Develop a LAN-based Password Management System Dedicated to Android Devices and Fresh Tomato Routers

Hussein Abdulkhaleq Saleh
Directorate General of Education in Dhi Qar
Al-Haboubi Street, Nasiriyah, 64001, Dhi Qar, IRAQ.

ABSTRACT
This research introduces a LAN-based password management system tailored for Android devices and FreshTomato routers, offering a secure, localized alternative to cloud-based and device-centric solutions. The system integrates an Android client application with a router-based storage server, utilizing the router's constant availability within the local network to ensure offline accessibility and reduce dependency on external services or the device itself. Passwords are safeguarded through AES-256 encryption, SHA-512 hashing, and continuous authentication, with communication between client and server facilitated by HTTP and JSON protocols. Testing on emulators and a physical device demonstrated the system's functionality, rapid response times, and minimal resource usage, confirming its usability and performance. However, limitations include reliance on router availability and location constraints within the LAN. This work establishes a decentralized framework for password management, enhancing user autonomy and resilience against cloud-centric threats. Future enhancements could expand compatibility with other router firmwares and incorporate recovery options.

Keywords
Password Management System, FreshTomato Firmware, LAN Passwords Storage, Android Application.

1. INTRODUCTION
In today's interconnected digital landscape, accessing most online services—from social media platforms and e-commerce sites to online banking and healthcare portals—invariably requires users to create accounts. These accounts serve as gateways to access those services, enabling features, tailored recommendations, and secure communication. Consequently, the proliferation of digital platforms has made account creation an unavoidable step for individuals seeking to engage with the modern cyber world. As a result, the average individual now manages a multitude of accounts, many of which include sensitive data integral to personal or financial matters. From a security standpoint, sensitive data may be vulnerable to unauthorized access. For example, a compromised email account could lead to identity theft [1], while breaching banking credentials might result in significant financial losses [2]. From another perspective, the human tendency to reuse simple, memorable passwords across platforms exacerbates vulnerabilities, creating a fragile security ecosystem where a single breach can jeopardize multiple accounts.

We live in an era marked by escalating cyber threats, like ransomware, phishing campaigns, and data breaches. Protecting sensitive information has never been more critical, and security frameworks must evolve to safeguard the data. Within this framework, passwords remain a foundational layer of defense, yet their effectiveness hinges on secure creation, management, and storage practices. A single weak or reused password can undermine even the most advanced security protocols, emphasizing the need for better management solutions. Common pitfalls include predictable patterns (e.g., "123456"), reuse across platforms, and infrequent updates [3]. These practices render accounts susceptible to brute-force attacks and credential-stuffing exploits. Moreover, the burden of memorizing complex, unique passwords for dozens of accounts leads many users to prioritize convenience over security, further eroding protection [4].

Storing passwords securely is as critical as creating strong ones. Improper storage methods—such as recording credentials in unencrypted digital notes, spreadsheets, or physical documents—expose users to significant risks [5]. A lost notebook or a breached cloud file can transform trivial negligence into a catastrophic privacy violation. Even widely adopted solutions like browser-based password managers, while convenient, centralize risk by storing data on third-party servers, which may become targets for large-scale attacks [6]. Moreover, many users still rely on outdated methods to manage passwords, including handwritten lists, static digital files, or memorization. These approaches lack scalability and security, offering no protection against physical theft, unauthorized access, or digital breaches. When passwords are stored locally on a user's device instead, new risks emerge, such as vulnerabilities tied to physical access.

To tackle these challenges, contemporary password managers have emerged, which offer encrypted vaults, cross-device synchronization, and automated password generation. However, many solutions depend on cloud infrastructure, necessitating trust in external providers. For users concerned about privacy, relying on third parties raises anxiety, especially after notable breaches of centralized platforms [7]. Moreover, cloud-based systems may struggle in offline situations or localized network environments, which limits their overall applicability. Even when passwords are stored locally on a user's device, new risks arise. Device-centric storage introduces vulnerabilities tied to physical access: if the device is lost, stolen, or temporarily borrowed, unauthorized parties could exploit poorly protected files or cached credentials [8]. These shortcomings highlight the need for approaches that reduce dependency on both cloud infrastructure and user-owned devices while maintaining security and accessibility. Considering the drawbacks of cloud-based and device-centric approaches, localized alternatives like repurposing network hardware, such as routers for secure storage, remain underexplored despite their potential to overcome these challenges.

Routers are ideal due to their constant availability and strategic position as network gatekeepers, enabling secure, localized storage. Particularly, those running open-source firmware like FreshTomato, OpenWRT, and DDWRT offer a unique opportunity to host localized password repositories. Their inherent role as network gatekeepers, combined with customization capabilities, positions them as promising candidates for decentralized, LAN-based security solutions. However, such systems require dedicated client interfaces to enable seamless user interaction, particularly in mobile-centric environments where smartphones serve as primary access points. Android, as the world's most widely used mobile OS [9], provides an ideal platform for developing lightweight, secure client applications that bridge users to LAN-hosted services.

This paper leverages these synergies to propose developing an integrated system consisting of an Android client application for managing passwords, paired with a FreshTomato router configured as a passwords storage server, which enables secure password management, prioritizes offline accessibility, user autonomy, and resistance to cloud-centric threats.

1.1 Paper contributions

This paper introduces a novel approach within the field of password management systems by developing a LAN-based user password storage system dedicated to Android devices and FreshTomato routers. The development process will involve the following:

- Design and implement an Android application to function as a client interface for the user based on usability and security standards.
- Customize a FreshTomato router to serve as a dedicated password repository accessible only via a local connection, and implement the server-side logic to perform tasks required by the Clint app.

This architecture enhances security through physical and network isolation while empowering users with direct oversight of their data. By bridging the gap between convenience and robust protection, the proposed system offers a resilient alternative to traditional and cloud-based solutions for users who own those types of devices, aligning with the evolving demands of digital security. The proposed system eliminates reliance on external cloud services and prioritizes localized control and offline accessibility. Additionally, the proposed system reduces dependence on the device itself for password storage, mitigating the risks associated with theft or loss.

1.2 Structure of the Paper

This paper is organized into nine sections. Following this introduction, Section 2 reviews existing literature on password management systems and prior implementations of LAN-based storage solutions. Section 3 details the system requirements and the methodology of implementing this work.

Section 4 elaborates on the GUI design process of the Android client application, based on some recommended functional requirements. Section 5 details the implementation of the application's functions and features, such as password management operations (adding, deleting, editing, etc.), encryption and decryption processes, network communication, and other related tasks. Section 6 shifts focus to server-side development. It begins by exploring FreshTomato firmware, customizing it to host the password vault, and implementing the server-side processes. Section 7 evaluates the integrated system through functional and security testing, identifies limitations and disadvantages, and discusses potential vulnerabilities. Finally, Section 8 concludes the work by summarizing findings and proposing directions for future research. A comprehensive list of references is provided in Section 9 to acknowledge foundational studies and tools utilized in this work.

2. LITERATURE REVIEW

With the growing need for secure and efficient password management, research has focused on developing robust solutions. This section reviews some existing password management systems, highlighting their key contributions, limitations, and the gaps addressed by the proposed LAN-based storage system.

2.1 Password Management Systems

Password management systems aim to simplify managing, storing, and retrieving passwords while ensuring security. PassMan [10] introduces a novel approach where passwords are generated on-the-fly using a master password, a phrase, and a hint, without storing the passwords locally or in the cloud. This approach prevents password theft from storage but requires users to remember multiple parameters. Similarly, Versipass [11] uses graphical password cues to help users associate passwords with accounts, reducing the cognitive load of remembering multiple passwords. However, these systems rely heavily on user input, which can be prone to errors or forgotten hints.

2.2 Local Storage and Encryption Techniques

Local storage solutions, such as BANK OF PASSWORDS [12], store passwords on the user's device using robust encryption techniques like AES-256 and SHA-256. This approach encrypts passwords as ciphertexts, preventing access even if the device is compromised. However, local storage solutions are vulnerable to device loss or theft, which could expose encrypted passwords to brute-force attacks if the master password is weak.

2.3 Cloud-Based Solutions

Cloud-based password managers, such as the one proposed by Kanela [13], use cloud storage to synchronize passwords across multiple devices. These systems encrypt passwords using AES-256 and store them in the cloud, ensuring availability and convenience. However, cloud-based systems introduce risks such as data breaches and reliance on third-party servers.

2.4 Security and Usability Trade-offs

Password management systems often struggle to balance security and usability. PassMan addresses this by generating passwords on-the-fly, eliminating the need for storage, but it requires users to remember multiple parameters. On the other hand, the BANK OF PASSWORDS focuses on local storage with strong encryption, ensuring security but limiting accessibility if the device is lost. Versipass improves usability by using graphical cues to help users remember passwords, but it relies on user input, which can lead to errors.

2.5 Gaps in Existing Systems

Existing systems reveal several key gaps:

- Cloud Dependency: Systems like the one proposed by Kanela rely on cloud storage, which introduces risks such as data breaches and reliance on third-party servers.

- ❖ Device-Centric Risks: Local storage solutions, such as BANK OF PASSWORDS, are vulnerable to device loss or theft, which could expose encrypted passwords.
- ❖ Usability Challenges: Systems like PassMan and Versipass require users to remember multiple parameters or cues, which can be prone to errors or forgotten inputs.

The proposed LAN-based password storage system addresses these gaps by leveraging localized storage on a FreshTomato router, eliminating reliance on cloud services for users who own that type of router. By combining the security of local storage with the convenience of an Android application, the system offers a robust and decentralized solution for password management.

3. IMPLEMENTATION METHODOLOGY

This section outlines the technical framework, tools, and processes employed to develop the LAN-based password management system. The methodology is divided into subsections to clarify the client-side (Android application) and server-side (FreshTomato router) components, along with data handling and communication protocols. Figure 1 clearly shows the complete view of the system workflow.

3.1 System Architecture

The system adopts a client-server architecture with the following components:

1. Client: An Android application for password management operations.
2. Server: A FreshTomato router configured as a localized password storage server.
3. Communication: HTTP-based interactions between the client and server using the OkHttp library. Data will be transmitted and received in the form of JSON objects.

3.2 Client-Side Development

The Android application is designed to handle user interactions and securely communicate with the server. Key aspects include:

3.2.1 Functional Requirements

- ➢ Password Management:
  - Store: Add new passwords with three fields:
    - Account Name (e.g., Facebook, Gmail).
    - Username (optional).
    - Password Value (encrypted).
  - Edit/Delete: Modify or remove stored passwords.
  - Copy to Clipboard: copy decrypted passwords.
- ➢ User Authentication:
  - Register/Login with a username and password (hashed using SHA-512).
  - Logout functionality to terminate sessions.
- ➢ Server Configuration:
  - Set server IP/port for HTTP communication.
  - Check server availability before executing tasks (e.g., adding/editing passwords).

3.2.2 Development Tools

- IDE: Android Studio 2024.2.1 for UI design and logic implementation.
- Language: Java 11 for developing app codes and backend logic.
- Networking: OkHttp 4.12.0 library for HTTP requests.
- SDK:
  - Min SDK: 24 (Android 7.0 Nougat).
  - Target/Compile SDK: 34 (Android 14).

3.2.3 Security Protocols

- Encryption: AES-256 for encrypting/decrypting password data during storage/retrieval.
- Hashing: SHA-512 for securing user login credentials.

3.3 Server-Side Configuration

The FreshTomato router hosts the password vault and processes client requests.

3.3.1 Server Setup

- Hardware: Netgear R8000 router with FreshTomato firmware (2024.3).
- Web Server: Nginx 1.27.0 embedded with the FreshTomato firmware to handle HTTP requests.
- PHP Scripts: Coded in Visual Studio Code 1.96.4 for backend operations:
  - Dedicated PHP pages for specific tasks (e.g., `add_password.php`, `login.php`).

3.3.2 Database Design

- Database Engine: MySQL via AdminerEVO 4.4.
- Tables:
  - Users: Stores `username` (plaintext) and `hashed_password`.
  - Passwords: Stores encrypted `account_name`, `username`, and `password_value`.
- Storage: A USB stick connected to the router hosts the database and PHP scripts.

3.4 Testing Environment

- ➢ Client Testing:
  - Emulator: Official Android Emulator (API 34).
  - Physical Device: Xiaomi Mi Max 3 smartphone (Android 10).
- ➢ Server Testing:
  - Validate PHP script functionality and MySQL queries.
  - Monitor HTTP response times and error rates.

4. APPLICATION GUI DESIGN

The GUI of the Android client application is designed to provide an intuitive and secure way to manage passwords stored on a FreshTomato router within a local area network (LAN). It focuses on usability and security, allowing users to perform password management tasks while protecting sensitive data. This section outlines the GUI components for password management (store, edit/delete, copy), user authentication (register/login, logout), and server configuration (set server IP/port, check availability).

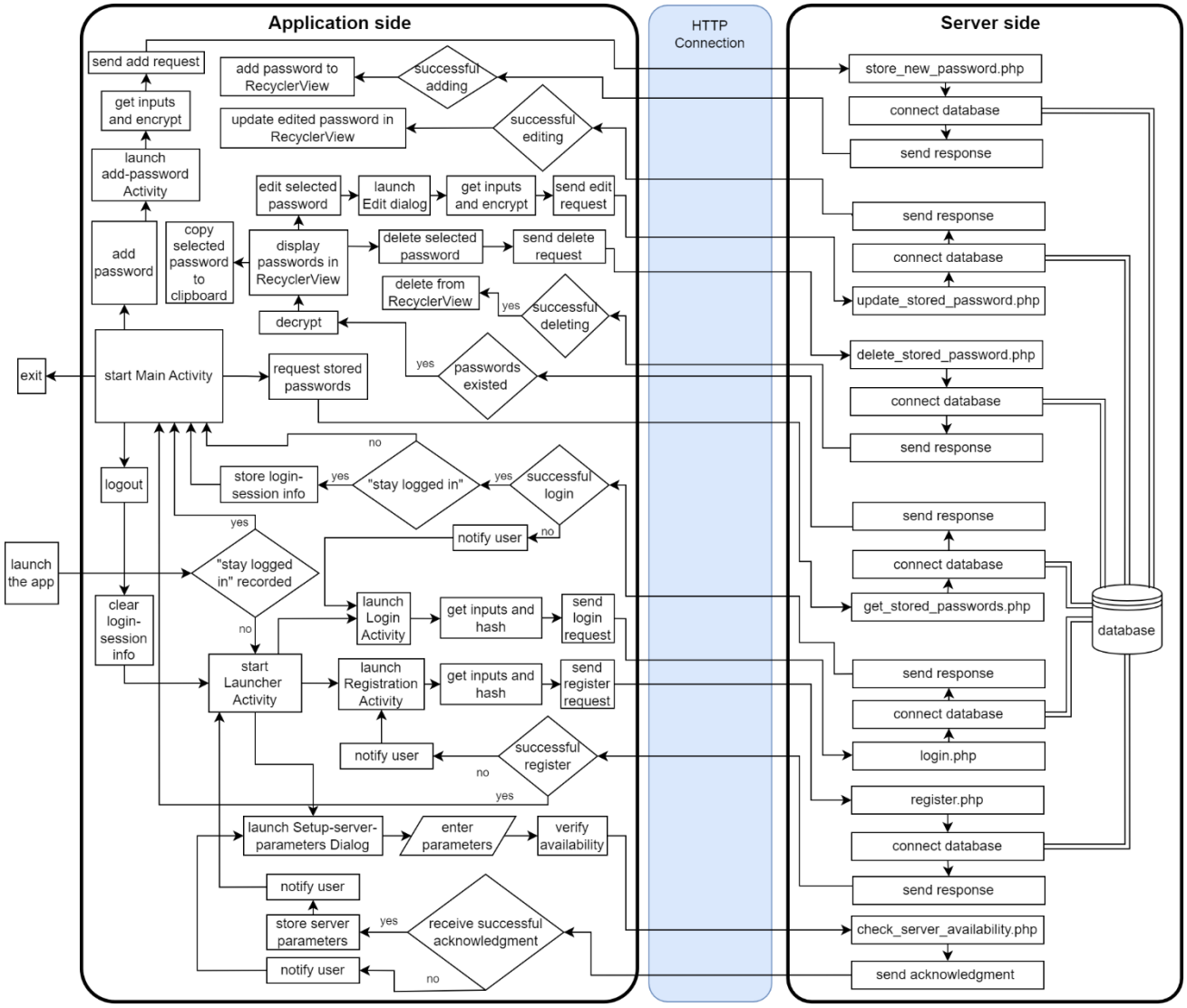

Fig 1: A diagram illustrating the architecture and workflow of the proposed system.

4.1 Launcher Activity

The launcher activity serves as the application's entry point, featuring instructional text and three main buttons: "LOGIN," "REGISTER," and "SERVER SETUP." Selecting "LOGIN" or "REGISTER" leads to the respective authentication screens, while "SERVER SETUP" opens a configuration dialog. The buttons use a consistent color scheme for better visual coherence. This layout separates core functions, making navigation easier. The instructional text helps first-time users understand each button's purpose, providing a streamlined starting point that enhances usability and directs users efficiently to key tasks like authentication.

4.2 Server Setup Dialog

Activated by the "SERVER SETUP" button in the launcher activity. This dialog enables users to configure the router connection. It features input fields for the server's IP address and port, pre-filled with defaults (e.g., "192.168.1.1" and "555"), an "OK" button, and a circular `ProgressBar` that activates during availability checks, as shown in Figure 2-D.By isolating server configuration in a dialog, the launcher activity remains focused on navigation, while the dialog provides a dedicated space for entering network details, as illustrated in Figure 2-D. This setup ensures reliable server communication by enabling the user to change the details and aligning with the server configuration requirements. The `ProgressBar` offers real-time feedback, confirming the app is processing the input and keeping the user informed during server communication, which improves user experience. Note that the ProgressBar component will be used when designing other activities and dialogs in this app for the same purposes mentioned earlier.

4.3 Registration Activity

The registration activity allows users to create accounts, featuring input fields for username, password, and password confirmation, alongside "Register" and "Cancel" buttons. A circular `ProgressBar` indicates processing during registration, and underlined input fields highlight active areas, guiding user interaction, as shown in Figure 2-B.The password confirmation field minimizes errors, enhancing security by ensuring accurate credential entry. This design supports the secure registration requirement, offering a clear and feedback-rich account creation process.

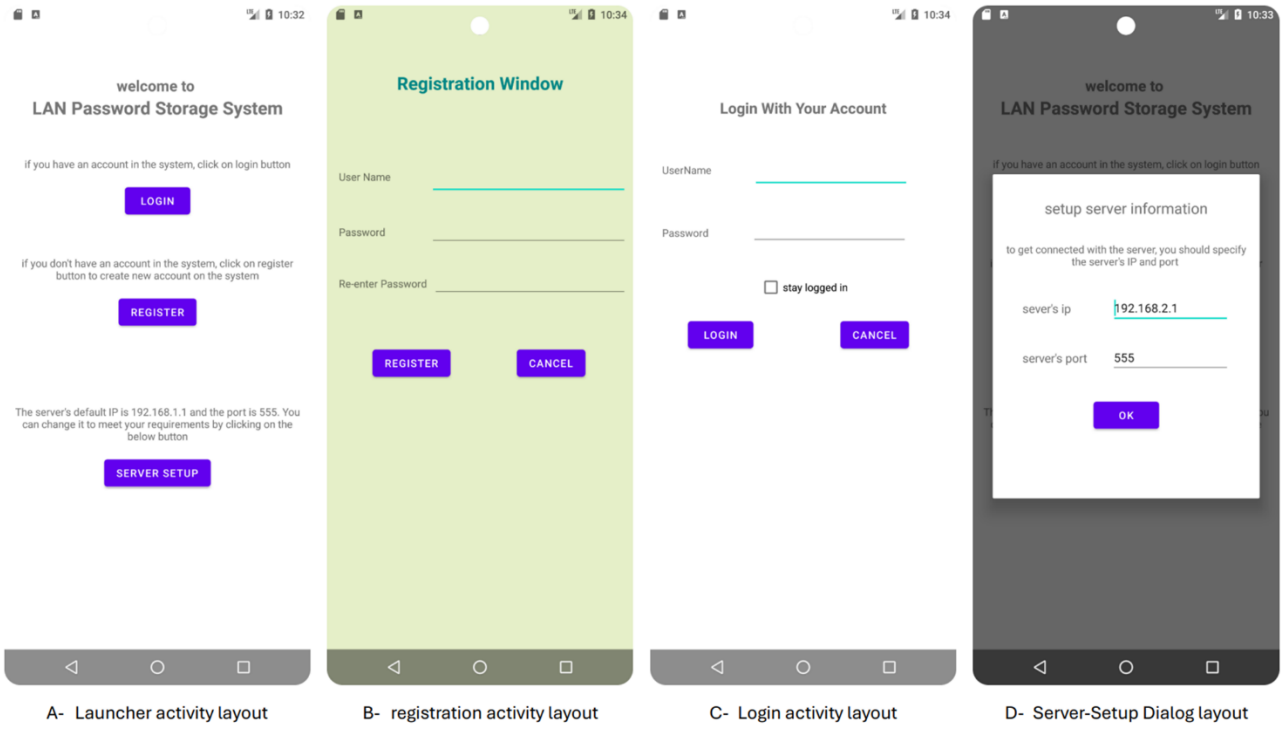

Fig 2: GUI Design – part 1.

4.4 Login Activity

Tailored for authenticating existing users, the login activity includes input fields for username and password, a "stay logged in" checkbox, "Login" and "Cancel" buttons, and a circular ProgressBar, as illustrated in Figure 2-C. Underlined input fields provide visual cues, indicating focus and simplifying data entry. A successful login directs users to the main activity.The "stay logged in" option improves convenience on trusted devices, balancing usability with security. This layout fulfills the login requirement, ensuring secure and user-friendly access to password management features.

4.5 Main Activity

The main activity serves as the password management hub, featuring a Toolbar with the title "LAN-Based Password Storage System" and a subtitle showing the logged-in username (e.g., "login as: [username]"). A Toolbar menu that includes "Logout" and "Exit" options is created on the right side, as shown in Figure 3-B. A RecyclerView in the center displays stored passwords in card format, and a Floating Action Button (FAB) with a "+" icon initiates new password entries. The complete layout of this activity is illustrated in Figure 3-A. Both the Toolbar and FAB use a consistent color scheme for visual unity.

The Toolbar provides session context and quick access to logout/exit, fulfilling the logout requirement. The RecyclerView leverages Android's efficient list management, ensuring scalability, while the FAB offers an intuitive, thumb-friendly way to store new passwords. This design centralizes password management, enhancing usability and accessibility.

4.6 RecyclerView Item Layout

Each RecyclerView item represents a password entry structured using a GridLayout with a light blue background, organizing the account name, username, and password in a grid format. Also, it includes three icons that enable copying, editing, and deleting the entry, as shown in Figure 3-A. The copy icon transfers the password to the clipboard, the delete icon prompts a confirmation AlertDialog, and the edit icon opens an update dialog.

The light blue background of each GridLayout provides visual separation between entries. The grid format organizes details clearly, and the icons enable quick, recognizable actions, supporting the copy-to-clipboard, edit, and delete requirements. The confirmation dialog prevents accidental deletions, reinforcing reliability. This layout facilitates efficient interaction with password entries, meeting key functional needs.

4.7 Update Stored Password Dialog

Accessed via the edit icon in each RecyclerView Item. This dialog allows users to modify password entries. It includes pre-filled input fields for account name, username, and password, an "Update" button, and a ProgressBar for feedback, as illustrated in Figure 3-C. Underlined fields highlight editable areas, aiding user focus. Pre-filled fields and a dedicated dialog streamline editing, minimizing disruption.

4.8 Add New Password Activity

Triggered by the FAB. This activity enables users to add new passwords, featuring input fields for account name, username (optional), password, a "Store Password" button, and a circular ProgressBar, as illustrated in Figure 3-D. Clear labels and optional fields offer flexibility. This design fulfills the store's requirement with an intuitive interface for password creation. Underlined fields provide visual feedback, guiding data entry.

4.9 Overall Design Considerations

The GUI employs standard Android components (e.g., RecyclerView, FAB, and Toolbar) for familiarity and ease of use. Feedback mechanisms like ProgressBar elements

mitigate LAN-based latencies, improving responsiveness. A consistent color scheme across buttons and key elements enhances visual coherence. Optimized for mobile devices, the layouts account for screen size and touch interactions, ensuring accessibility. By aligning with the functional requirements and security protocols from Section 3, the GUI balances usability and protection, achieving the objectives of the proposed system.

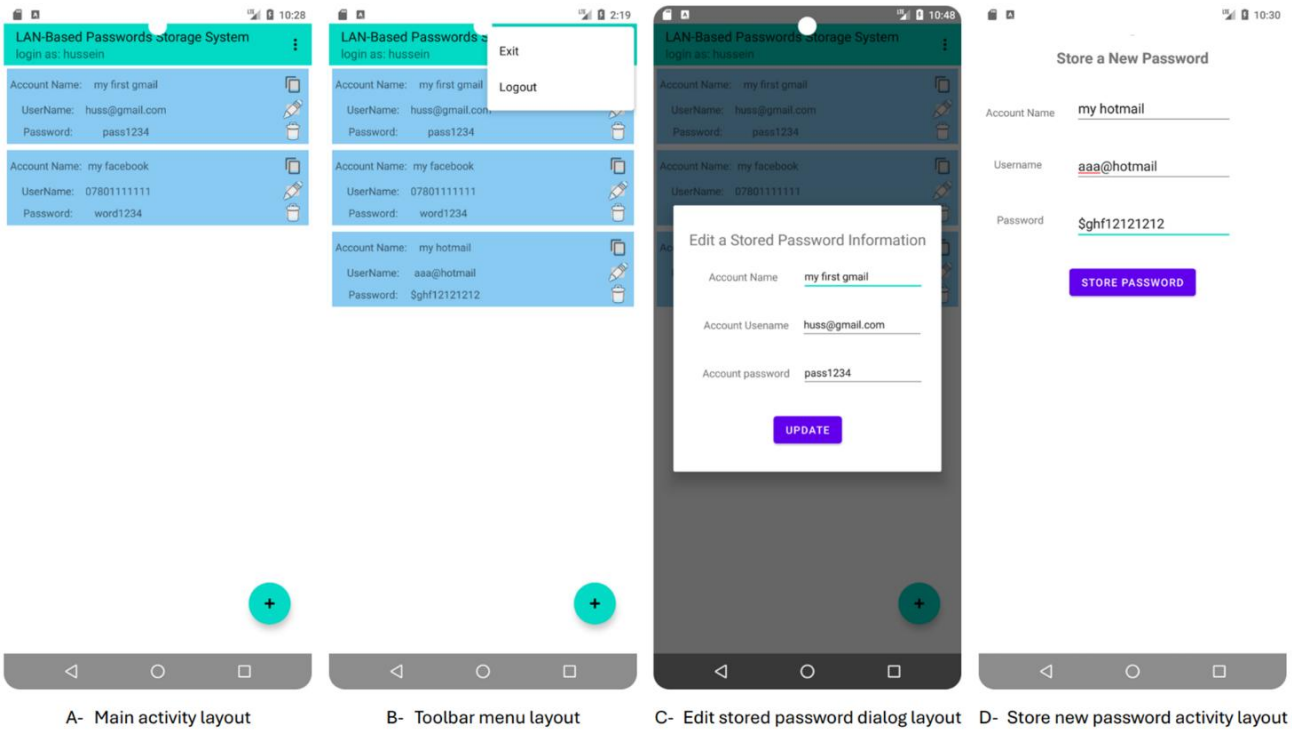

Fig 3:GUI Design – part

5. APPLICATION LOGIC IMPLEMENTATION

This section details the implementation of the Android client application's functionalities and its background processes. As stated in section 3.2.2, this app will be developed using Android Studio (version 2024.2.1), the official IDE recommended by Google for Android app development. Java will be used to create the code due to its popularity and ease of use [14]. The application will target devices running Android 7.0 (SDK 24) and above, up to Android 14 (SDK 34), due to its widespread use, according to Android Studio statistics.

The application uses the OkHttp library (version 4.12.0) for efficient network communication with the FreshTomato router. All network processes (e.g., login, register) will use CompletableFuture for asynchronous execution, ensuring responsiveness with a ProgressBar. Data will be transmitted as JSON objects. SharedPreferences will store lightweight data like server parameters and login credentials, as recommended for small persistent storage in Android. For security, SHA-512 will hash user login credentials, while AES-256 will encrypt/decrypt stored passwords.

The Android Manifest.xml file of the proposed application will declare permissions and configurations critical for network functionality in the LAN environment:

- Permissions:
  - <uses-permission android:name="android.permission.INTERNET" />: Enables HTTP requests to the FreshTomato router-hosted server.
  - <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" /> and <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />: Monitor connectivity status to ensure reliable server interactions.
- Cleartext Traffic: The attribute android:usesCleartextTraffic="true" permits HTTP communication, required for compatibility with the LAN server's lack of HTTPS support. Note that to mitigate security risks, sensitive transmitted data is encrypted at the application before transmission.

The application's logic is structured into modular components, each addressing specific functional requirements. The following subsections detail the core workflows, including activity lifecycle management, user authentication, password operations, and network reliability mechanisms. These components collectively enable secure and efficient interaction with the LAN-hosted server while adhering to Android platform best practices.

5.1 Launcher Activity

The LauncherActivity serves as the entry point, managing navigation and server setup. It checks the login state in SharedPreferences: if "login", it skips authentication and navigates to MainActivity for seamless access on trusted devices; otherwise, it offers "LOGIN", "REGISTER", and "SERVER SETUP" options, as illustrated in Figure 4.

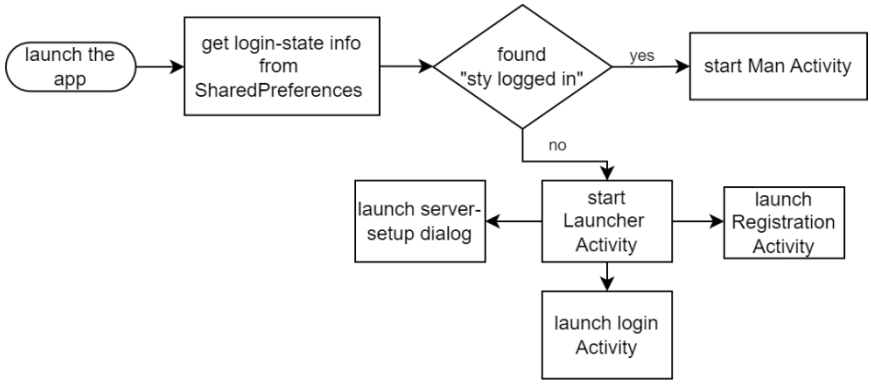

Fig 4: Launcher-activity workflow diagram

Server Configuration: The "SERVER SETUP" button opens a dialog for inputting the server's IP (default: "192.168.1.1") and port (default: "555"). These defaults are initialized in SharedPreferences upon first launch if unset, ensuring immediate usability while allowing customization for flexibility. The JSON object will include one small message: "hello server". An asynchronous GET request to 'http://[ip]:[port]/check_server_availability.php' validates connectivity, with a ProgressBar providing feedback. Successful validation saves settings in SharedPreferences for flexible use, as illustrated in Figure 5.

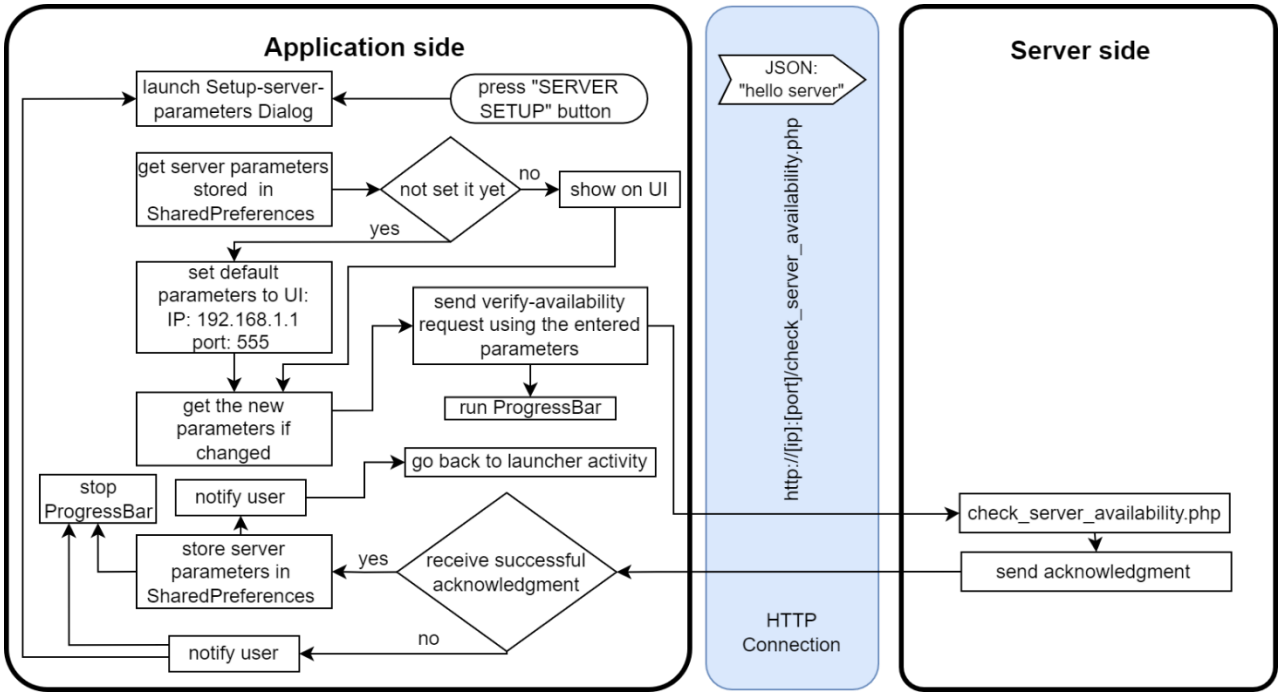

Fig 5: server-parameters setup process diagram

5.2 User Authentication

Authentication, handled by RegistrationActivity and LoginActivity, secures credentials with SHA-512 hashing before transmission. Cryptographic operations are offloaded to a dedicated Encryption class, enhancing modularity and reusability.

- Registration: Users enter a username and password (with confirmation), which is hashed using SHA-512 and sent to 'http://[ip]:[port]/register.php'. Client-side validation enforces a minimum password length of 8 characters to ensure strength. The success registration stores credentials in SharedPreferences with a "logout" state, redirecting to MainActivity, as detailed in Figure 6. Setting login-state to "logout" means it is the first time the user logs into the system, and they should enter the login credentials next time to access their account.
- Login: LoginActivity hashes the password and sends it to 'http://[ip]:[port]/login.php'. A "stay logged in" option sets the login state in the SharedPreferences to "login" for future automatic access, as Figure 7 shows.

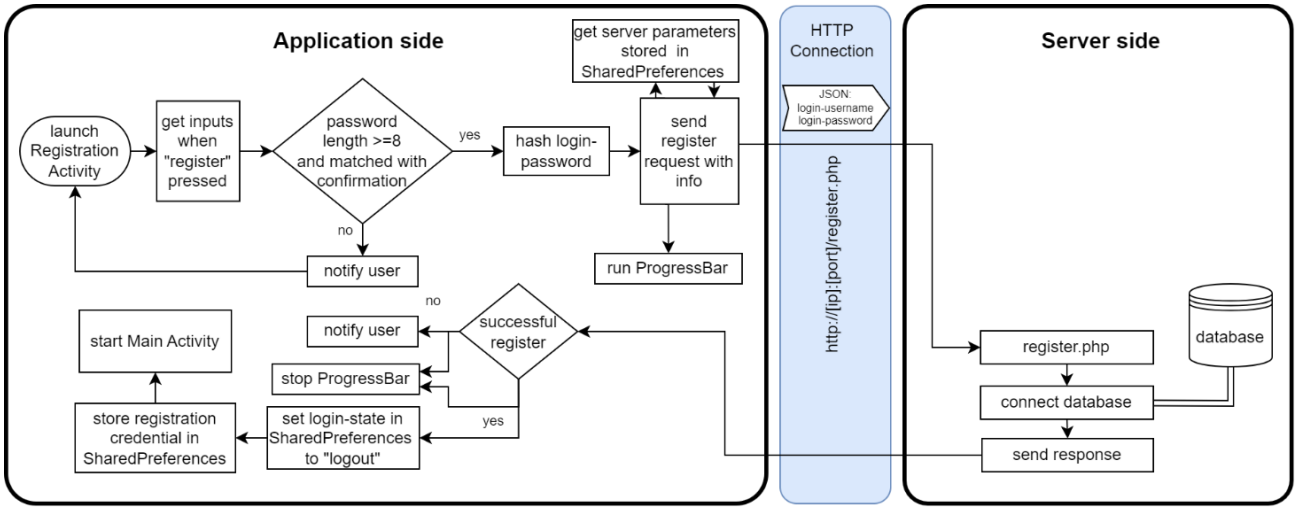

Fig 6: Registration process diagram

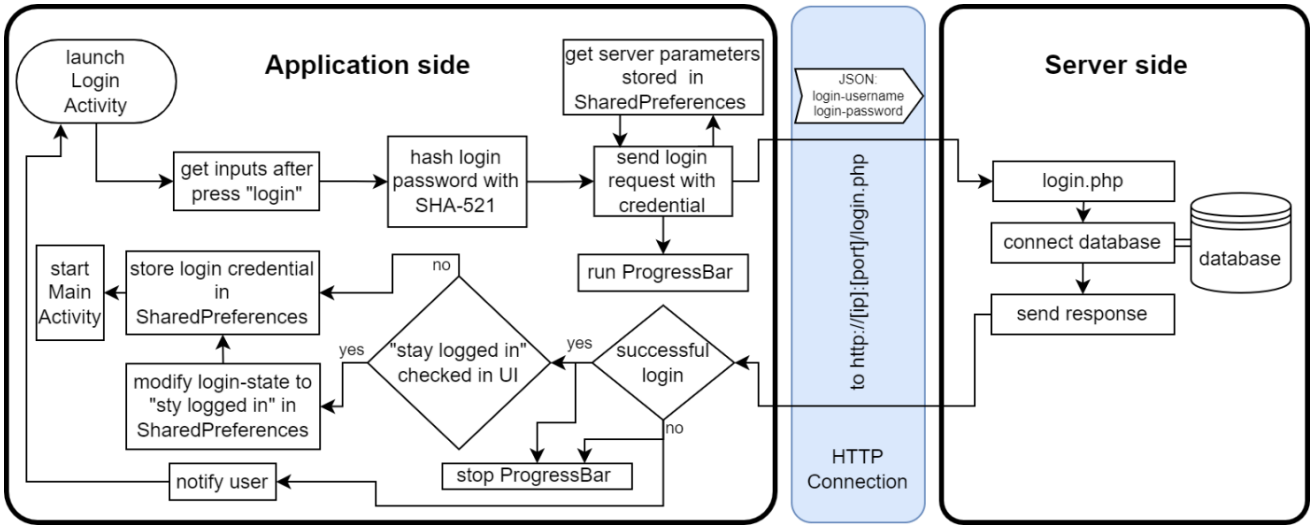

Fig 7: Login processes diagram

5.3 Password Management Operations

Password management is centralized in MainActivity, using a RecyclerView for display and a Floating Action Button (FAB) for adding entries. Optimistic UI updates refresh the RecyclerView immediately after local operations, ensuring responsiveness despite network latency. This activity includes a Toolbar menu (as depicted in Figure 3-B) offering 'Logout' and 'Exit' options for session control. The logout process clears the login state and returns to LauncherActivity, as illustrated in Figure 8.

Except for copy operations, password management operations employ AES-256 encryption to secure password data (account name, username, and password, as shown in Figure 3-A). The key used for this algorithm is derived dynamically from user credentials (username and password) using PBKDF2 with 10,000 iterations. This ties decryption to the user's master credentials, ensuring that even if the server's database is compromised, passwords remain protected.

The 10,000-iteration count aligns with NIST recommendations (SP 800-63B) for PBKDF2 [15], balancing security against brute-force attacks and computational efficiency on mobile devices.Each operation request to the server includes the user's login credentials for authentication and as a unique database identifier. This ensures that actions like retrieving or modifying passwords are limited to the user's data, maintaining account isolation, and preventing data leakage. A progress bar also appears with each request, providing feedback during data fetching.

- Viewing Passwords: MainActivity fetches encrypted passwords in the background after launch directly via 'http://[ip]:[port]/get_stored_passwords.php', decrypts them using AES for immediate display, and populates the RecyclerView with the decrypted entries for efficient user access, as detailed in Figure 8. The 'get-stored-passwords' request will include login credentials, allowing the server to search for stored passwords relevant to the user account.
- Storing Passwords: The FAB launches Add_New_Password_Activity, where inputs are encrypted and sent to http://[ip]:[port]/store_new_password.php after clicking on 'store password'. Success updates the local list and RecyclerView, then triggers a Toast notification, as depicted in Figure 9. Similar to the get-passwords request, the store-new-password request will include login credentials in addition to the new password's info to be stored.
- Editing and Deleting Passwords: The same encryption and network patterns in the previous process have been

applied here. The request for the edit-password process includes the new password information and the old one, as shown in Figure 10. The password-delete process triggers a confirmation AlertDialog that prevents accidental deletions, as detailed in Figure 11. The request to delete the selected password includes the login credentials to delete the relevant password on the server side.

- Copying Passwords: A copy icon button in `RecyclerView_Adapter` uses `ClipboardManager` to transfer decrypted passwords to the clipboard, notifying users via `Toast`. This function does not require any network communication to execute it.

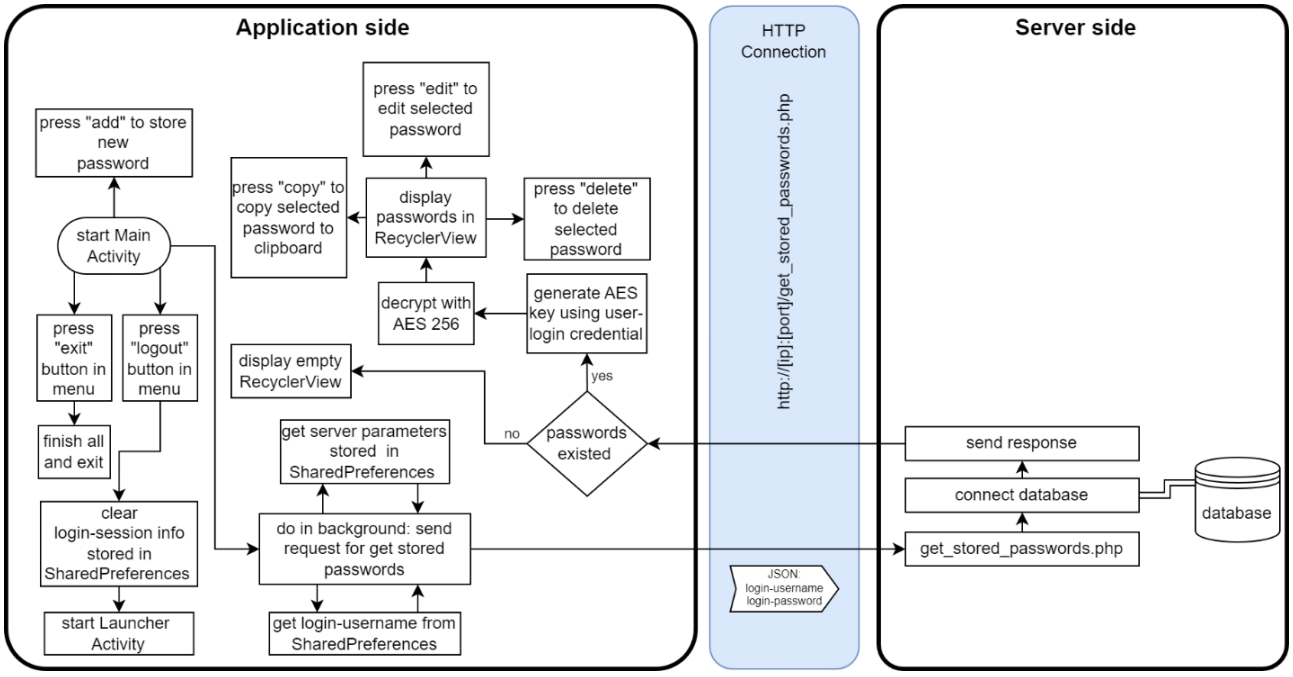

Fig 8: A diagram details the MainActivity process workflow (exit, logout, display stored passwords)

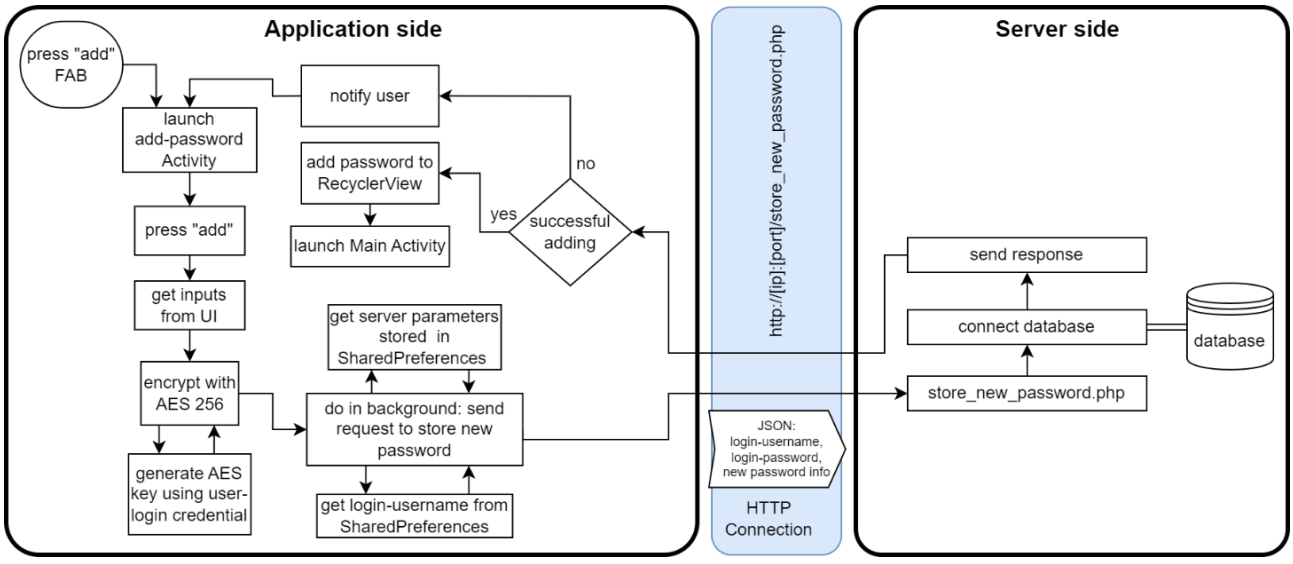

Fig 9: A diagram illustrating the workflow of storing a new password

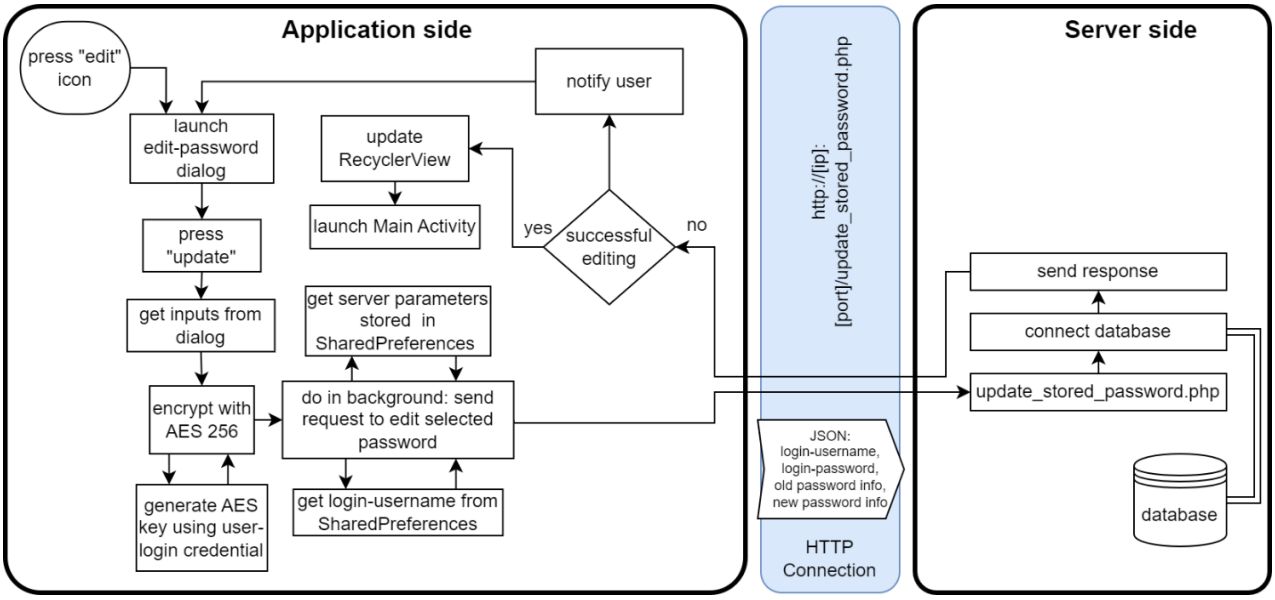

Fig 10: Edit password process diagram

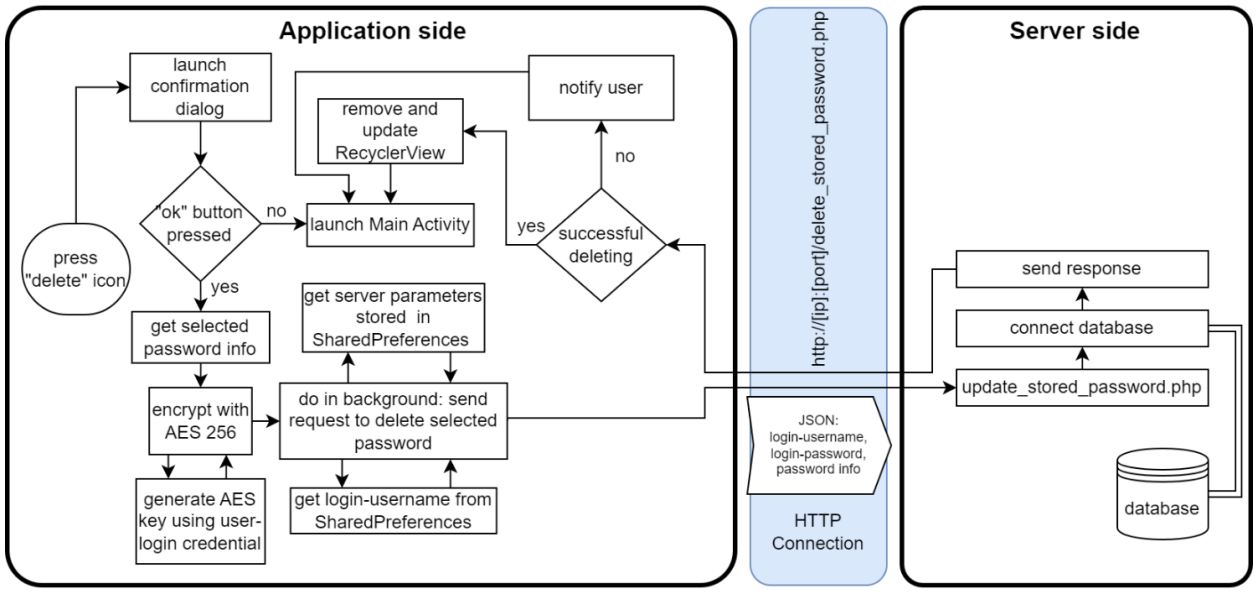

Fig 11: Delete password process diagram

5.4 Network Communication and Error Handling

The Android app uses OkHttp for HTTP communication with the FreshTomato router, emphasizing responsiveness through asynchronous execution with CompletableFuture. Network operations (authentication, password management, server availability checks) run in the background to avoid UI freezes and keep users engaged with ProgressBar feedback. To combat unstable LAN conditions, OkHttpClient enforces default timeouts (10 seconds for connection and read), ensuring users get alerts like “server not found” if the router is unresponsive.

Data transmission uses JSON formatting for interoperability, with encrypted payloads serialized into structured objects. Error handling is granular: timeouts trigger warnings, JSON parsing failures prompt server reconfigurations, and invalid credentials reset the UI for session integrity. User feedback is prioritized through Toast notifications, while server responses are validated with a "state": "success" flag. For instance, password deletion requests proceed only after the server confirms database removal.

The network layer integrates into the broader workflow (Figure 1). For example, when storing a password, the client creates a JSON object, sends it to ‘/store_new_password.php’, and updates the RecyclerView on the main thread via runOnUiThread(). Although the system purposely avoids automatic retries for failed requests to maintain simplicity, it ensures reliability through strong timeout safeguards and user-friendly error messaging.

5.5 Security Considerations

The system utilizes a multi-layered security strategy to protect user data. During registration, client-side validation ensures a minimum password length of 8 characters and prohibits empty fields. User passwords are hashed using SHA-512 before transmission, preventing plaintext exposure on the server. For every password management request (e.g., viewing, storing, or editing passwords), login credentials (username and hashed password) are embedded to verify user authenticity

continuously. This approach reduces risks related to session hijacking and reinforces overall security.

As mentioned previously, stored passwords use PBKDF2 (10,000 iterations) to dynamically derive encryption keys from the user's master password, ensuring decryption is tied to the user. Data is Base64-encoded for JSON compatibility and stored in SharedPreferences with MODE_PRIVATE to limit access on rooted devices. Activities are secured by setting non-entry components to android:exported="false" in the manifest and overriding the back button in MainActivity to end sessions with finishAffinity(), which prevents accidental navigation to sensitive screens. These measures create a strong defense-in-depth architecture that balances usability and security against digital and physical threats.

6. SERVER-SIDE CONFIGURATION IMPLEMENTATION

In the preceding sections, we elaborated on the design and implementation of the Android client application, which provides an intuitive interface for users to manage their passwords securely. However, the functionality of this client application depends on a robust server-side infrastructure to store and process password data within a local area network (LAN). This section shifts focus to the server-side development, utilizing a FreshTomato router as the backbone of our LAN-based password management system. We will explore the FreshTomato firmware and its embedded software, configure the Netgear R8000 router to serve as a dedicated password storage server, design and build the database using AdminerEvo, and implement server operations through PHP scripts. By the end of this section, the server will be fully equipped to handle requests from the Android client, ensuring secure, localized password management.

6.1 Exploring FreshTomato Firmware and Embedded Software

FreshTomato is an open-source firmware project based on Linux, designed for routers with Broadcom chipsets. As stated on freshtomato.org, it is dedicated to routers with a Broadcom chipset and distributed under the GPL license. This firmware enhances the capabilities of consumer-grade routers by providing advanced features such as the ability to host web servers and databases, which are functionalities critical to this research. Its user-friendly web interface simplifies configuration, making it accessible to both novice and experienced users, while its robust feature set meets the technical demands of our password management system.

The advantages of FreshTomato include:

- Broad Device Support: Compatible with numerous routers, such as the Netgear R7000, Asus RT-AC68U, and Linksys E4200, in addition to the Netgear R8000 used in this work.
- Server Hosting Capabilities: Supports hosting web servers (nginx) and databases (MySQL), critical for storing and managing password data.
- USB Functionality: Enables external storage via USB, facilitating portable hosting of PHP pages and databases.
- Community Support: Benefits from regular updates and an active user community, ensuring reliability and ongoing development.

In comparison to other open-source firmwares like DD-WRT, which offers extensive features but a steeper learning curve, and OpenWRT, which provides high customization but requires significant expertise, FreshTomato strikes a balance between ease of use and advanced functionality. Its streamlined interface and simplified configuration workflows reduce resource consumption, making it an ideal choice for this project, where hosting a web server and database on a router is essential without necessitating overly intricate configurations.

The embedded software employed in this system includes:

- Nginx: A lightweight, high-performance web server that processes HTTP requests from the Android client application.
- mysqld: The MySQL database server, responsible for storing user accounts and password entries.
- AdminerEvo: A web-based database management tool that facilitates the design and administration of the MySQL database.

These tools are pre-integrated into FreshTomato, streamlining the setup process for our server-side implementation. The base device for this work is the Netgear R8000 router, equipped with a dual-core 1.8 GHz ARM processor and 256 MB RAM [16], running FreshTomato firmware version 2024.3. Selected for its robust hardware (including multiple USB ports and sufficient processing power), the R8000 is well-suited to handle the server tasks required by our system. Notably, the proposed solution is designed to be compatible with any ARM- or MIPS-based router running the same FreshTomato 2024.3 firmware, enhancing its applicability across similar hardware platforms, though performance may vary depending on router specifications.

For storage, a USB 3.0 flash drive formatted as NTFS is connected to the router's USB 3.0 port. This stick hosts both the PHP scripts and the MySQL database, providing a portable and manageable solution for data storage within the LAN environment.

6.2 Configuring the Router for Server Functionality

Configuring the Netgear R8000 as a server involves setting up USB storage, the NGINX web server, and the MySQL server. These steps ensure the router can host the password management system's backend, as depicted in the system architecture (Figure 1).

Accessing the Web Interface:

- Open a web browser and enter the router's default IP address: `192.168.1.1`, then log in using the default credentials: username `root`, password `admin`.

Setting Up USB Storage:

- Navigate to 'USB and NAS' > 'USB Support' in the firmware's web interface, and enable Core USB, USB 3.0/2.0, USB Storage, and NTFS support.
- Check 'Automatically mount all partitions to sub-directories in '/mnt' under the 'Automount' section to mount the USB partitions automatically.
- Click 'Save' and reboot the router to ensure the settings take effect.
- Post-reboot, verify the mounted partition in the 'Attached Devices' section, where the USB stick appears as `/mnt/sda5` (see Figure 13).

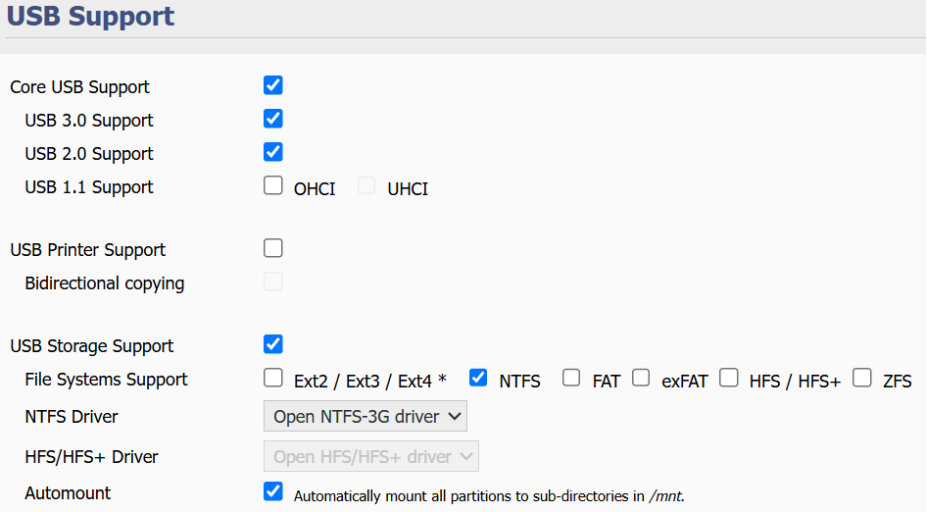

Fig 12: Setting Up USB Storage in FreshTomato firmware

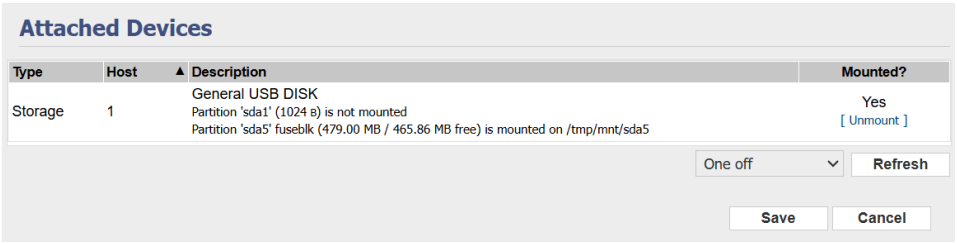

Fig 13: The mounted partition in the 'Attached Devices' section

Configuring the Web Server (nginx version 1.27.0):

- Navigate to 'Web Server' > 'Nginx & PHP', and enable:
  - 'Enable on Start': Ensures the web server starts automatically on router boot.
  - 'Enable PHP Support': Allows execution of PHP scripts.
- Set 'Server Port' to `555`, aligning with the Android client's configuration.
- Set 'Document Root Path' to `/mnt/sda5/lan_password_managment_system` based on our case, where `/mnt/sda5` is derived from the mounted USB partition, and `lan_password_managment_system` is the folder for PHP files.
- Click 'Save'. The nginx configuration is illustrated in Figure 14.

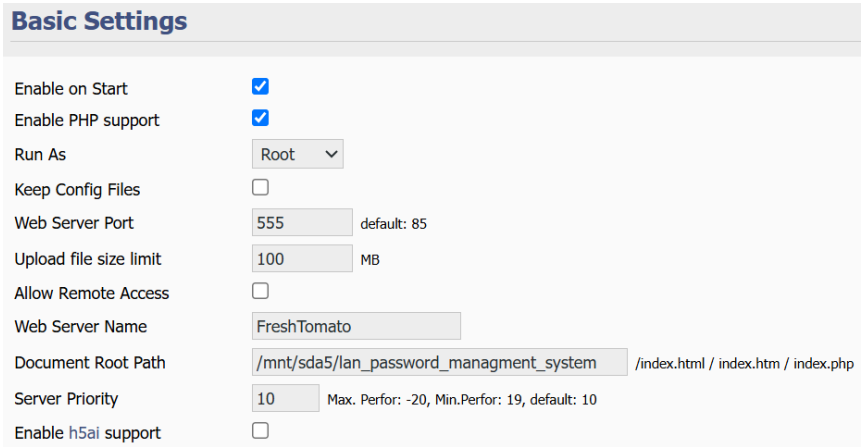

Fig 14: NGINX configuration in FreshTomato firmware

Configuring the MySQL Server:

- Navigate to 'Web Server' > 'MySQL Server', and check 'Enable on Start' to launch the database server on router startup, then click 'Save'. The MySQL server setup is shown in Figure 15.

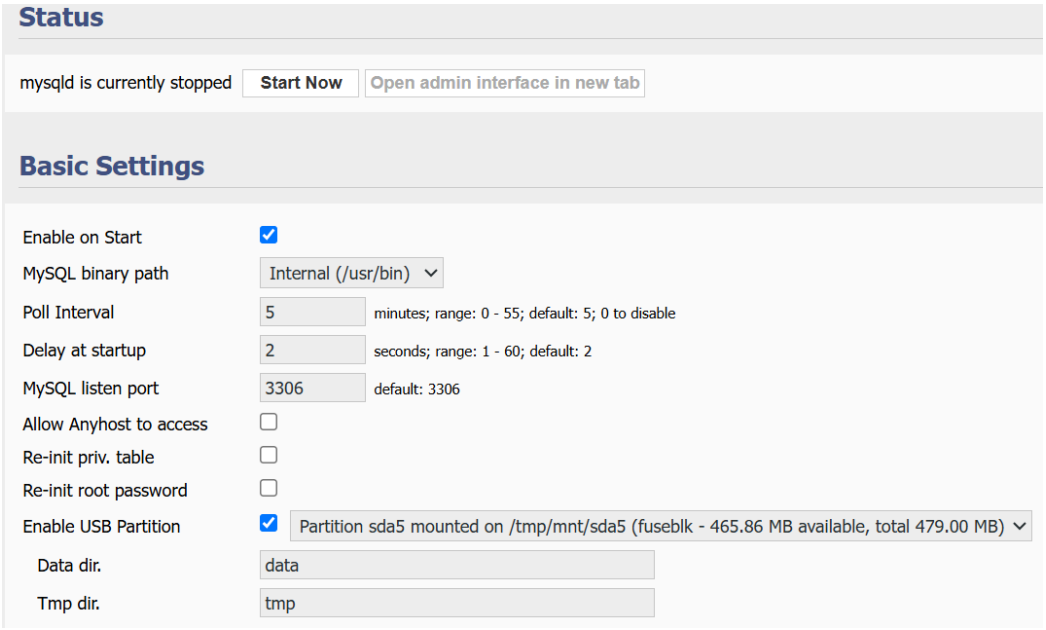

Fig 15: The MySQL server setup in FreshTomato firmware

These configurations establish the router as a fully functional server, capable of hosting the web and database services required for the password management system.

6.3 Designing the Database with AdminerEvo

The database is designed and built using AdminerEvo (version 4.8.4), leveraging the MySQL server running on the router. The steps are as follows:

- In the 'MySQL Server' settings, click 'Start Now' to activate mysqld.
- Click 'Open admin interface in new web' to launch AdminerEvo in a new browser tab.
- Log in with the default credentials: username `root`, password `admin`.
- Select 'Create Database' and name it `Lan_Password_Storage_System`.
- Create two tables within this database:
  - users_table:
    - `login_username` (VARCHAR(50), PRIMARY KEY): Unique identifier for each user.
    - `login_password` (VARCHAR(128)): Stores the hashed user password.
  - passwords_table:
    - `user_login_username` (VARCHAR(50), FOREIGN KEY to `users_table.login_username`): Links passwords to a specific user.
    - `account_name` (VARCHAR(100)): Name of the account (e.g., "Facebook").
    - `account_username` (VARCHAR(100)): Account username (optional).
    - `account_password` (VARCHAR(256)): Encrypted password value.

This schema ensures data isolation by associating each password entry with a unique user via the foreign key constraint. The database is stored on the USB stick, within the MySQL data directory configured to utilize the mounted `/mnt/sda5` partition.

6.4 Implementing Server-Side Operations with PHP

Server-side operations are implemented through seven PHP scripts, as illustrated in Figure 1. Each one handles specific requests from the Android client. These scripts were developed using Visual Studio Code and deployed to the USB stick in the directory `/lan_password_managment_system`.

The PHP pages and their functionalities are:

- `check_server_availability.php`: Returns a JSON response (`{"state":"success","message":"server available"}`) to confirm server connectivity (see Figure 16).
- `register.php`: Validates if a username exists in `users_table`. If not, insert the new username and hashed password, returning a success or error JSON response.
- `login.php`: Authenticates users by verifying the username and hashed password against `users_table`, returning a success or failure JSON response.
- `get_stored_passwords.php`: Retrieves all password entries for an authenticated user from `passwords_table`, returning them as a JSON array.
- `store_new_password.php`: a new encrypted password entry into `passwords_table` for the authenticated user, confirming success via JSON.
- `update_stored_password.php`: Updates an existing password entry in `passwords_table` based on old and new values, returning a success or error JSON response.
- `delete_stored_password.php`: Deletes a specified password entry from `passwords_table` for the authenticated user, confirming success via JSON (see Figure 17).

Each script uses PDO (PHP Data Objects) to securely interact with the MySQL database, preventing SQL injection by employing prepared statements. Inputs are received as JSON objects via `php://input`, and responses are returned in JSON format, aligning with the client's communication protocol (Section 5.4).

The deployment Process includes:

- Write the PHP scripts in Visual Studio Code.
- Copy the files to the USB stick in the `/lan_password_managment_system` directory.
- Connect the USB stick to the router's USB 3.0 port and reboot the router to recognize the files.

Figures 16 and 17 illustrate the workflows of these server-side operations, detailing how client requests are processed and how responses are generated. With these configurations and implementations, the FreshTomato router is fully operational as a server, supporting the LAN-based password management system by securely storing and managing user passwords within the local network.

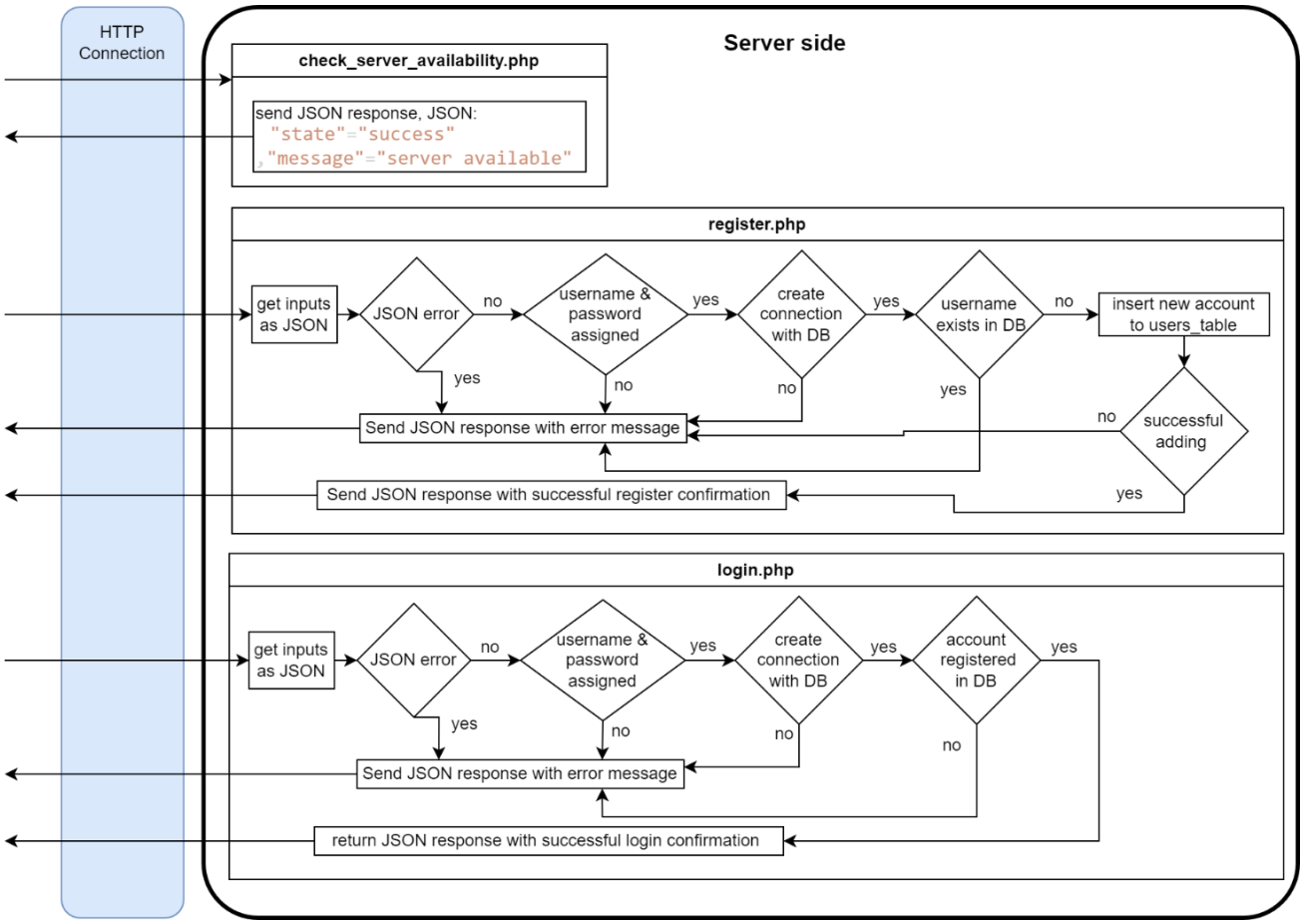

Fig 16: Server-side operations logic - part 1

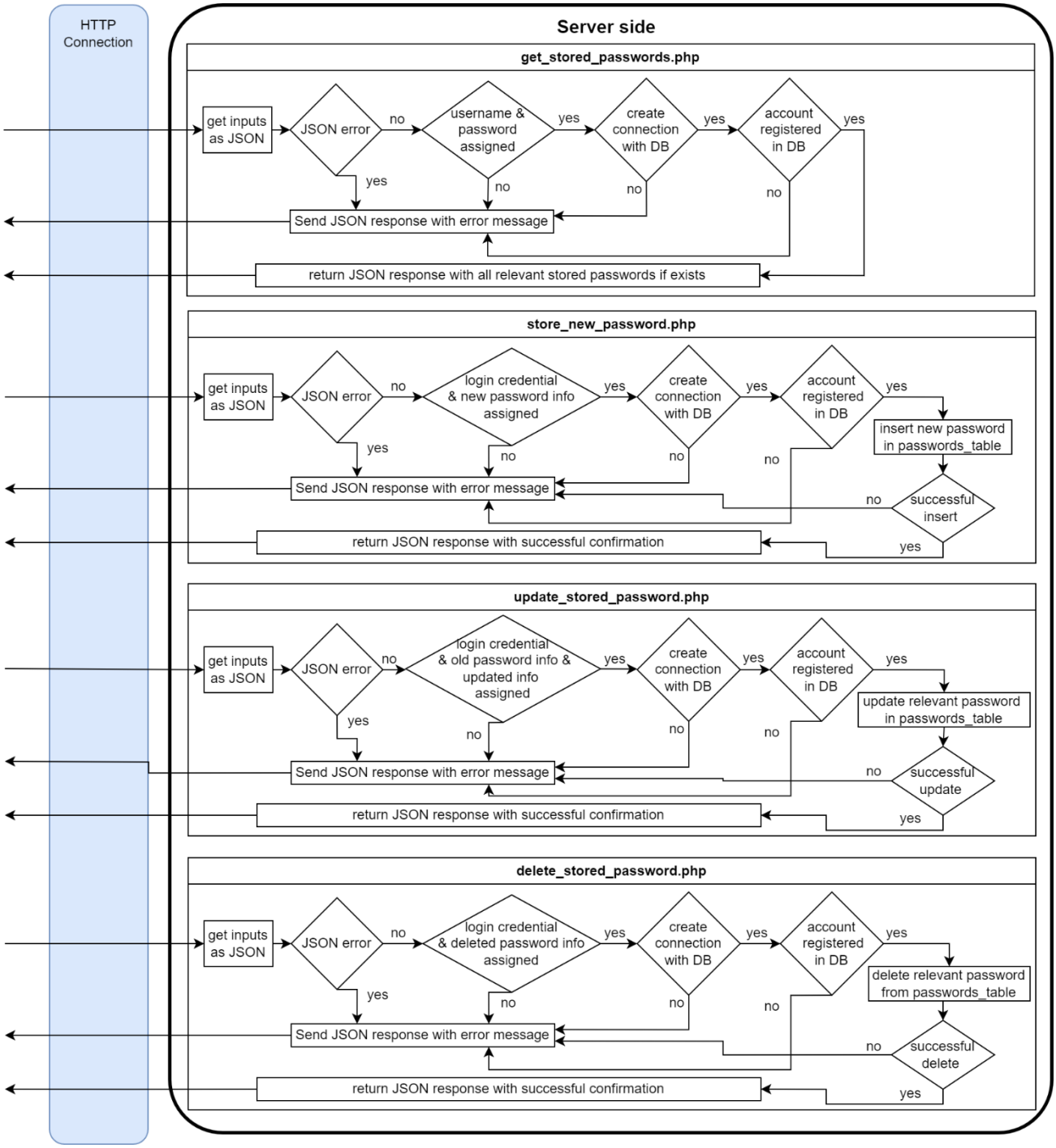

Fig 17: Server-side operations logic - part 2

7. Testing, Evaluation, and Limitations

This section evaluates the proposed system through a structured testing process, assesses its performance, and identifies inherent limitations. The system, comprising an Android client application and a FreshTomato router configured as a password storage server, was tested for functionality, reliability, and usability within a local area network (LAN) environment. The following subsections outline the testing methodology, functional testing outcomes, performance metrics, and the system's limitations, providing a comprehensive analysis of its practical implementation.

7.1 Testing Methodology

To ensure the system's compatibility and functionality across diverse Android environments, two distinct testing approaches were employed:

- Emulator Testing: Two virtual devices were created using the Android Studio Emulator, both simulating a Pixel 7 smartphone. The first device was configured with a minimum SDK of 24 (Android 7.0 Nougat), while the second targeted SDK 34 (Android 14). This range was selected to verify the application's performance across a broad spectrum of Android versions, encompassing the system's supported SDK boundaries as specified in Section 5. Testing on these emulators ensured

compatibility with older and newer Android iterations without requiring exhaustive testing on every intermediate version.

- ❖ Real-Device Testing: A Xiaomi Mi Max 3 smartphone running Android 10 was used as the physical test device. An Android Package (APK) file was generated from Android Studio and transferred to the device via USB for installation. This approach validated the system's behavior on actual hardware, accounting for real-world factors such as device-specific performance and network conditions that emulators might not fully replicate.

Both testing methods were conducted within a controlled LAN environment, with the Netgear R8000 router (running FreshTomato firmware version 2024.3) serving as the server. The router was connected to a USB 3.0 flash drive hosting the MySQL database and PHP scripts, as detailed in Section 6. This setup ensured that all network interactions occurred locally, aligning with the system's design objectives of offline accessibility and localized control.

7.2 Functional Testing

Functional testing was performed to confirm that each component of the system operated as intended. The tests conducted on both the emulator and real device covered the user interface, server setup, authentication, password management operations, and network reliability. The results are summarized below:

- User Interface (UI) Layouts: All activities, dialogs, and visual components—such as buttons, input fields, and the RecyclerView—are displayed correctly across both testing environments. The layouts, designed in Section 4, adhered to Android standards, ensuring usability and visual consistency (e.g., Figures 2 and 3).
- Server Setup: The server configuration functionality was tested by inputting the router's IP address (default: "192.168.1.1") and port (default: "555") via the Server Setup Dialog (Figure 2-D). The application successfully stored these parameters in SharedPreferences and established a connection to the server, validated by a GET request to `check_server_availability.php`. The ProgressBar provided real-time feedback during this process, confirming operational success.
- Registration: This process was tested by entering a username and password in the Registration Activity (Figure 2-B). The client enforced a minimum password length of 8 characters, hashed the password using SHA-512, and stored the credentials in SharedPreferences with a "logout" state. A POST request to `register.php` successfully added a new row to the `users_table`, and the application transitioned to the Main Activity, fulfilling the design requirements from Section 5.
- Login: Login functionality was verified using existing credentials in the Login Activity (Figure 2-C). With correct inputs, the app authenticated the user via `login.php` and navigated to the MainActivity. The "stay logged in" checkbox, when selected, updated the login state in SharedPreferences to "login," enabling direct access to the Main Activity on subsequent launches, as intended (Figure 7).
- Password Management Operations:
  - Storing Passwords: The Add New Password Activity (Figure 3-D) was used to input account name, username (optional), and password. The data was encrypted with AES-256, sent to `store_new_password.php`, and a new row was added to the `passwords_table`. The RecyclerView was updated to reflect the new entry, with a Toast notification confirming success (Figure 9).
  - Editing Passwords: Editing an existing password via the Update Stored Password Dialog (Figure 3-C) updated the corresponding entry in the `passwords_table` through `update_stored_password.php`. The modified data remained encrypted, and the RecyclerView reflected the changes accurately (Figure 10).
  - Deleting Passwords: Deleting a password via the RecyclerView item's deleted icon triggered a confirmation AlertDialog. Upon confirmation, a request to `delete_stored_password.php` removed the entry from the `passwords_table`, and the RecyclerView updated accordingly (Figure 11).
  - Copying Passwords: The copy icon in the RecyclerView transferred the decrypted password to the clipboard using ClipboardManager, verified by pasting it into a text field, with a Toast notification confirming the action.
  - All operations were tested multiple times, achieving a 100% success rate with no errors in both the emulator and physical device environments, confirming the robustness of the client-server interaction, as shown in Table 1.
- Network Operations: Network reliability was assessed during all server interactions (e.g., login, password storage). Responses were fast and consistent, with no application freezing observed. Success and failure messages were displayed appropriately via Toast notifications, and ProgressBars provided feedback during asynchronous operations, aligning with the OkHttp implementation in Section 5.4.
- Background Behavior: After closing the application using the "Exit" option in the Main Activity's Toolbar menu, no background processes persisted, as confirmed by device monitoring tools. This behavior matched the design goal of minimizing resource usage (Section 5).
- LauncherActivity Logic: Upon relaunch, the LauncherActivity (Figure 2-A) correctly displayed based on the SharedPreferences login state: "logout" prompted authentication, while "login" bypassed it, navigating directly to the Main Activity (Figure 4).

All tests were conducted with the FreshTomato router operational, confirming the functionality of server-side PHP scripts and MySQL interactions as outlined in Section 6. Table 1 summarizes the system's Functional Test Results.

Table 1: Summary of Functional Test Results

Function	Test Case	Expected Outcome	Actual
Server Setup	Enter the correct IP and port	Connection successful, parameters saved	As expected
	Enter an incorrect IP	Connection fails, error message	As expected
Registration	Register with a new username	User added, transition to Main Activity	As expected
	Register with an existing username.	Error message displayed	As expected
Login	Login with the correct credentials	Authentication successful	As expected

	Login with an incorrect password	Authentication failed, error message	As expected
Store Password	Add new password	Password stored, list updated	As expected
Edit Password	Modify an existing password	Password updated, list reflects changes	As expected
Delete Password	Remove password	Password deleted, list updated	As expected
Copy Password	Copy password to clipboard	Password copied, notification displayed	As expected

7.3 Performance Evaluation

The system's performance was assessed based on key metrics derived from the testing process:

- Response Times: Network operations, facilitated by the OkHttp library with a 10-second timeout, exhibited rapid response times (typically under 1 second) within the LAN environment. No delays or freezes were observed during authentication, password management, or server availability checks, indicating efficient client-server communication.
- APK Size: The generated APK file measured 6.19 MB, a reasonable size for an Android application with the described functionality. This optimized size ensures that the application can be quickly installed and run on devices with limited storage, which is particularly important for older Android versions supported by the system (SDK 24 and above). Also, this aligns with the lightweight design goals outlined in Section 5, ensuring compatibility with devices of varying storage capacities.
- Resource Usage: The application ceased all background activity upon exit, consuming no additional CPU or memory resources, as verified on Xiaomi Mi Max 3. Monitoring with Android Profiler confirmed that no memory leaks or lingering processes persisted after the application was closed, aligning with the design goal of minimal resource impact.

Overall, the system performed reliably under normal LAN conditions, with the FreshTomato router effectively handling HTTP requests and database operations via its nginx web server and MySQL server, as configured in Section 6.

7.4 Limitations and Disadvantages

While the system achieved its functional and security objectives, several limitations were identified during evaluation, reflecting its LAN-based architecture and hardware dependencies:

- Router Dependency: The system's operation hinges on the FreshTomato router's availability. If the router is powered off, malfunctions, or becomes inaccessible, all password management services cease, rendering the application unusable until the router is restored.
- Location Constraints: Access to password management features requires the client device to be connected to the router's Wi-Fi network. Users outside the LAN (e.g., at a different location) cannot interact with the system, limiting its utility for mobile users requiring remote access.
- Wi-Fi Coverage Impact: Poor Wi-Fi signal strength or limited router coverage can degrade system performance, leading to slower response times or connection failures, particularly in larger or obstructed environments.
- Physical Security Risks: If the router is stolen or physically accessed, the USB flash drive containing the encrypted database and PHP scripts could be compromised. While data remains encrypted, a determined attacker with sufficient expertise might attempt to extract it, posing a security risk.
- USB Flash Drive Reliability: The system relies on the USB flash drive for storing the MySQL database and PHP scripts. Disconnection, corruption, or hardware failure of the drive will disable the server, halting all functionality and potentially causing data loss if not backed up.
- Lack of Credential Recovery: No recovery mechanism exists for forgotten login credentials. If a user forgets their username or password, they cannot access the system or retrieve stored passwords, resulting in permanent inaccessibility of their data. This absence is intentional to prioritize security over convenience.
- Scalability Constraints: The system is optimized for individual or small-scale use within a single LAN. Supporting multiple users or larger networks may require significant modifications to the database schema and server capacity, which were not tested in this implementation.
- Absence of Backup Mechanisms: No automated backup process is implemented for the database stored on the USB flash drive. Hardware failure or data corruption could lead to an irreversible loss of stored passwords, a risk not addressed in the current design. To address this vulnerability, adopting a manual backup approach is highly advisable.

8. CONCLUSION AND FUTURE WORK

This research presents a novel LAN-based password management system designed for Android devices and FreshTomato routers, delivering a secure, localized alternative to conventional cloud-based and device-centric solutions. By integrating an intuitive Android client with a router-based storage server, the system ensures offline accessibility, leverages the router's constant availability, and mitigates risks tied to device theft or loss. It employs advanced encryption and continuous authentication to safeguard data, while the lightweight, user-friendly Android interface enhances accessibility across a broad range of devices. Rigorous testing on emulators and physical hardware confirms its practical effectiveness.

Despite its strengths, the system's reliance on router availability and Wi-Fi connectivity limits its use beyond the local network, and the absence of data recovery options heightens the risk of credential loss. Scalability issues also restrict their applicability to small-scale deployments. These trade-offs between security and convenience highlight the challenge of aligning robust protection with seamless usability.

Ultimately, this work establishes a compelling framework for decentralized password management, empowering users with greater control over their credentials. Future enhancements could extend compatibility to additional router firmwares, introduce recovery mechanisms, and improve scalability, further advancing secure, user-centric password storage.

Building on this research, several directions are proposed to broaden its impact:

- Expansion to DD-WRT and OpenWRT Firmware:The current implementation leverages FreshTomato firmware, but adapting the system for other open-source

router platforms, such as DD-WRT and OpenWRT, would increase its reach. DD-WRT offers extensive customization and a broad device compatibility list, while OpenWRT provides a highly flexible Linux-based environment. Porting the server-side logic (e.g., nginx, MySQL, PHP scripts) to these firmwares would require adjusting configurations to their unique interfaces and resource constraints, enabling users with diverse router hardware to adopt this localized password management approach.

- Cross-Platform Expansion: Extending the client application to other platforms, such as iOS or a web-based interface for desktop browsers, would enhance versatility. This would require adapting the app's logic to new ecosystems while maintaining compatibility with the FreshTomato server, broadening the system's user base.

9. REFERENCES

[1] R. G. Brody, K. Mulig, and J. R. Kimball, "Phishing, pharming, and identity theft," J. Corp. Account. Finance, vol. 18, no. 2, pp. 43–49, Jan. 2007, doi: 10.1002/jcaf.20278.

[2] N. N. Cele and S. Kwenda, "Do cybersecurity threats and risks have an impact on the adoption of digital banking? A systematic literature review," J. Financial Crime, vol. 32, no. 1, pp. 31–48, Apr. 2024, doi: 10.1108/JFC-10-2023-0263.

[3] A. Adams and M. A. Sasse, "Users Are Not the Enemy: Why Users Compromise Computer Security Mechanisms and How to Take Remedial Measures," Commun. ACM, vol. 42, no. 12, pp. 40–46, Dec. 1999, doi: 10.1145/322796.322806.

[4] E. Stobert and R. Biddle, "The Password Life Cycle: User Behaviour in Managing Passwords," in Proc. Symp. Usable Privacy Security (SOUPS), Menlo Park, CA, USA, Jul. 2014, pp. 243–255.

[5] A. Karole, N. Saxena, and N. Christin, "Why Johnny Can't Store Passwords Securely: A Usability Evaluation of Password Managers," in Proc. Symp. Usable Privacy Security (SOUPS), Pittsburgh, PA, USA, Jul. 2011, pp. 1–16.

[6] R. Biddle, E. Stobert, and S. Chiasson, "A Security Analysis of Browser-based Password Managers," in Proc. Netw. Distrib. Syst. Security Symp. (NDSS), San Diego, CA, USA, Feb. 2016, pp. 1–14.

[7] S. K. Sharma and M. Warkentin, "Privacy Concerns and Trust in the Context of Cloud-Based Services: A Study of User Reactions to Data Breaches," J. Inf. Privacy Security, vol. 15, no. 3, pp. 123–139, Jul. 2019, doi: 10.1080/15536548.2019.1640974.

[8] D. McCarney, D. Barrera, J. Clark, and P. C. van Oorschot, "Security and Usability Challenges of Moving to Local Password Storage," in Proc. Annu. Comput. Security Appl. Conf. (ACSAC), New Orleans, LA, USA, Dec. 2014, pp. 256–265, doi: 10.1145/2664243.2664261.

[9] Statcounter, "Mobile Operating System Market Share Worldwide," Sep. 2024. [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/worldwide. [Accessed: Oct. 15, 2024].

[10] J. B. Billa et al., "PassMan: A New Approach of Password Generation and Management without Storing," in 2019 7th International Conference on Smart Computing & Communications (ICSCC), 2019, pp. 1–6. doi: 10.1109/SCSC.2019.8840591.

[11] E. Stobert and R. Biddle, "A Password Manager that Doesn't Remember Passwords," in Proceedings of the 2014 New Security Paradigms Workshop (NSPW '14), Victoria, BC, Canada, 2014, pp. 1–12. doi: 10.1145/2683467.2683471.

[12] H. A. Saleh, "BANK OF PASSWORDS: A Secure Android Password Manager Implemented Based on Specific Requirements," Al-Kitab Journal for Pure Sciences, vol. 8, no. 1, pp. 40–62, Mar. 2024. doi: 10.32441/kjps.08.01.p5.

[13] M. Kanela et al., "Secure and Manage Passwords with Encryption and Cloud Storage," in 2021 4th International Conference on Innovative Computing and Communication (ICICC), 2021, pp. 1–4. doi: 10.2139/ssrn.3833469.

[14] A. Petersen, J. Ko, and J. Pane, "Factors related to the difficulty of learning to program in Java—an empirical study of non-novice programmers," Inf. Softw. Technol., vol. 46, no. 2, pp. 99–107, Feb. 2004, doi: 10.1016/S0950-5849(03)00112-5.

[15] National Institute of Standards and Technology, "NIST Special Publication 800-63B: Digital Identity Guidelines," Jun. 2017. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63b.pdf. [Accessed: Oct. 15, 2024].

[16] Netgear, "Nighthawk X6 AC3200 Tri-Band WiFi Router (R8000) Data Sheet," 2021. [Online]. Available: https://www.downloads.netgear.com/files/GDC/datasheet/en/R8000.pdf. [Accessed: Oct. 15, 2024].