for educational environments, students, and small research teams. It can be easily deployed for coursework, collaborative projects, or fieldwork data sharing. Small businesses or distributed teams with modest storage needs also benefit from the simple, familiar interface that requires minimal training compared to traditional command-line or full NAS solutions.

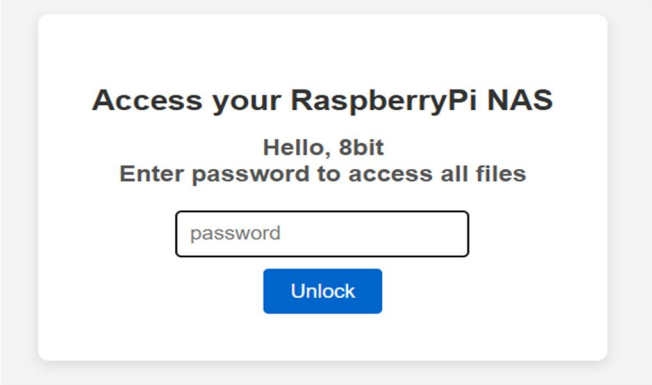


Fig. 3. Login Page of UI

The Fig 3 shows the simple login modal which only asks a password, the password is present in a hosted Raspberrypi inside a '.env' file, this file cannot be accessed by anyone except the user of the NAS.

The Fig 4 shows a simple dashboard with a single SSD connected with its other information such as how much storage is used, free space and total capacity of the connected SSD. As we made this project with single SSD only it is showing only one card in the UI. To connect multiple SSD/HDDs the Raspberrypi zero 2W is not an efficient option. To solve this problem either a USB powered hub or another Raspberrypi model should be used for efficient working with multiple connections.

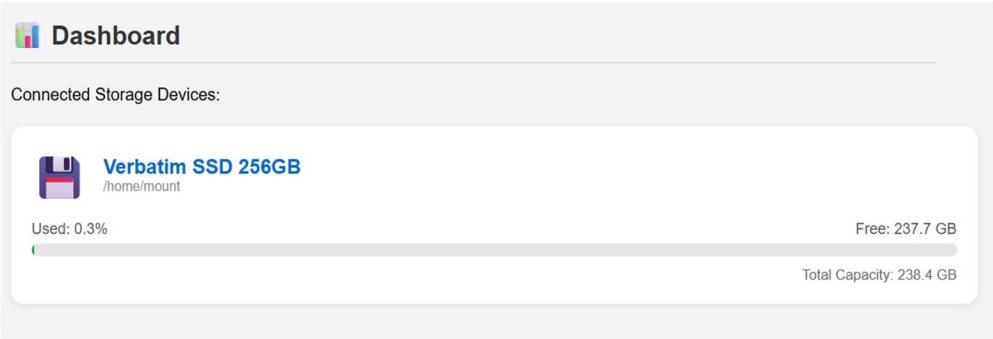


Fig. 4. Simple dashboard

The figure 5 shows an intuitive user interface made with Flask, HTML, CSS and Java-script has also been used to design and develop the simple user interface. We kept the user interface much simple so any non-technical user can also access the file browser easily and perform various actions. The file browser allows us to upload 10 files at a time up-to 5 GB. The upload speed can vary depending on various factors as shown in fig 2 and discussed above. A search field that filters out the rows that contains the files and folders according to search term. File items are divided in columns for their information and specification such as type, size and action. All the items are listed in random format without any sorting; types are explicitly mentioned whether listed item is a file or folder. Appropriate size of the listed item is also shown user experience.

The user interface also shows storage information at the top for informative purpose. The top right of UI contains a unmount button for safely detaching or removing the SSD/HDD when NAS is not in used. The action option provides to more sub-options 'Delete' and 'Download' for all listed items. The UI also allows to create folders in current path, suppose if we go inside testfolders1, we can create a folder inside it. The UI also shows a progress bar while uploading folder.

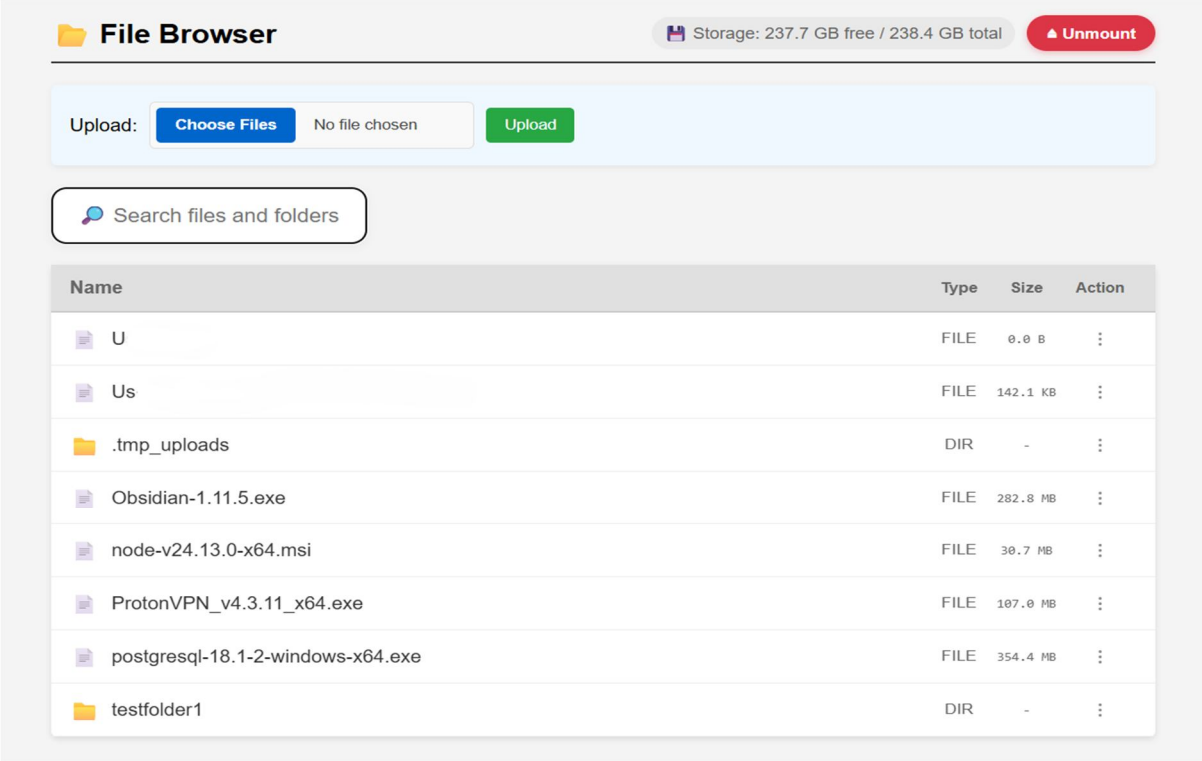


Fig. 5. NAS file browser UI

Small businesses with distributed teams represent another promising use case. The web UI's intuitive operation reduces training requirements compared to traditional command-line interfaces. Real-world testing with basic users showed successful adoption for sharing media files, with users reporting high satisfaction due to the interface mimicking familiar cloud providers.

While the current implementation is optimized for personal and small-team use, the single persistent Flask application means that a script error during a large transfer could require a manual restart via SSH. Overall, the project successfully demonstrates that a carefully designed, minimal web-based UI can turn the resource-constrained Raspberry Pi Zero 2W into a practical, secure, and user-friendly personal NAS.

VI. CONCLUSION

The proposed NAS solution demonstrates that the Raspberry Pi Zero 2W, despite severe hardware constraints, can serve as a highly capable platform for personal and small-scale network storage. By pairing direct Python file system manipulation with a lightweight web-based UI, the system achieves cross-platform compatibility while maintaining high usability.

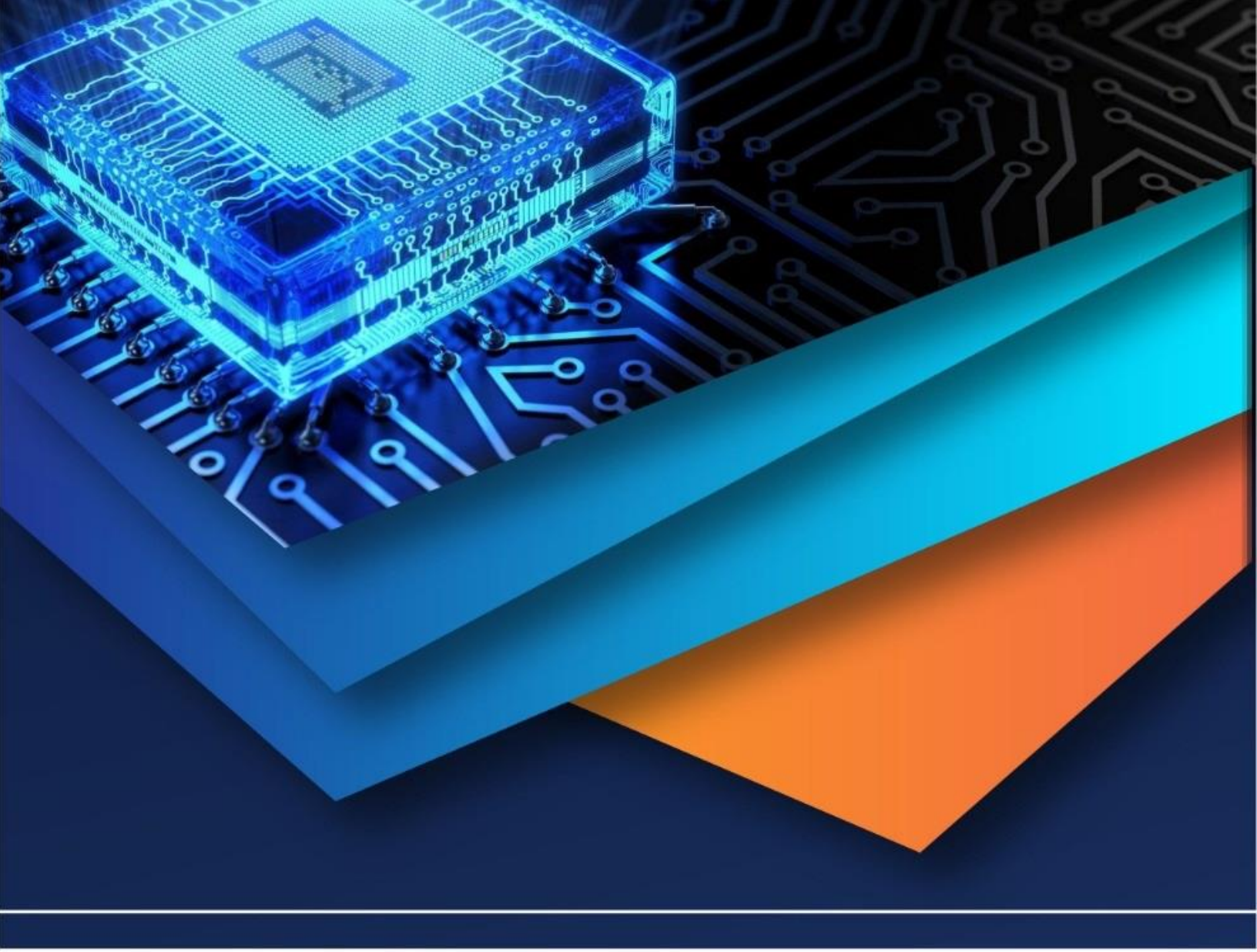
The software explicitly mitigates hardware limitations through, client-side rendering for search filtering, and safe unmounting procedures to protect data integrity. Furthermore, the integration of Tailscale effectively solves the complexities of remote access, providing secure global connectivity without the risks associated with traditional port forwarding.

This work demonstrates that the Raspberry Pi Zero 2W can form the basis of a fully functional, user-friendly NAS when paired with a carefully designed Flask web interface. The system provides secure login, intuitive file management, real-time upload feedback, client-side search, and safe storage management all while staying within the hardware's modest limits. By keeping the implementation lightweight and avoiding unnecessary dependencies, we created a solution that is both practical today and easy to extend. The combination of the web UI with Tailscale remote access offers a balanced, low-cost alternative to commercial NAS devices for home users, students, and small teams.

While hardware limitations restrict this architecture from scaling to enterprise-level deployments, the solution successfully fills a critical niche for cost-sensitive applications where commercial NAS systems would be heavily overprovisioned. The project contributes a practical, open-source blueprint for transforming single-board computers into secure, highly functional data storage devices.

REFERENCES

[1] S.~P. Khandale, “IoT based network attached storage,” International Journal for Research in Applied Science and Engineering Technology (IJRASET), vol.~12, no.~6, pp. 1234--1245, 2024. doi: 10.22214/ijraset.2024.65616.
[2] N.~Malse, A.~Lad, A.~Mandpe, and N.~Lanjewar, ``Developing a secure private cloud storage system," IJRASET Journal for Research in Applied Science and Engineering Technology, vol.~12, no.~5, pp. 5678--5687, 2024. doi: 10.22214/ijraset.2024.65348.
[3] R.~Ritzkal, K.~Kodarsyah, A.~R.~S. Nudin, I.~H. Setiadi, F.~Riana, and B.~Wulandari, “Enhancing data storage and access in CSN labs with Raspberry Pi 3B+ and open media vault NAS,” Information Dynamics and Applications, vol.~2, no.~2, pp.~63--76, 2023. doi: 10.56578/ida020202.
[4] R.~Messinetti, “Integration and development of a dynamic web server on embedded microcontrollers,” Master’s thesis, Politecnico di Torino, 2018.
[5] U.~Chatterjee, “Optimizing Samba file sharing performance on Raspberry Pi devices for efficient and lightweight network storage solutions,” 2024.
[6] A.~C. Jaya, “Single-board computer for affordable personal data storage server,” Jurnal Mantik, vol.~5, no.~3, 2021.
[7] E.~Gamess and G.~Hester, “Using a Raspberry Pi as a network-attached storage device: performance and limitations,” in Proceedings of the 2025 Conference on Embedded Systems, 2025.
[8] A.~Kuriakose, “Own Cloud Using Raspberry: comparing OwnCloud Storage Using Raspberry Pi and Commercial Clouds for Data Storage,” theseus.fi, 2025.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY