AirX, a cross-platform text and file sharing system

Shijunyi Liu shijunyil@mun.ca
Chang Guan cguan@mun.ca
Jiabao Guo jiabaog@mun.ca
Zhen Guan zguan@mun.ca

Memorial University of Newfoundland

Abstract

This paper introduces AirX, an innovative open-source data sharing system that allows for seamless synchronization of the clipboard and enables file sharing across various devices over the Local Area Network or the Internet. Beginning with an exhaustive literature review, this paper pairs each of AirX's unique features with corresponding research papers, providing a comprehensive context. Detailed insight into the implementation of each module showcases the system's technical efficiency, while rigorous testing underscores its robustness and reliability, emphasizing its potential for wide application. This paper concludes by outlining future plans, indicating a promising path for the system's evolution.

Contents

List of Acronyms

AES Advanced Encryption Standard

API Application Programming Interface

CORS Cross-Origin Resource Sharing

FFI Foreign Function Interface

HTTP Hypertext Transfer Protocol

JNI Java Native Interface

JSON JavaScript Object Notation

JWT JSON Web Token

LAN Local Area Network

MSIX Microsoft Installer eXtension

MVVM Model-View-ViewModel

REST Representational State Transfer

RSA The Rivest-Shamir-Adleman algorithm

SHA Secure Hash Algorithm

TCP Transmission Control Protocol

UDP User Datagram Protocol

UID Unique Identifier

URL Uniform Resource Locator

WPF Windows Presentation Foundation

XAML Extensible Application Markup Language

1 Introduction

In the era of advanced network technology, we have successfully constructed an expansive virtual world that seamlessly connects every individual across the globe. This digital revolution has made it possible to achieve communication at the millisecond level, effectively eradicating the barriers posed by physical distance.

Despite these technological advancements, there exists a persistent challenge in our daily life and work - **the efficient sharing of simple text and files across different devices.** This issue may seem trivial in the grand scheme of the digital revolution, but its implications are far-reaching and impact the efficiency of our day-to-day digital interactions.

Consider, for instance, a LAN. This network technology is designed to connect computers within a limited area such as a home, school, or office building. A LAN is a high-speed network that facilitates data transfer at an impressive rate, making it an ideal solution for sharing files and text within the same physical location.

However, despite the availability and efficiency of LANs, **most people do not fully utilize this technology for data transfer**. Instead, they resort to other methods that are often less efficient and more time-consuming. Emails and instant messaging applications, for example, are commonly used for sharing files and text. While these methods are effective, they are not designed for high-speed data transfer and can be slow, especially for large files. Furthermore, some people even resort to using USB flash drives for data transfer. This method, while simple, is highly inefficient. It involves physically moving the USB flash drive from one device to another, which can be time-consuming and inconvenient.

In order to address the text and file sharing problem, many solutions have been proposed:

- **Apple Airdrop** is a proprietary solution developed by Apple Inc. It is only available on Apple devices.
- **LocalSend** and **LANDrop** are open-source projects that enable text and file sharing between devices in the same LAN. However, they are only available over LAN and are unable to work over the Internet.
- **Instant Messaging Apps** such as Telegram, WhatsApp, Messenger and WeChat are widely used for text and file sharing. However, they are not designed for this purpose and are not suitable for sharing data over LAN.
- **Network Disk Services** such as Google Drive, Dropbox, OneDrive and iCloud Drive are designed for file sharing only.

In this paper, a complated cross-platform data sharing system, AirX, is proposed to make it easy for individuals to share text and files across different devices that running on different platforms. By leveraging the high-speed data transfer capabilities of LANs and the Internet, AirX ensures efficient and rapid sharing of data of any kind.

Table 1 compares AirX with existing solutions.

1.1 Cross-platform Support

AirX is designed with native front-end user interfaces (UIs) for Windows, macOS, Android, and Linux platforms. The support for iOS is contingent on the clipboard policy of Apple Inc.

Table 1: Compare with Existing Solutions. IM represents for Instant Messaging.

Subject	AirX	AirDrop	LocalSend	LANDrop	IM	Netdisks
Cross-platform	✓		✓	✓	✓	✓
Clipboard Sync over LAN	✓	✓				
Clipboard Sync over Internet	✓	✓				
Clipboard Sync over Bluetooth		✓				
Text Sharing over LAN	✓	✓				
Text Sharing over Internet	✓	✓			✓	
File Sharing over LAN	✓	✓	✓	✓		
File Sharing over Internet	✓	✓			✓	✓
Privacy Guarantee	✓	?	✓	✓		
Censorship Resistance	✓	?	✓	✓		
End-to-end Encryption	✓	✓	✓	✓	✓	✓
Private Server	✓					
Open Source	✓		✓	✓		

1.2 Native Development

In order to create a tool that is not only functional but also efficient and user-friendly for each platform, the "Cool Launch Time" (a metric indicating the duration it takes for an application's first launch since the device was powered on) is of paramount importance. Native development offers the fastest possible cold launch time, a detail that will be further elaborated in the architecture section.

1.3 Clipboard Synchronization

Clipboard synchronization is the core feature of AirX. It essentially extends the "Copy and Paste" function across different devices. By enabling AirX on multiple devices, users can copy text or files on one device and paste them on another, provided the devices are connected to the Internet or are in the same LAN. Users can also share their clipboard with others if they have mutually added each other as friends.

1.4 LAN-based File Synchronization

Copied files are automatically sent to all devices that meet the following conditions:

- Logged into the same account,
- Located within the same LAN,
- The total file size is less than 10 MB, and
- The receiver does not already have the file.

In such cases, the receiving devices automatically download the file, save it to the default download directory, and generate a notification. In other scenarios, files must be manually sent through AirX's UI, and receivers are prompted to accept or decline the file download.

1.5 Internet-based File Synchronization

Internet-based file synchronization is facilitated by a netdisk service where users can upload files and share the link with others. The link either directly accesses the file or redirects to a user-friendly

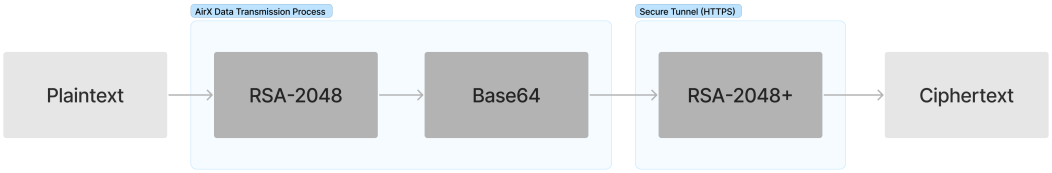


Figure 1: AirX Data Transmission Process

404 page when the link expires. The uploader controls the expiration time of the link. Users are not required to install a client-side app to access any feature of the netdisk service.

1.6 Privacy Guarantee and Censorship Resistance

While AirX guarantees that user data will be strongly encrypted before uploading to the server and can only be decrypted by the user, there are still potential concerns:

- Unverifiability: Users may not trust the server to not exploit their data.
- Policy facts: The server may be compelled to disclose users' identities by related authorities.

To address these issues, AirX is open source and offers a private server deployment service.

1.7 Open Source and Private Server Deployment Service

AirX is open source under the MIT License. It also provides a private server deployment service based on the open-source version of the server-side implementation. Users can deploy their own server and enjoy a similar experience as with the public server. By deploying their own server, users can verify the server-side implementation and ensure their privacy.

1.8 Encryption over Secure Tunnel

A popular netdisk service, mega.nz, uses AES (Advanced Encryption Standard) encryption over an HTTPS (Hypertext Transfer Protocol Secure) channel to guard against potential security exceptions like middleman attacks. Similarly, AirX uses RSA-2048 (Rivest-Shamir-Adleman Encryption Algorithm 2048-bit) to encrypt files and clipboard data before uploading to the server. It creates private and public key pairs for each user upon account registration. The private key is stored locally on the client's device, and the public key is uploaded to the server. This theoretically guarantees that only the user can decrypt their own data.

2 State of the Art

AirX contains multiple components, which are related to different areas of software development. This section will discuss the current state of the art in each of these areas as a literature review.

The section will be organized as follows:

- for libairx, the shared library for AirX that implements device discovery and data transmission over LAN:
  - Device discovery over LAN
  - Reliable Data Transmission over LAN
- for Windows client:
  - WinUI 3, the modern Windows desktop development framework
  - Tray Icon and Context Menu
- for macOS client:
  - SwiftUI, the modern macOS development framework
- for Android client:
  - React Native, a cross-platform mobile development framework
  - Android Security and Permissions
- for backend services:
  - Netdisk Service: Hash Match
  - User Authentication and Authorization
  - Message Queue

2.1 Device Discovery over LAN

Device discovery over a Local Area Network (LAN) is a fundamental aspect of network communication. It allows devices within the same network to identify and communicate with each other. One of the most commonly used methods for device discovery over LAN is the User Datagram Protocol (UDP) broadcast. This method was highlighted in the implementation by Wu, Jie et al. [1]. They pointed out the challenge of data exchange between devices without prior knowledge of each other's IP addresses.

The UDP broadcast method addresses this challenge by sending a UDP packet to the broadcast address. This action allows the sender to reach all devices within the same network segment without the need to know the IP address of each neighboring device. The receiving devices can then respond to the sender with their respective IP addresses, enabling the sender to establish direct communication with each of them. This approach was adopted in the development of libairx, the shared library for AirX that implements device discovery and data transmission over LAN.

In addition to UDP broadcast, Wittmann et al. introduced the concept of UDP multicast [2]. This method offers more flexibility than broadcast as it allows the sender to specify the target devices. However, it requires the router hardware to support multicast, which is not always guaranteed in every network environment.

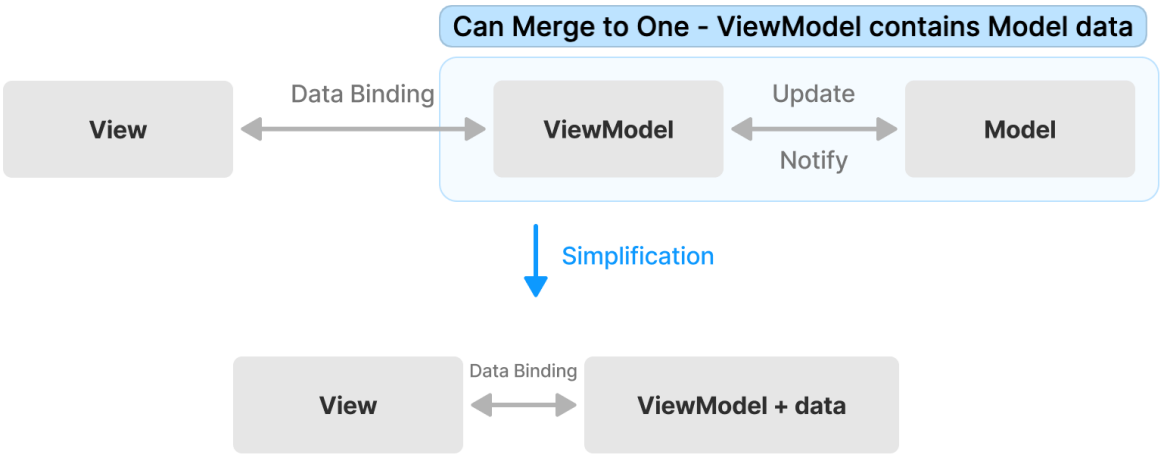


Figure 2: Simplified MVVM Architecture: ViewModel containing both business logic and data

2.2 Reliable Data Transmission over LAN

Reliable data transmission is another crucial aspect of network communication. The Transmission Control Protocol (TCP) is a widely used protocol for ensuring reliable data transmission. As highlighted by Pakanati Chenna Reddy et al., TCP incorporates mechanisms for resending data, checksum, and flow control [3]. These mechanisms ensure the successful delivery of data in most scenarios.

However, TCP can fail under certain circumstances. For instance, W. Feng et al. suggested that TCP might not be reliable in distributed computational grids [4]. In such scenarios, the network conditions can be highly variable and unpredictable, leading to potential data transmission failures. Therefore, while TCP is a robust protocol for reliable data transmission over LAN, it is essential to consider the specific network conditions and requirements when choosing the appropriate data transmission protocol.

libairx tried to mitigate this problem by implementing application layer resend and checksum over Transmission Control Protocol (TCP).

2.3 WinUI 3: A Modern Framework for Windows Desktop Development

In the ever-evolving landscape of Windows desktop development, the introduction of the Windows App SDK by Pagani et al. represents a significant leap forward. The Windows App SDK, a state-of-the-art Windows desktop development framework, enables developers to build native Windows applications that incorporate Fluent Design [5].

WinUI 3, a subset of the Windows App SDK, is particularly noteworthy in this context. It relies on Extensible Application Markup Language (XAML), a markup language specifically designed for crafting user interfaces in Windows apps.

Designing user interfaces with XAML has a rich legacy in Windows development, tracing back to the launch of the Windows Presentation Foundation (WPF) in 2006. When designing the Windows client for AirX, the author referred to multiple resources related to XAML. In Filipova-Petrakieva et al.'s paper, the authors provide a detailed explanation of project structures using view models, a feature central to the Model-View-ViewModel (MVVM) architecture employed by both WinUI 3 and the Windows client of AirX, as depicted in figure 2.

It is noteworthy that the author of AirX has chosen to use a simplified form of the MVVM architecture. This adaptation maximizes the benefits of observable objects while avoiding over-complication by dispensing with the need for a seperate model layer.

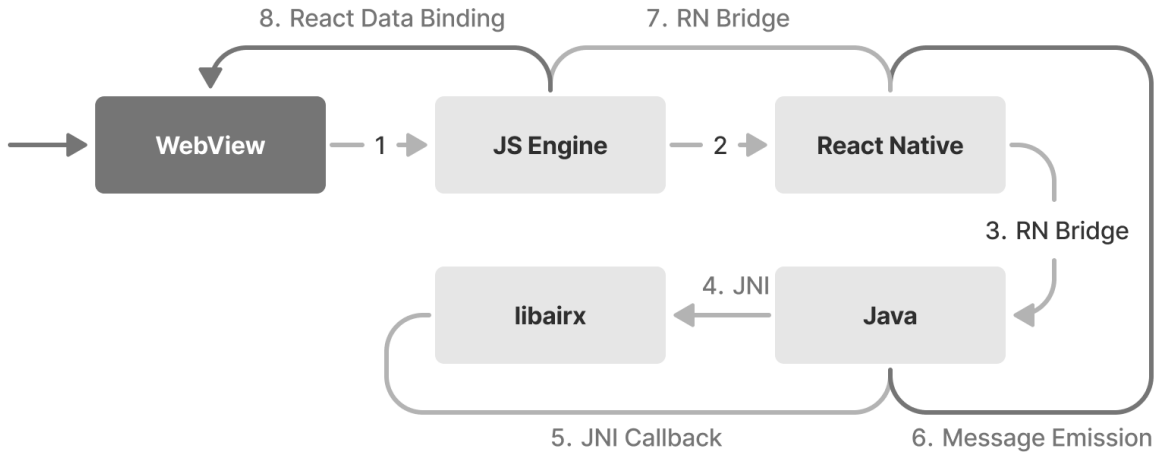


Figure 3: Android Client Architecture: How Webpage Calls libairx and Obtain Return Value and Callback. RN Bridge represents React Native Bridge.

2.4 Windows Tray Icon and Context Menu: An Enduring Feature

Despite Microsoft's efforts to dissuade developers from using the tray icon in Windows 11, it remains a highly demanded feature among users. This preference among users can be attributed to the extensive history and familiarity of the Windows operating system. Therefore, it makes sense to include this feature in AirX.

While no official publications explicitly advise against the use of the tray icon in Windows 11, it is worth noting that the Windows App SDK has discontinued the provision of APIs for tray icons and context menus.

2.5 SwiftUI: The Future of macOS Development

SwiftUI, a declarative framework for building user interfaces for Apple platforms, has emerged as the natural successor to UIKit [6]. In a study by Wiertel et al., they demonstrated that compared to UIKit, SwiftUI showed superior performance when rendering complex windows populated with user controls.

2.6 React Native: Bridging the Gap in Cross-Platform Mobile Development

React Native is a cross-platform mobile development framework that empowers developers to build native mobile applications using JavaScript and React [7]. It essentially operates as a web view, rendering the user interface as a web page, and therefore shares the same performance challenges associated with web applications.

Despite these performance issues, React Native, as a high-level framework, has the capability to call Android Java methods from JavaScript. This bridge enables a web page to execute libairx functions through the Java Native Interface (JNI), as shown in figure 3.

Although this call is feasible, it introduces a performance bottleneck. Unlike on Windows or macOS clients, where calling libairx is a simple and fast "CALL" CPU instruction, on an Android client, the call must first be processed by React Native. This process involves the creation of a Promise object and the corresponding Java object, which is then injected into the Java-side handler method, followed by waiting for the promise to resolve. For callback functions, Java-side code needs to create an intermediate data structure ("WritableMap"), necessitating data copy. This information is then passed to the JavaScript

side via a callback event, awaiting processing by React Native's message queue. This roundabout procedure is significantly slower and leads to perceptible delays.

2.7 Android Security and Permissions: Implications for AirX

The robust security protocols deployed by Google on Android devices pose certain limitations on the functionality of applications like AirX. Specifically, these protocols prevent access to clipboard data from the background. The reason behind this restriction is a privacy-oriented design philosophy that Google has implemented to prevent unauthorized access to sensitive user data.

In the case of AirX, this security restriction results in the application only being able to access and read clipboard data when it is actively opened and running in the foreground, i.e., when a user manually opens the AirX application from the launcher. This means that clipboard data can't be read in the background or when the application is minimized.

One significant impact of this restriction is that it eliminates the possibility of AirX monitoring clipboard data passively and sharing this data automatically with nearby devices. This limitation poses a challenge to AirX's intended functionality, as the feature to monitor and share clipboard data across devices is a central aspect of the application's design and appeal.

2.8 Netdisk Service and the Principle of Instant Upload

Cloud storage services have revolutionized the way users store and access data. One significant innovation in this sphere is the practice of performing a hash calculation on a file before it is uploaded. The server checks if a file with the same hash already exists in the system. If it does, instead of uploading a duplicate of the file, the server simply links the pre-existing file to the user's account. This process is often referred to as "instant upload" because, from the user's perspective, the file appears to be uploaded to the server instantly, regardless of the actual size of the file.

In a paper by M Kathiravan et al., they explore this fast-deduplication algorithm in detail, revealing that the algorithm typically used is the MD5 hash function. However, in the case of AirX, the more advanced SHA-256 hash function is used. SHA-256, or Secure Hash Algorithm 256, is part of the SHA-2 family, which is known for its increased security and is extensively used in cryptographic applications and protocols.

The use of SHA-256 ensures an added layer of security for the files processed by AirX, enhancing the reliability and trustworthiness of the application. Although this might seem like a minor detail, it reflects the commitment of the AirX developers to data security and integrity, providing users with peace of mind when using the application.

2.9 User Authentication and Authorization

In the area of user authentication and authorization, JSON Web Token (JWT) has emerged as a widely accepted standard. This was explored in depth by Haekal et al. in their paper, where they looked into the complexities of JWT and its applications in web development [8]. JWT, also known as RFC 7519, is a standard that facilitates the secure transmission of JSON objects between parties. The security of these transmissions is ensured through a signature algorithm that verifies the authenticity of the JWT and checks for any data tampering using a secret key. Two commonly used signature algorithms are HS256 and RS256. HS256, a symmetric algorithm, uses the same secret key to issue

and verify the signature. On the other hand, RS256, an asymmetric algorithm, employs a set of public and private keys, using the public key for issuing and the private key for verification.

2.9.1 Purpose of Signature

The purpose of the signature in JWT is elaborated in the paper. The signing process involves signing both the header and the payload content. Cryptographic algorithms are designed to produce different outputs for different inputs, making it highly unlikely for two different inputs to produce the same output. Therefore, if someone decodes the header and payload, modifies them, and then encodes them again, the new header and payload signatures will differ from the original ones. Furthermore, without knowledge of the server's encryption key, the resulting signature will inevitably be different.

Upon receiving the JWT, the server application re-signs the header and payload using the same algorithm. If the newly calculated signature differs from the received one, it indicates that the token content has been tampered with. In such cases, the server should reject the token and return an HTTP 401 Unauthorized response.

2.9.2 Insights from the Authors

JWT is a lightweight specification that enables secure information exchange between users and servers.

The authors, Muhamad Haekal and Eliyani, proposed a scenario where a user follows another user, and the system sends an email to the followed user with a link to confirm the action. The traditional approach requires the followed user to log in first, which can be cumbersome. The authors posed the question: can this process be simplified to allow the followed user to confirm the action without logging in? Given the characteristics and structure of JWT, it is indeed possible.

The authors' analysis suggests that JWT is particularly suitable for transmitting non-sensitive information to web applications. This includes operations such as adding friends, placing orders, and more. The most common use case for JWT is authorized login. Once a user logs in, each subsequent request will include the JWT, granting the user access to the routes, services, and resources permitted by the token. Moreover, JWTs provide a secure means of transferring information between parties. Since JWTs can be signed, for instance, using a public/private key pair, the sender's identity can be verified. Additionally, the integrity of the content can be ensured due to the use of signatures computed from headers and payloads.

2.9.3 Comparison of JWT and Traditional Authentication

The HTTP protocol, by its nature, is stateless, meaning that each request is processed independently without any knowledge of previous requests. This poses a challenge for user authentication, as each request needs to be authenticated separately. In traditional session authentication, user login information is stored on the server and passed to the client in the form of a cookie. This cookie is then included in subsequent requests, allowing the server to identify the user making the request.

However, this approach has its limitations. For each authenticated user, the server needs to maintain a record, which can lead to significant overhead as the number of authenticated users increases. Furthermore, since the session information is stored in memory, subsequent requests from the user need to be routed to the same server, limiting the scalability of the application and the capacity of load balancers in distributed applications.

JWT addresses these issues effectively. Upon successful authentication, the server generates a JWT containing the user's ID and other information, signs it, and returns it to the client. The client then includes this JWT in subsequent requests, allowing the server to verify the user's identity without needing to maintain a session record. This not only reduces server overhead but also allows for greater scalability and flexibility in distributed applications.

2.9.4 Potential Flaws in JWT

Despite its advantages, JWT is not without its flaws:

- **Message Exposure**: JWTs are encoded using Base64, which is not a form of encryption but merely a way of representing binary data in an ASCII string format. This means that anyone can decode the header and payload of a JWT, potentially exposing sensitive information. Furthermore, if an attacker can obtain the secret key, they can tamper with the JWT's content, posing a significant security risk.
- **Unrevocability**: Once issued, a JWT remains static and valid until its expiration. This means that any changes in user privileges or account status during the JWT's lifetime will not be reflected in the JWT itself. Furthermore, there is no built-in mechanism in JWT for instant revocation or invalidation. As a result, a compromised JWT can continue to be used until it expires, even if the user's account has been disabled or logged off.

Despite these potential flaws, JWT remains a powerful tool for user authentication and authorization, offering a more efficient and scalable alternative to traditional session-based authentication. However, it is crucial to implement proper security measures, such as secure transmission of JWTs and regular rotation of secret keys, to mitigate the risks associated with message exposure and unrevocability.

2.10 Backend Server Message Queue

Apache Kafka, a distributed message queue system, has gained significant popularity in the realm of backend services. This is largely due to its robust design principles that emphasize scalability and fault tolerance, making it an ideal choice for distributed systems. These principles were discussed in a paper authored by Wang, Guozhang and their team, where they analyzed the details of Apache Kafka's design and its implications for distributed systems [9].

The concept of a distributed message queue is a fundamental aspect of Apache Kafka. In essence, it is a system that allows for the storage and retrieval of messages across a distributed network. This is particularly useful in scenarios where there is a need to process large volumes of data in real-time. The distributed nature of Kafka ensures that data can be processed in parallel across multiple nodes, thereby improving the overall efficiency and throughput of the system.

Apache Kafka's design is inherently scalable. This means that as the volume of data or the number of transactions increases, the system can be easily expanded by adding more nodes to the network. This scalability is not just limited to handling larger volumes of data, but also extends to the ability to serve a larger number of consumers or producers. This makes Apache Kafka a versatile solution that can adapt to the changing needs of a business.

Fault tolerance is another key feature of Apache Kafka. In the event of a failure, Kafka's design ensures that there is no data loss and the system can continue to function effectively. This is achieved

through data replication across multiple nodes, which means that even if one node fails, the data is still available on other nodes. This redundancy is crucial in maintaining the integrity and reliability of the system.

In the context of AirX, Apache Kafka plays a pivotal role due to its efficient and distributed message queue. This middleware not only facilitates the efficient processing of messages but also contributes to the clarity of the code logic for subscribers. The use of Apache Kafka in AirX allows for a more streamlined and efficient data flow, thereby improving the overall performance of the application.

3 Implementation

This section will focus on the intricate implementation details of AirX, offering a detailed understanding of its inner workings. The structure of this section is organized as follows:

- **Architecture Design**: This subsection provides an overview of the overall structure of AirX. It elucidates how various components within the system are designed to interact with each other, forming a cohesive and efficient network for data sharing.
- **libairx**: This part of the section focuses on libairx, a shared library that forms the backbone of all LAN-based functionalities in AirX. We will explore how libairx contributes to the efficient functioning of the system.
- **Client Implementations for Windows, macOS, and Android**: This subsection delves into the specifics of how AirX is implemented on various platforms. We will discuss the unique aspects of each platform and how AirX has been tailored to function optimally on Windows, macOS, and Android devices.
- **Backend Server**: The final part of this section illuminates the inner workings of the backend server of AirX. We will discuss how the server manages data flow, ensures efficient communication between different devices, and maintains the overall performance and reliability of the system.

Each of these subsections will provide a detailed exploration of the respective components, offering a thorough understanding of how AirX achieves efficient cross-platform data sharing. By the end of this section, readers should have a clear picture of the technical prowess that underpins AirX, and how it contributes to enhancing the efficiency of our digital interactions.

3.1 Architecture Design

Figure 4 illustrates how users interact with AirX and figure 5 shows the architecture of AirX. The technology used for each component is shown in figure 6.

3.1.1 Users within the Same LAN

In the context of AirX, clients operating under the same subnetwork have the capability to publicize their physical addresses to each other. This feature facilitates seamless communication and data sharing among devices within the same LAN.

When a user updates their clipboard, the new text is automatically broadcasted to the clipboards of all other devices within the same subnetwork. This silent broadcasting ensures that the updated text is instantly available across all devices, enhancing the efficiency of data sharing.

File sharing within the same LAN is handled on a point-to-point basis. This means that files are directly transferred from one device to another without the need for an intermediary. However, to ensure security and user control, a transmission can only proceed once the receiving device has granted permission.

3.1.2 Users across Different LANs

For clients operating on different LANs, AirX utilizes a centralized public server to facilitate communication and data exchange. This server acts as a bridge, connecting devices across different LANs and enabling them to share data efficiently.

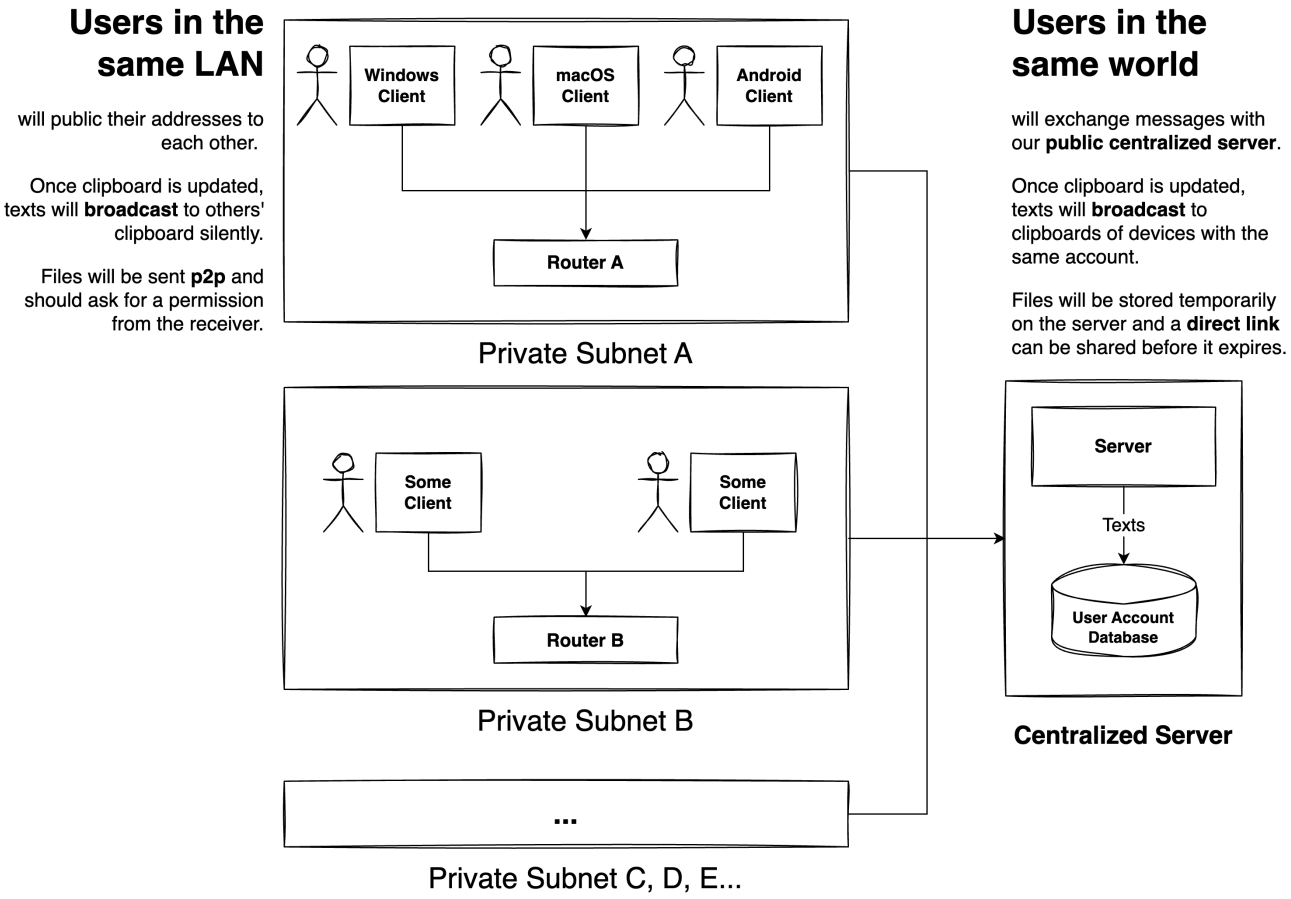


Figure 4: How Users Interact with AirX

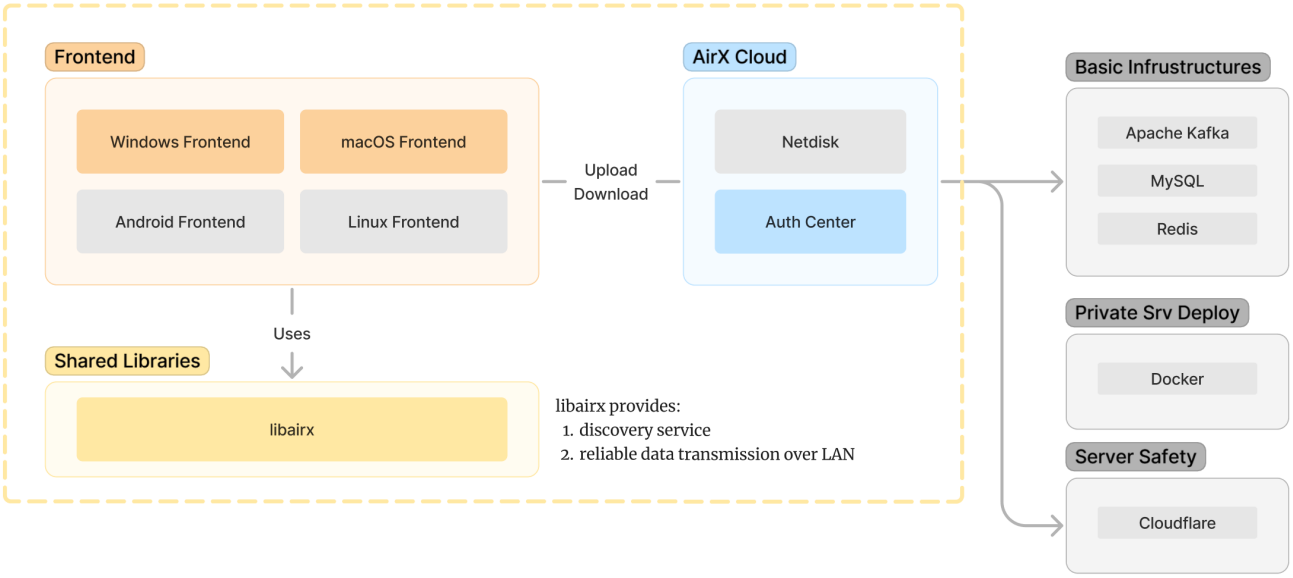


Figure 5: Architecture of AirX

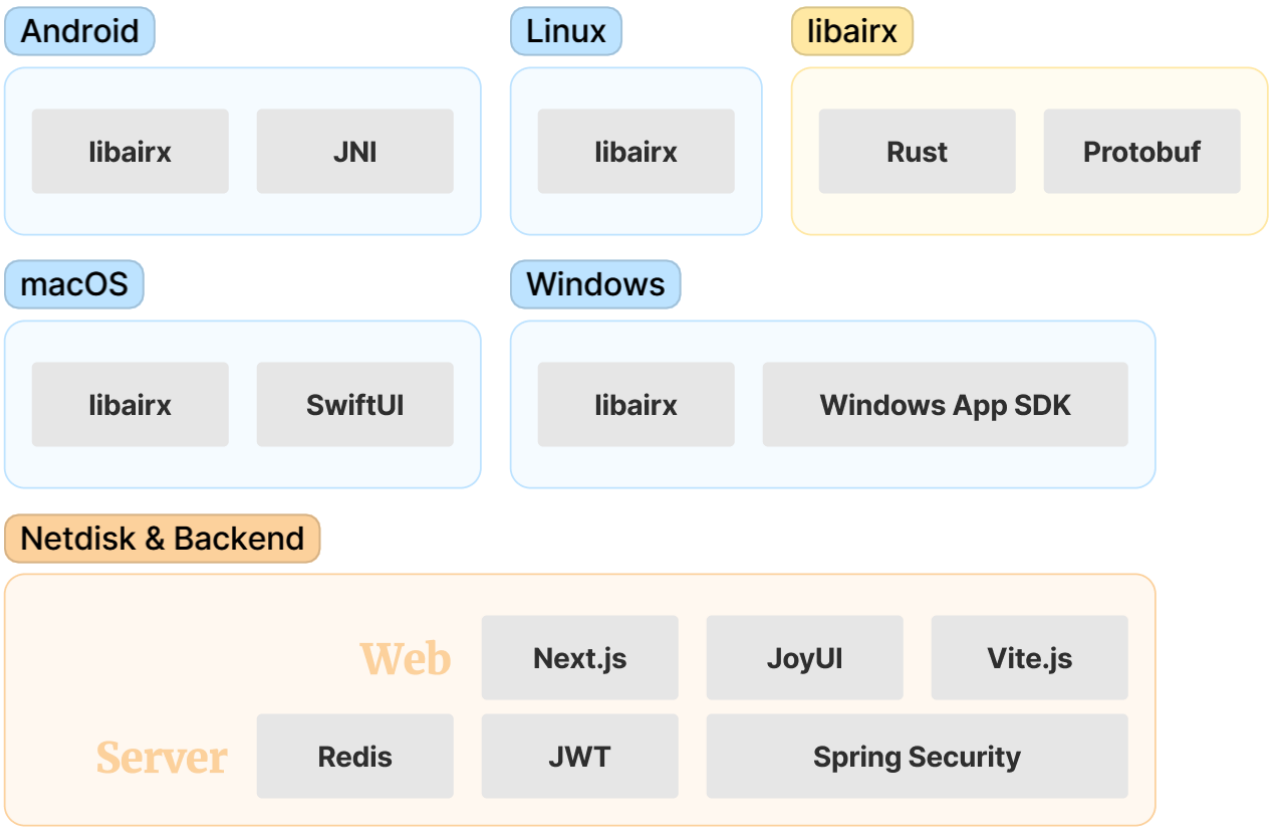


Figure 6: Technology Stack of AirX

When a user updates their clipboard, the new text is broadcasted to the clipboards of all other devices that are logged into the same account, regardless of the LAN they are on. This ensures that the updated text is instantly available across all devices associated with the user's account.

File sharing across different LANs is handled through temporary storage on the server. When a user uploads a file, it is temporarily stored on the server and a share link is generated. This share link provides direct access to the file, making it easy for other users to download the file.

However, if the link expires, users will be directed to a user-friendly 404 page. This ensures that users are not left confused or frustrated when trying to access an expired link.

To ensure security and user control, the uploader has the ability to control the permissions of the file. They can set a password to protect the file and specify an expiry date for the share link. This gives the uploader full control over who can access the file and for how long, thereby enhancing the security of file sharing in AirX.

3.2 libairx

libairx is the shared library of AirX serving as the backbone for providing an efficient discovery service and ensuring the reliable transmission of data across Local Area Networks (LANs) to all AirX clients. It contains wait-free design and is thread-safe.

Playing a central role in our application, libairx facilitates seamless communication and data exchange among diverse client devices. The discovery service it offers allows clients to effortlessly locate and establish connections with each other within the same LAN, fostering a cohesive and interconnected user experience.

Data Packet

2 Bytes	8 Bytes	N Bytes	2 Bytes
Magic Number	Data Length	Raw Data	hash(length)

Figure 7: Generic Data Packet

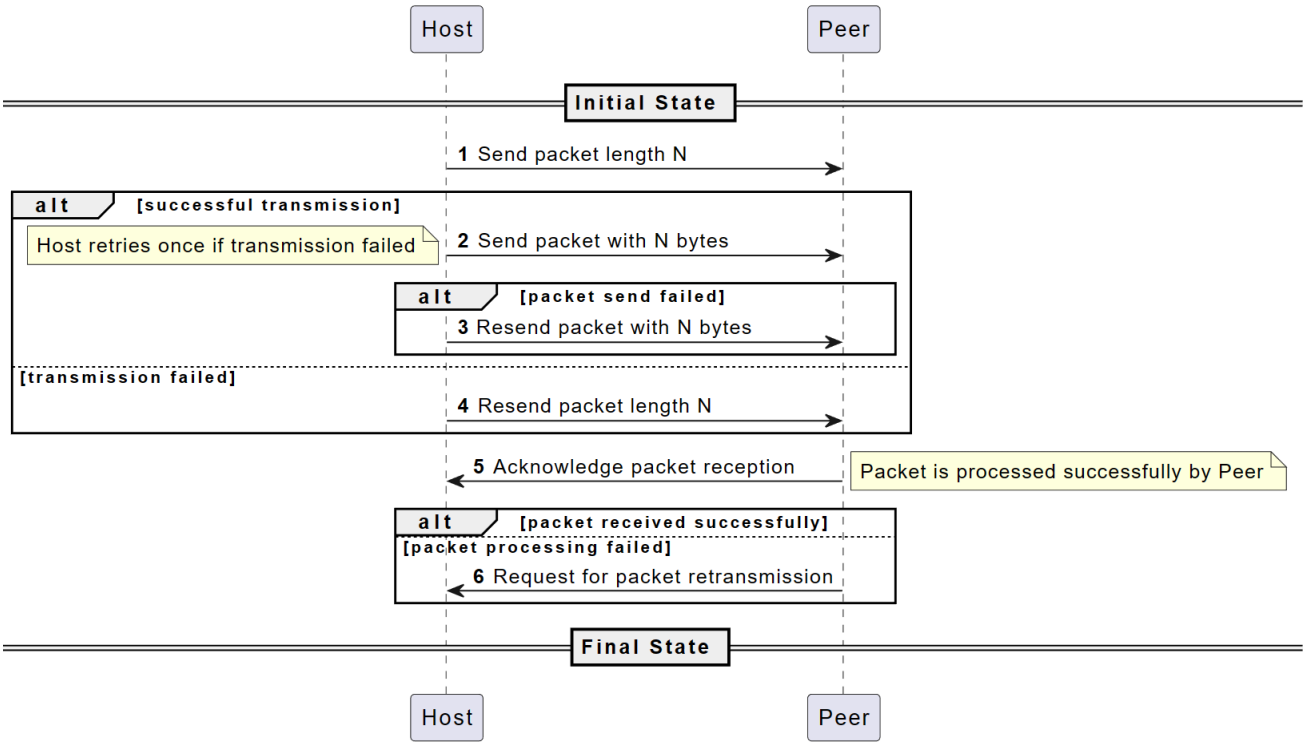


Figure 8: Data Packet Transmission Sequence

3.2.1 Data Service: Reliable Data Transmission

In the shared library's implementation, UDP multicast (currently broadcast) is utilized to enable the efficient discovery service, while TCP takes charge of the data service. This two-pronged approach leverages the strengths of both protocols to achieve optimal performance and reliability.

To ensure the integrity and reliable delivery of data over UDP and TCP, a custom data format with a resend mechanism is employed. Figures 7 and 8 illustrate the structure of the data packets and the sequence diagram for data transmission.

In cases where packet processing failure occurs due to peer inability to deserialize the packet or verify the hash, the resend mechanism is triggered. Upon detecting a corrupted packet, the receiver explicitly sends a resend request, prompting the sender to retransmit the data. This proactive approach ensures that the data is successfully delivered, even in the face of occasional corruption during transmission.

The resend happens when the receiver explicitly sends a resend request if the packet is corrupted. For failed deliveries that receiver can not detect, the resend task will be handled by TCP itself or no resend regarding UDP.

Text Packet

4 Bytes	N Bytes	2 Bytes
Text Length (UTF-8)	Text (UTF-8)	hash(length)

File Sending Request Packet

8 Bytes	4 Bytes	N Bytes	2 Bytes
File Size (Bytes)	File Name Length (UTF-8)	File Name (UTF-8)	hash(size,len)

File Receive Response Packet

1 Byte	8 Bytes	4 Bytes	N Bytes	1 Bytes
File ID	File Size	File Name Length (UTF-8)	File Name (UTF-8)	Accept?

File Part Packet (No rehash for performance)

1 Byte	8 Bytes	8 Bytes	N Bytes
File ID	Offset	Length	Data

Data Packet in Use

2 Bytes	8 Bytes	N Bytes	2 Bytes
Packet Type as Magic Number	Wrapping Packet Size	Wrapping Packet. For example, a File Part Packet.	hash(length)

Figure 9: All Packets and Data Packet In Use

Discovery Packet (Proto)

Slot#	Name	Type
1	address	uint32
2	server_port	uint32
3	group_identifier	uint32
4	need_response	bool
5	host_name	string

Figure 10: Discovery Packet (Protobuf)

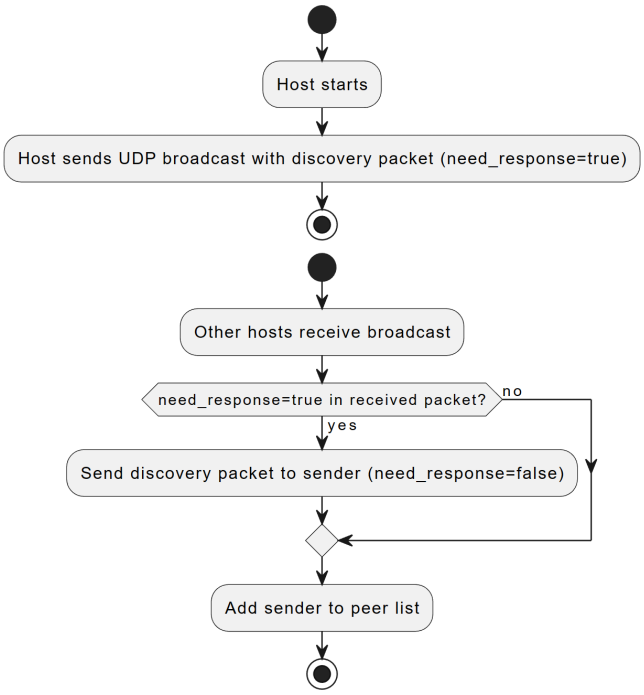


Figure 11: Discovery Procedure

3.2.2 Data Service: Rehash

The data packet shown in Figure 7 is a versatile packet format designed to encapsulate other packets for various purposes within the shared library, as illustrated in Figure 9. This data packet plays a crucial role in determining the type of the containing packet and calculating its hash using an extremely simple and efficient hash function, ensuring optimal performance with constant time complexity O(1).

Even though TCP already provides reliability in data transmission, some low-frequency packets still include their own hash bytes as an additional layer of protection. This rehash mechanism is implemented to safeguard against potential data corruption caused by malicious middlemen.

3.2.3 Discovery Service and Google Protocol Buffers

The structure of a discovery packet and the associated discovery procedure are illustrated in Figure 10 and Figure 11, respectively. To achieve minimal packet size, guaranteed integrity, and ease of extensibility during UDP transmission, the shared library utilizes a specialized discovery packet based on Google Protocol Buffers (protobuf).

Google Protocol Buffers (protobuf) is a language-agnostic data serialization format renowned for its efficient communication and storage of structured data. It offers a simple and flexible mechanism to define data structures, enabling the generation of code in various programming languages for seamless integration into diverse applications. However, protobuf does not inherently provide a data integrity check.

By using protobuf in conjunction with the generic data packet, the discovery packet benefits from minimized size, assured integrity, and enhanced resilience against corruption during UDP transmission. This combination provides an ideal solution for discovery services, promoting interoperability and efficient communication in complex network environments.

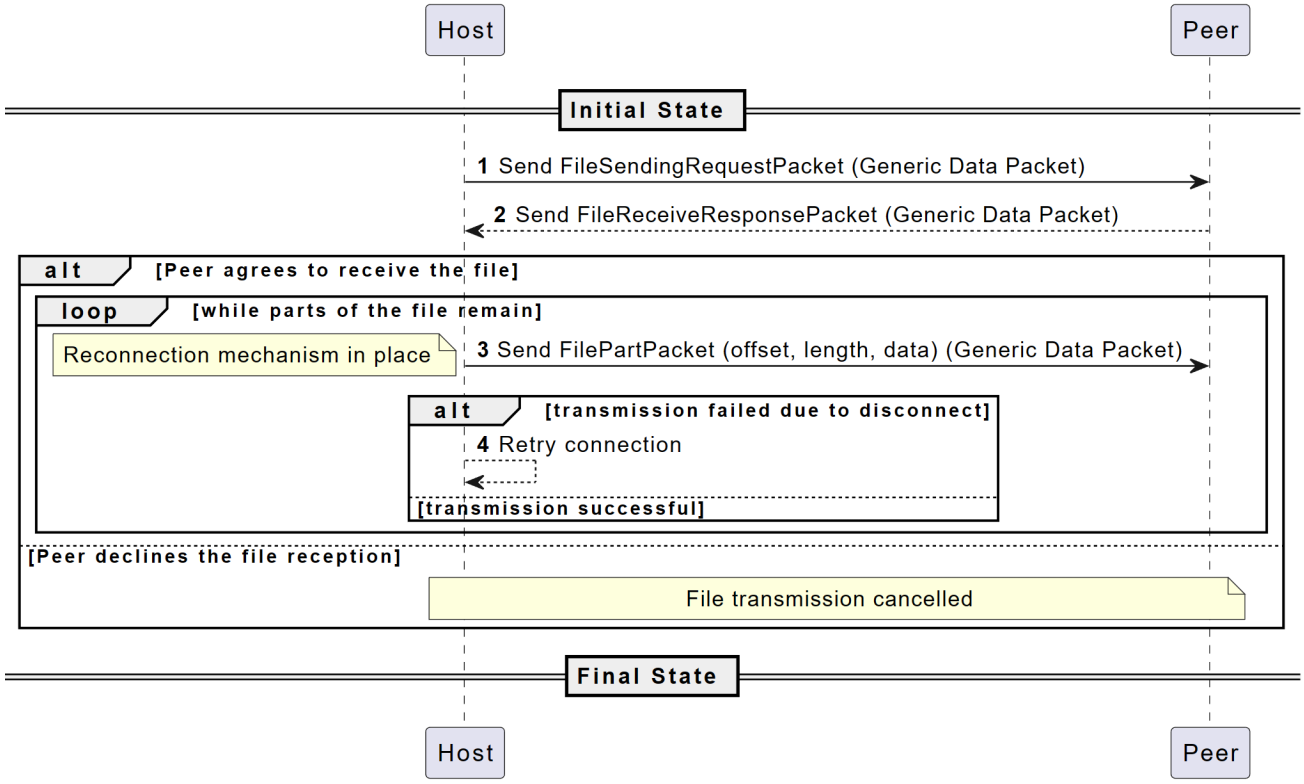


Figure 12: File Transfer Sequence

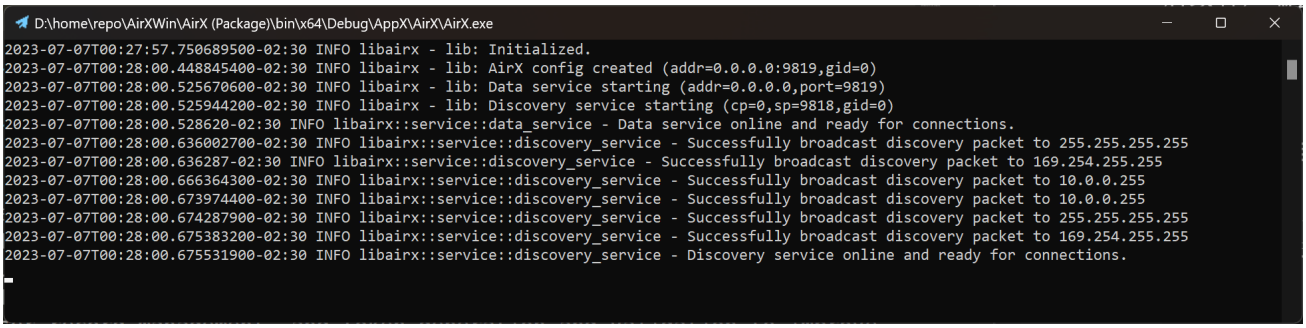


Figure 13: Logging in libairx

3.2.4 Data Service: File Transmission

Figure 12 illustrates the sequence of file transmission, where the file is divided into smaller chunks. Each chunk is then encapsulated first in a file part packet and subsequently in a data packet. The file size can be of any magnitude because the shared library adopts a memory-efficient approach that avoids loading the entire file into memory at once.

During the file transmission process, interruptions may occur due to connection losses. However, the shared library is equipped with an automatic reconnection mechanism that enables it to resume the transmission seamlessly from the point where it was last successful. To ensure data integrity, the receiver diligently discards any duplicated chunks it might receive.

Similar to the hash function, the serialization and deserialization procedures of the file part packet have been meticulously designed to be exceedingly simple and efficient, operating with a time complexity of O(n). This design choice is instrumental in maintaining optimal performance throughout the file transmission process, even for substantial file sizes. The streamlined serialization and deserialization processes minimize computational overhead and facilitate faster data transfer, contributing to an overall robust and reliable file transmission mechanism.

3.2.5 Trace-level Logging

For debugging purposes, the shared library offers trace-level logging, as depicted in Figure 13. This logging feature allows developers to obtain detailed information about the internal workings of the library. The logs are printed to the standard output (stdout) and can be easily redirected to a file if desired. By enabling trace-level logging, developers can gain insights into the library's execution flow, variable values, and other relevant diagnostic information, which proves invaluable in identifying and resolving issues during development and testing.

3.2.6 Discovery Service: Group Identifier

The shared library incorporates a significant feature known as the Group Identifier (GID) within its packets. The GID serves the purpose of dividing a Local Area Network (LAN) into distinct user groups. When a device transmits packets, it includes its designated GID within the packet's metadata. On the receiving end, the shared library examines the GID of incoming packets and selectively discards any packet that does not belong to the same group as the receiver.

The incorporation of the Group Identifier ensures that only devices within the same user group can discover and communicate with each other. This adds a layer of security and isolation, preventing unwanted devices from interfering with the communication process. Consequently, devices within the same GID can effectively discover and establish connections with one another while remaining isolated from devices in other groups. This feature is particularly useful in scenarios where multiple user groups exist within the same LAN and need to communicate separately and securely.

3.2.7 Unified Endian

To address the issue of different endianness on various platforms, the libairx library employs a *Unified Endian* mechanism. This approach ensures consistent and predictable byte order for data serialization and deserialization across all platforms.

The library defines a trait called `UnifiedEndian`, which is parameterized by the constant `SIZE`, representing the size of the numeric type in bytes. The trait provides two methods: `to_bytes()` and

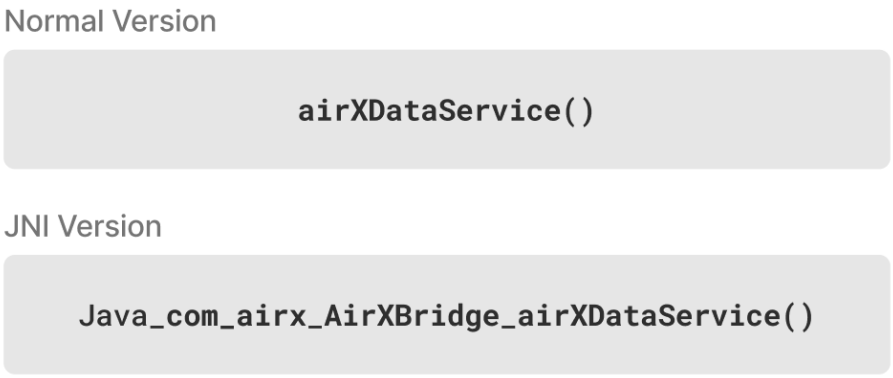


Figure 14: JNI Integration

`from_bytes()`. These methods are responsible for converting numeric types to and from byte representations, ensuring data consistency during communication between systems with different endianness.

To make the implementation easier, a macro called `impl_unified_endian` is used. It simplifies the process of defining the `UnifiedEndian` trait for multiple numeric types such as `u8`, `u16`, `u32`, `u64`, `u128`, `usize`, `i8`, `i16`, `i32`, `i64`, `i128`, and `isize`.

With the Unified Endian mechanism in place, the libairx library can confidently perform data serialization and deserialization in little-endian format, regardless of the underlying platform's endianness. This ensures that data exchange between systems with different endianness remains accurate and seamless, promoting platform independence and consistent behavior across all supported architectures. As a result, data communication and handling within the library are optimized for performance and compatibility across a wide range of platforms.

3.2.8 JNI Integration

To facilitate usage on Android platforms, the libairx library has been enhanced with JNI Integration. The library already contains a set of exported functions that can be utilized by other client applications. However, to make these functions accessible to Android applications, JNI versions have been added to each of the existing exported functions. Figure 14 illustrates a pair of exported functions and their corresponding JNI versions.

The Java Native Interface (JNI) is a programming framework that enables seamless communication between Java code and native code written in languages like Rust. By incorporating JNI versions of the existing exported functions, the libairx library becomes compatible with Android's Java-based environment.

Now, Android applications can call the JNI versions of the libairx functions to leverage the library's capabilities seamlessly. This integration opens up new possibilities for Android developers to utilize the power and features of the libairx library directly from their Java code. The JNI integration ensures smooth interaction between the Java-based Android application and the native functionality provided by libairx, expanding the library's reach and enhancing its applicability across different platforms.

3.2.9 C Header Binding Generation with cbindgen

Cbindgen is a powerful tool that simplifies the process of generating C header bindings from Rust code. Its primary goal is to facilitate the seamless integration of Rust libraries into C-based projects or other languages with C-compatible interfaces. In the context of libairx, cbindgen is used to generate

Rust Function

```
fn airx_version() -> u32;
```

C Binding

```
int airx_version();
```

Figure 15: C Header Binding Generation with cbindgen

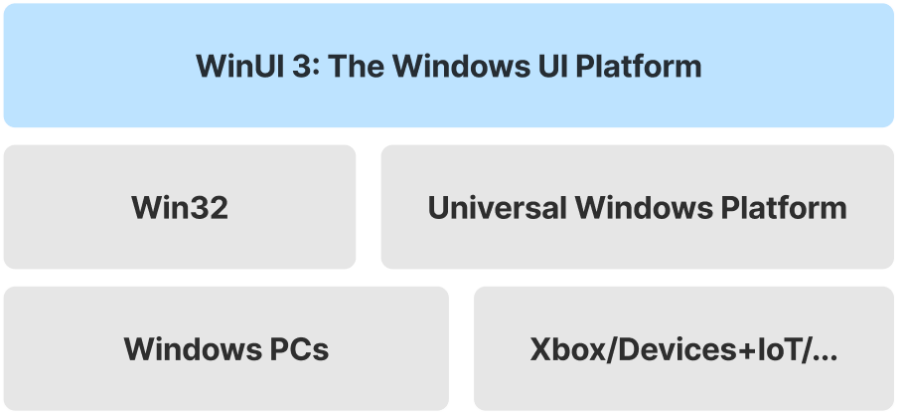


Figure 16: WinUI 3 Hierarchical Architecture

C header bindings for the shared library, enabling it to be easily integrated into other projects, as illustrated in Figure 15.

When using cbindgen, developers can rely on the tool's ability to analyze Rust code and extract relevant information about types, functions, and other symbols. With this information, cbindgen then automatically generates a corresponding C header file that accurately represents the Rust library's interface.

The generated C header file ensures type safety and preserves the semantics of the original Rust code. This aspect is particularly valuable when dealing with complex data structures and function signatures, as cbindgen handles the translation between Rust and C representations.

By employing cbindgen, Rust developers can avoid the tedious and error-prone task of manually writing Foreign Function Interface (FFI) code, thereby reducing potential sources of bugs and making the integration process smoother.

3.3 Windows Client

The Windows client for AirX is developed using WinUI 3, the latest user interface framework from Microsoft, designed to create modern and native Windows applications. It provides a rich set of controls and APIs, allowing developers to build visually appealing and responsive user interfaces for Windows applications.

3.3.1 Front-end implementation: WinUI 3

WinUI 3 serves as a native user experience (UX) framework tailored for Windows applications, catering to both desktop and Universal Windows Platform (UWP) apps. Incorporated within the Windows App SDK, it presents developers with an efficient and robust platform to develop production-ready desktop apps compatible with Windows 10 and Windows 11. These apps can be conveniently published to the Microsoft Store, expanding their reach to a vast user base.

The flexibility of WinUI 3 accommodates the use of various programming languages, including C++ and C#, allowing developers to choose their preferred coding environment. Furthermore, it facilitates seamless migration of existing Windows Forms (WinForms) or Windows Presentation Foundation (WPF) apps, streamlining the transition process to this modern UX framework. Hence, WinUI 3 is an excellent tool for the creation of sophisticated and user-friendly applications that run on the latest Windows devices. Figure 16 illustrates the hierarchical architecture of WinUI 3 - it is built on top of the Win32 and Windows App SDK, which in turn is built on top of the Universal Windows Platform (UWP).

The WinUI 3 platform offers several significant advantages that make it a standout choice for application development:

- **Modern GUI Design:** WinUI 3 facilitates the creation of contemporary and visually appealing applications. With its modern GUI design, applications remain in sync with the latest control versions and visual elements, without the necessity of an updated Windows SDK.
- **Operating System Versatility:** WinUI 3 possesses the flexibility to adapt and function seamlessly across different versions of the operating system, catering to a broad range of user requirements.
- **Backward Compatibility:** WinUI ensures compatibility with numerous versions of Windows 10. This feature allows developers, even those not utilizing the most recent Windows 10 version, to build and publish applications equipped with innovative XAML features as soon as they are released.
- **Elimination of Version Checks:** Applications constructed with WinUI don't require version checks to employ key controls or UWP XAML features, streamlining the user experience and reducing potential compatibility issues.
- **Leveraging Extensible Application Markup Language (XAML):** XAML is a potent tool that simplifies the creation of user interfaces. By segregating user interfaces from the program logic, it enables an extensible and searchable syntax. It's an analytical language that makes the user creation process easier, seamlessly incorporating code and configuration into your application.
- **Independent UI Updates:** WinUI empowers developers to utilize the latest UI controls without the need for updating their Windows version. This independence provides developers with the freedom to continually improve their applications with cutting-edge user interface tools.

3.3.2 Account Login

The login window, shown in figure 18, mainly realizes the connection between the front and back ends, pops up automatically when a user failed to login with the saved credentials.

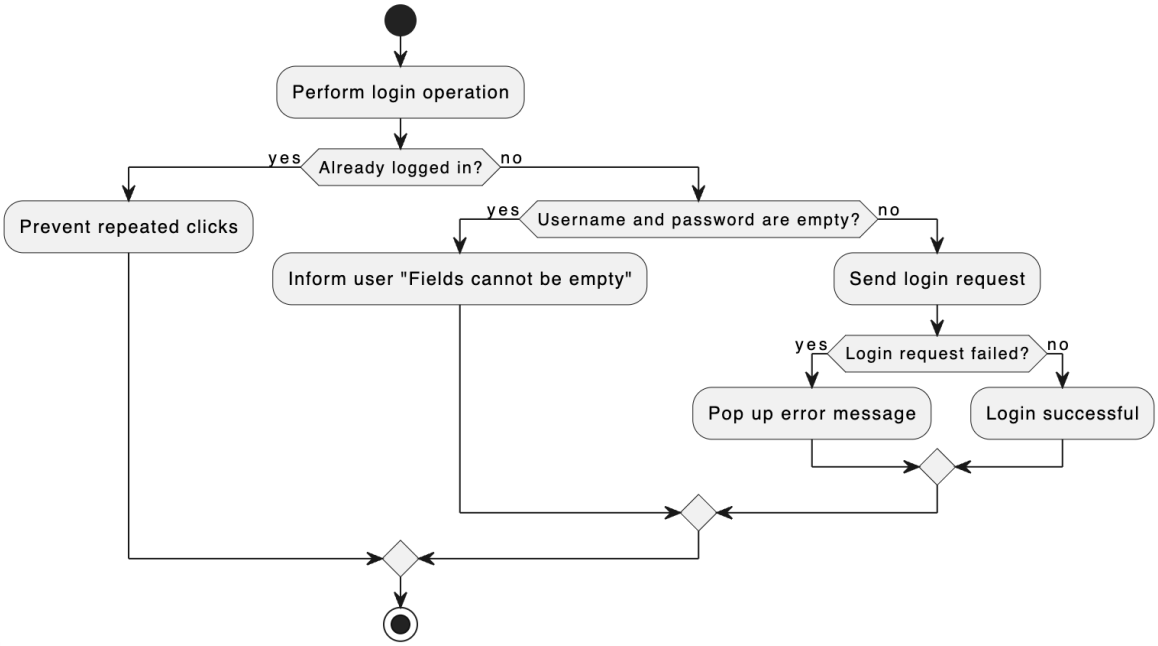


Figure 17: Account Login Activity Diagram for Windows Client

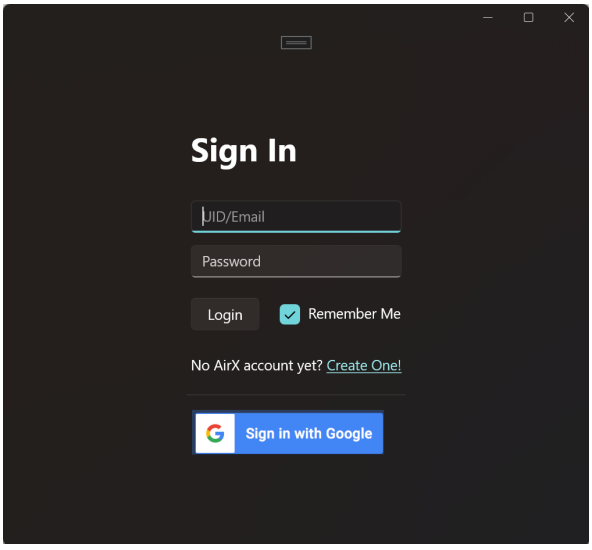


Figure 18: Account Login Interface for Windows Client

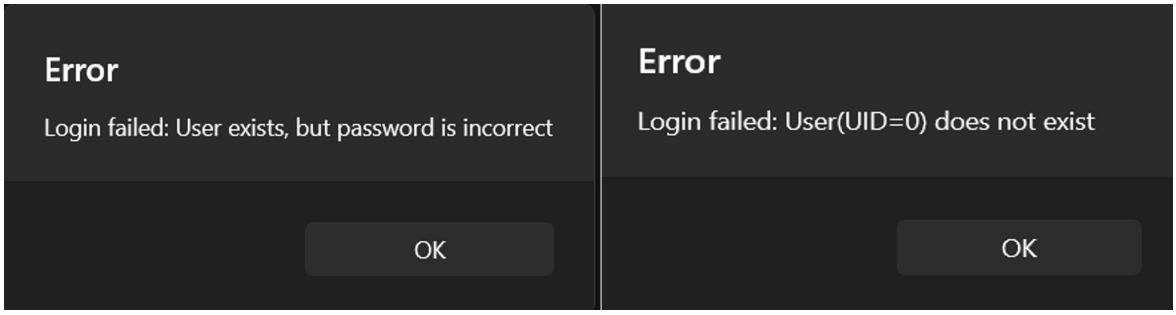


Figure 19: Login Fail Popup for Windows Client

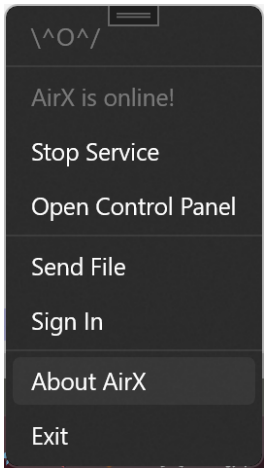


Figure 20: Windows Design: Tray Menu

Figure 17 depicts the sequence of actions based on conditions such as being already logged in, the username or password fields being empty, and the login request failing. This diagram begins with the ”Perform login operation” action. Then, it checks if the user is already logged in. If yes, it performs an action to prevent repeated clicks. If the user is not logged in, it checks if the username and password fields are empty. If they are, it informs the user that these fields cannot be empty. If the fields are not empty, it sends a login request. Depending on whether the login request is successful, it either pops up an error message or proceeds with successful login.

For failed login requests, a popup window is displayed, as shown in figure 19. This window informs the user that the login request has failed and prompts them to try again.

3.3.3 Tray Menu

Placing a tray icon along with a tray menu is a common practice for Windows applications. Figure 20 shows the tray menu that contains the following items:

- **Online Status:** Indicates whether the core library is running.
- **Control Panel:** Opens the control panel window.
- **Send File:** Opens the file picker dialog and sends the selected file to selected peer.
- **Sign In / Sign Out:** Opens the sign in dialog or signs out the current user.
- **About AirX and Exit:** Opens the about dialog or exits the application.

3.3.4 Control Panel

Figure 21 provides a comprehensive look at the window control panel. This panel plays a pivotal role in shaping the user interaction with the system. The components of the control panel include a variety of features that enrich the user experience.

The first aspect to note is the 'Currently logging user' element. This feature is an indicator of the user identity, adding an extra layer of user interface personalization. Though the system in its current state only shows the user's identity, the potential for expansion is vast. Future additions could include options for signing out, changing the user profile, and other relevant user-specific functions that can enhance the user's interaction with the system.

Secondly, the 'Dashboard' feature is a real-time display of the client's status. It provides a snapshot of the number of devices discovered and can potentially contain other pertinent details that will provide the user with a thorough overview of the system at a glance.

Thirdly, the 'Preferences' element is a nod to user customization. It empowers users to modify the client settings according to their liking or requirement. For instance, it allows for changes to the group identifier, thus enabling the user to better organize the system.

Lastly, the 'Sent and received files' feature is an organizer of all files that have been exchanged through the system. It provides a clear log of file transfers, thus helping users track their file sharing activities.

3.3.5 Mica Effect

As of figure 22, Mica, an opaque, and dynamic material, is designed to merge theme and desktop wallpaper components to form a visually pleasing background for persistent windows such as apps and settings.

Incorporating Mica into an application can significantly improve the user experience. It creates a visual hierarchy that draws the user's attention to the window in focus, thereby enhancing productivity. Mica is designed with optimal app performance in mind, limiting the sampling of the desktop wallpaper to a single instance for visualization creation.

The comparison in Figure 22 effectively illustrates the distinction between Mica effect when it is switched on and when it is turned off. This comparison further underscores the impact Mica can have on the overall aesthetic of the user interface.

3.3.6 Blocklist

The Blocklist feature in the system provides a layer of user protection by blocking unwanted file and text transmissions from specified devices. It is essentially a curated list of devices that the user has deemed unnecessary or potentially harmful. This list, which is completely managed by the client, is exclusive to the system and is not an integrated part of the core library. The file receiving dialog, depicted in Figure 24 (Left), demonstrates the block button in action. This button enables users to conveniently add intrusive or unwanted devices to the blocklist.

3.3.7 Desktop Design Prototypes

Figure 23 presents the design prototypes for both file and text receiving for desktop clients. The goal of these prototypes is to provide a clear and intuitive user interface, optimizing the user experience and ensuring seamless navigation through the client. These designs, which are already brought to life

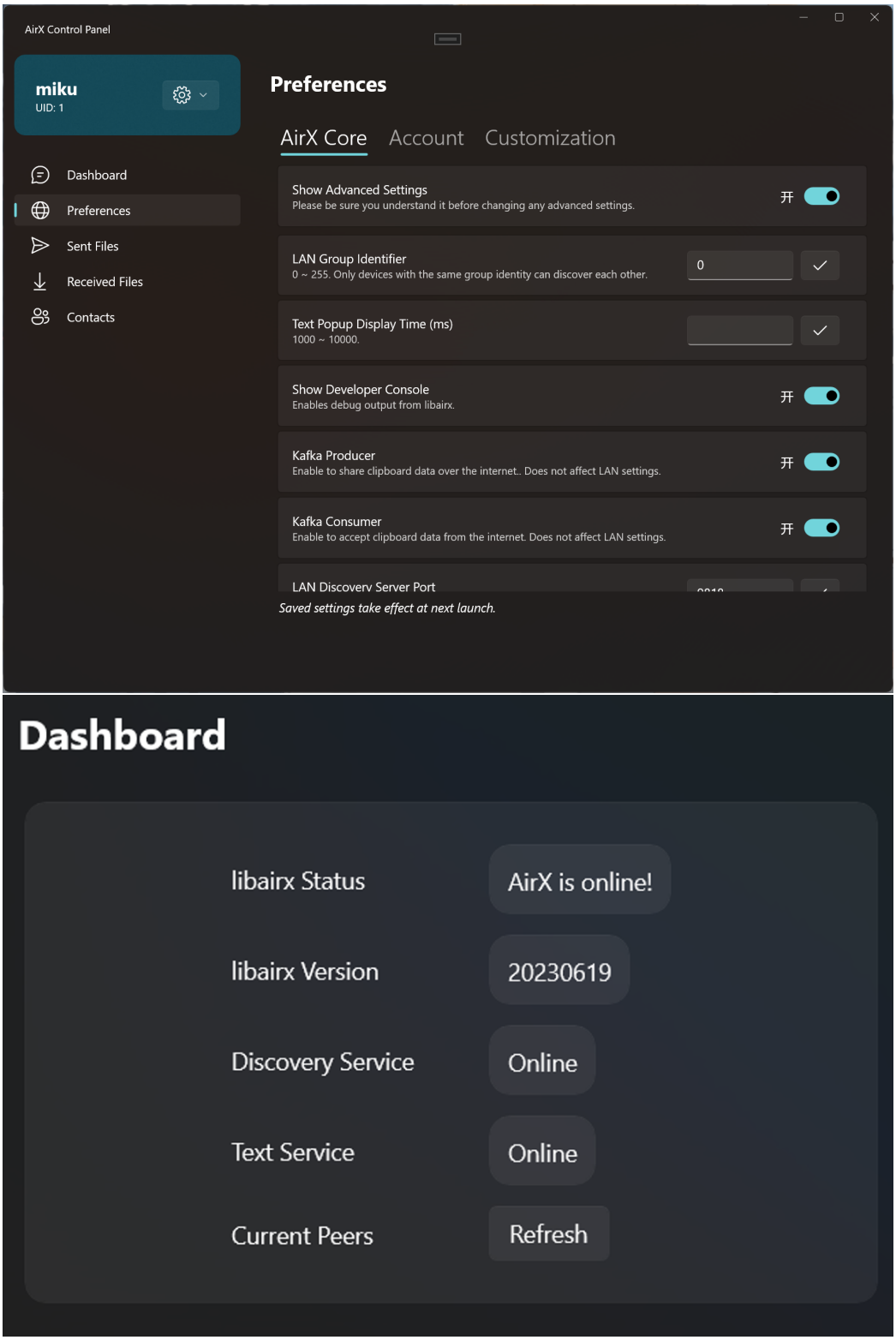


Figure 21: Windows Design: Control Panel with Mica Effect

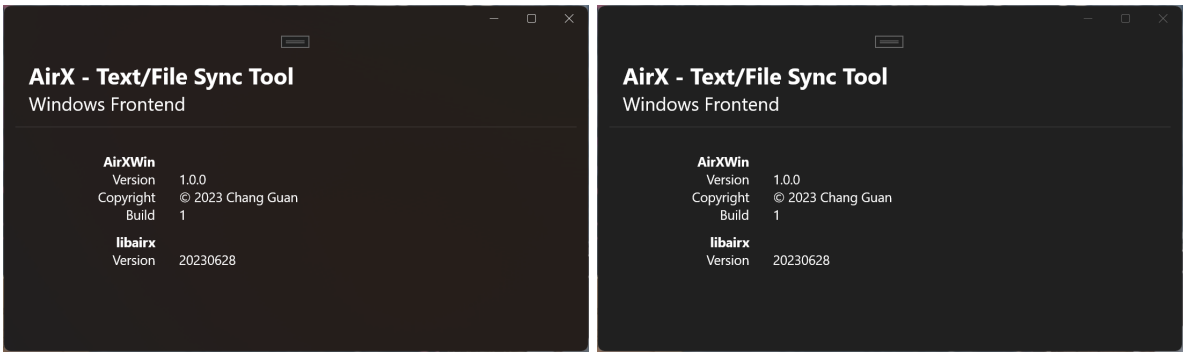


Figure 22: Mica Effect: On (Left) and Off (Right)

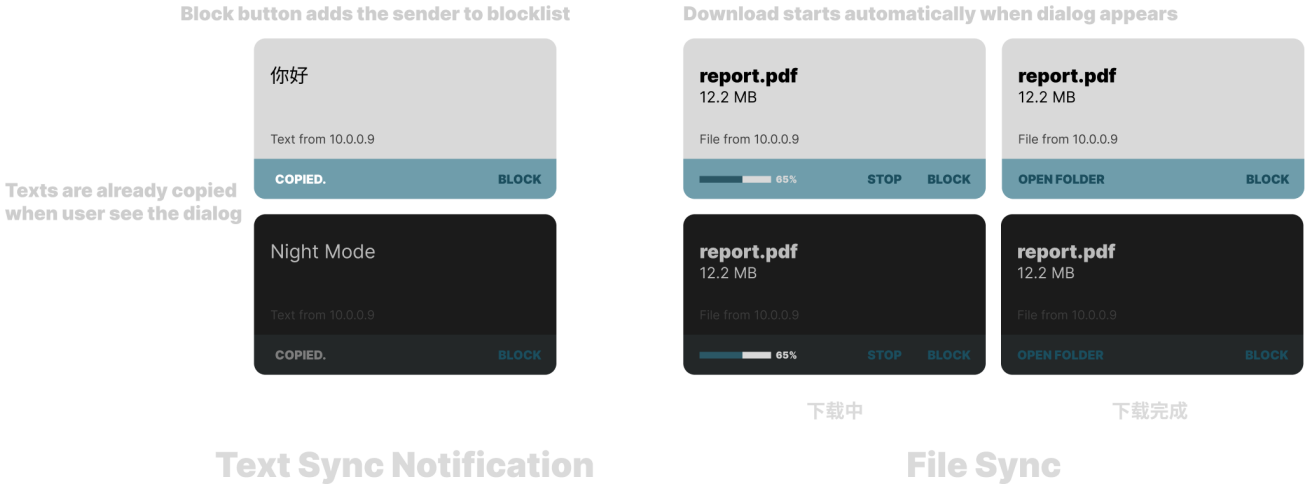


Figure 23: Desktop Design Prototypes: File and Text Received

in the macOS and Windows desktop clients as shown in Figure 24, serve as the foundation for the development of other platform-specific clients. The cross-platform implementation of these prototypes ensures a consistent user experience, regardless of the operating system used.

3.3.8 Send File

Figure 25 illustrates the sequence of actions for sending a file. In this sequence, the user first selects a file. If no file is selected, the process ends immediately. If a file is selected, the system fetches the number of online peers. If no peers are online, a prompt is displayed. However, if there are online peers, a window for Peer selection pops up. The user then selects a Peer and initiates the file sending process. For each file to be sent, a SendFile object is created.

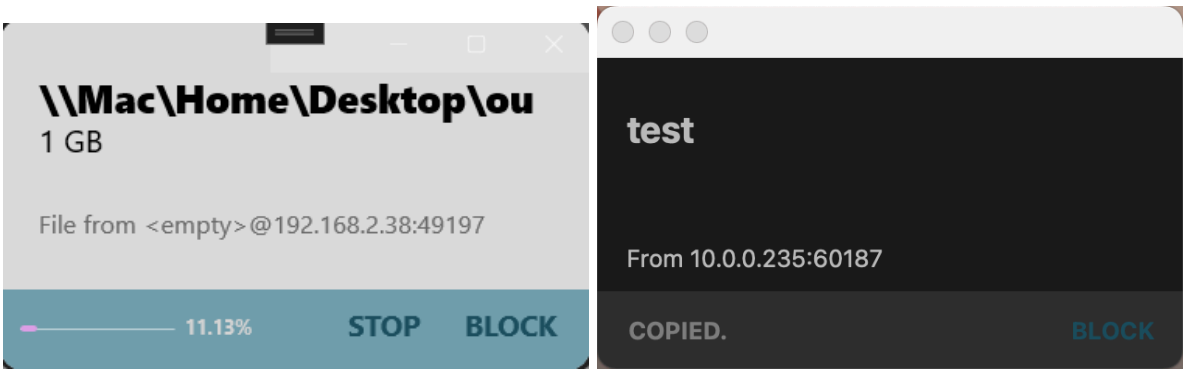


Figure 24: Windows (left, light mode) and macOS (right, dark mode) Design Implementation

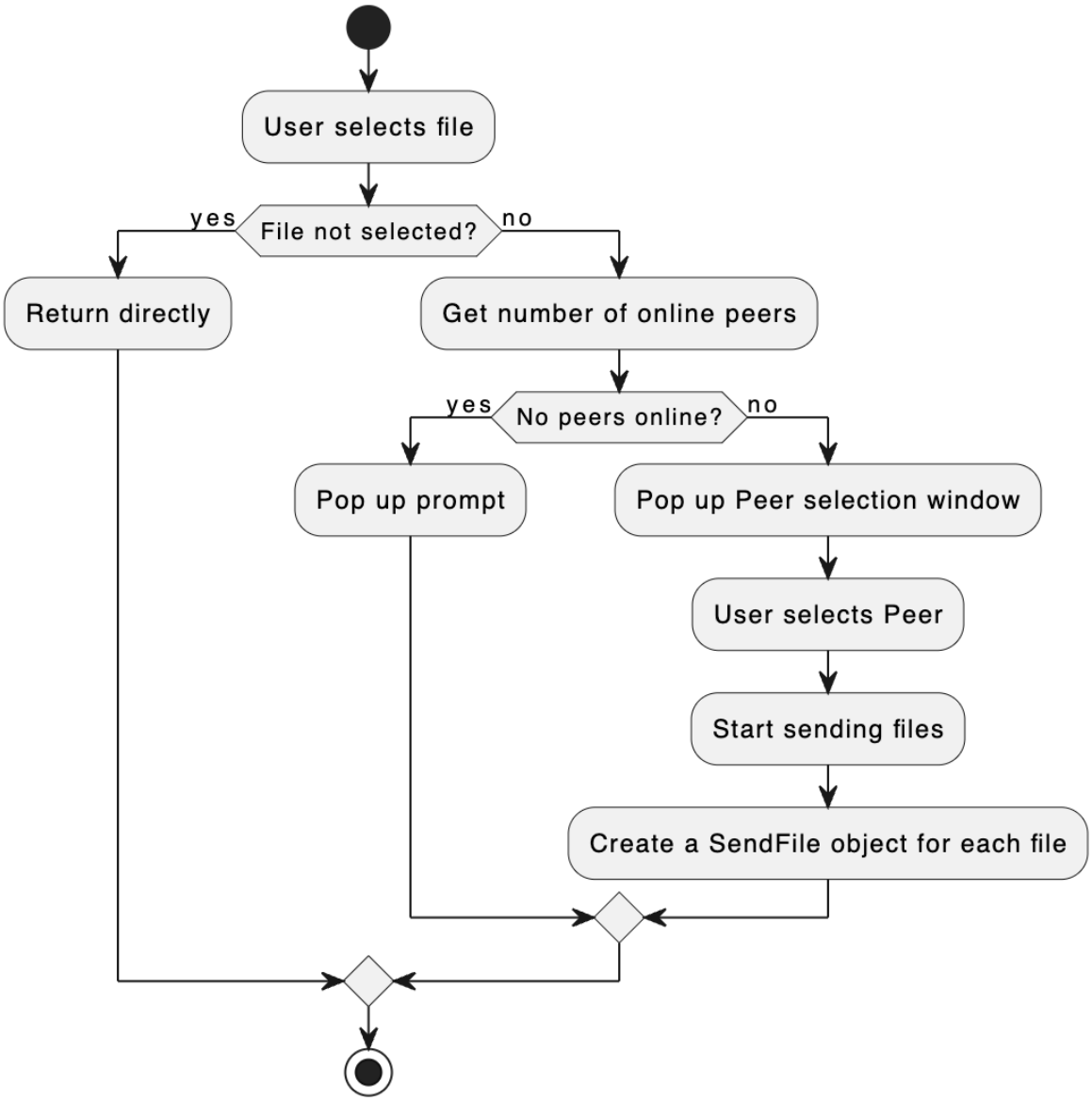


Figure 25: Send File Activity Diagram for Windows Client

3.3.9 Receive File

Figure 26 shows the sequence of actions for receiving a file. In this diagram, the process is triggered by an incoming file transfer. The user is then given a decision to accept or reject the file. If accepted, the user interface prompts for acceptance, preparations are made to receive the file, and the sender is notified of acceptance. The system then determines the target folder (defaulting to 'Downloads/AirXFiles'), obtaining the 'Download' directory URL and appending 'AirXFiles' to it. If the target directory and file exist, they are opened directly; otherwise, they are created and then opened. Finally, a ReceiveFile object is constructed and the file is marked as ready to be received. If the file transfer is rejected, the sender is notified accordingly.

3.4 macOS Client

macOS Client for AirX has essentially the same functionality and behaviour as the Windows Client. Therefore some subsections that introduced before will be omitted due to redundancy.

3.4.1 Design Aesthetics and Approach

With regards to the macOS component ofAirX, we aim to craft an application that perfectly aligns with the native macOS design language and caters to the distinctive operational preferences of the macOS user demographic. Anyone who has had the opportunity to interact with a macOS system will attest to its streamlined, user-friendly interface that facilitates an intuitive user experience. Since the majority of macOS users gravitate towards creative and professional work, it becomes paramount to avoid any flashy or distracting elements in the interface.

Thus, to fully comply with the macOS design language, it is crucial for AirX to integrate seamlessly with the native design elements of the system, thereby ensuring a clean and accessible user interface. Additionally, the software's design blueprint takes into account its potential future scalability: while it is primarily intended for use on macOS, it could feasibly be extended to other Apple operating systems such as iOS, WatchOS, and beyond. Given these considerations, we resolved to adopt SwiftUI as the primary software development language for the macOS client.

3.4.2 Notable Attributes of Swift and SwiftUI

Within the realm of Apple's development milieu, there is a selection of different languages available to developers. Prior to the emergence of the Swift language, the predominant medium for Apple client development was Objective-C. Presently, however, Swift and Objective-C stand as two principal languages, each with its own merits in the development of native Apple software.

Swift, as a state-of-the-art programming language rolled out by Apple in recent years, was conceived with the aim of catering to Apple users' predilection for a uniform and harmonious user experience across various Apple products. To this end, Apple offers the "Human Interface Guidelines", which stipulate that developers must ensure that the visual design of applications meets the specific criteria outlined in the guidelines. These guidelines are instrumental in determining an application's eligibility for release. Swift facilitates the creation of apps that align with Apple's design philosophy, thus enabling developers to easily satisfy these requirements.

In addition to aesthetics, Swift significantly alleviates the developmental learning curve for developers. Swift's intuitive syntax enables rapid mastery of Apple platform development. At the 2019 Apple Worldwide Developers Conference (WWDC), Apple introduced SwiftUI, a declarative framework

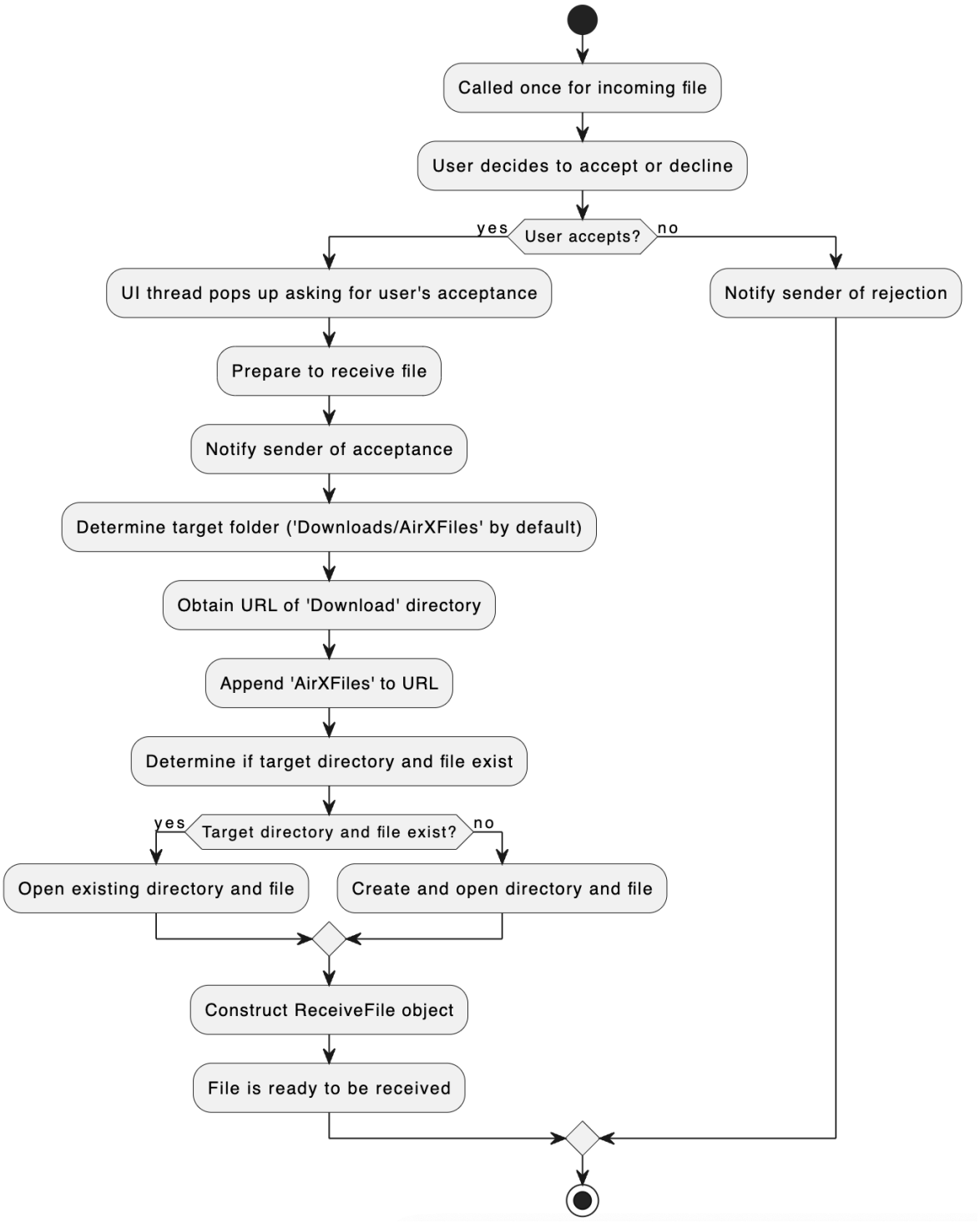


Figure 26: Receive File Activity Diagram for Windows Client

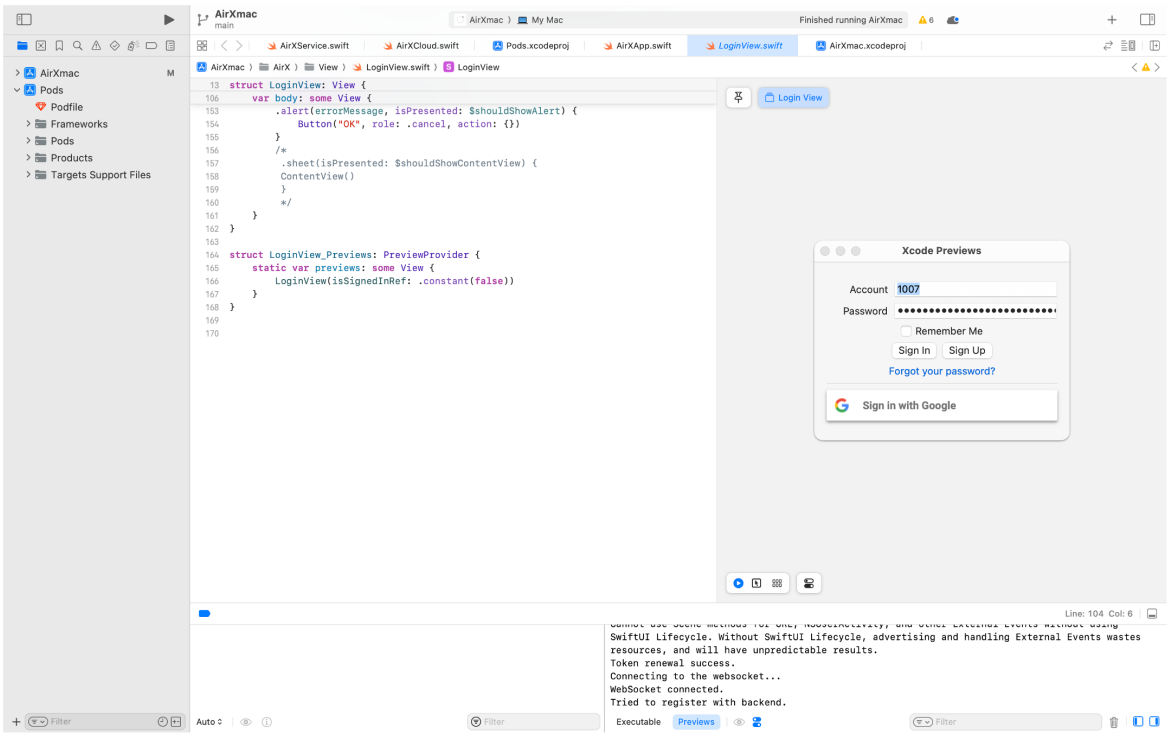


Figure 27: macOS Design: Development Environment

built on Swift. SwiftUI empowers developers to create software for the full suite of Apple platforms - WatchOS, tvOS, macOS, and iOS. This innovation effectively unified Apple's development ecosystem, a task previously segmented across different toolkits such as AppKit for macOS and UIKit for iOS.

As a native development language, SwiftUI facilitates the creation of applications that align more closely with Apple's aesthetic vision, while simultaneously enhancing developer productivity and reducing learning curve costs.

Furthermore, SwiftUI assumes control over many variables, eliminating the need for developers to manually arrange each user interface element, as was necessary with frameworks like UIKit. UIKit and similar frameworks utilized an imperative programming approach, requiring extensive computational overhead, even for minor details such as line breaks and sentence termination in a line of text. If a user switched devices or altered their screen resolution, the application's adaptive capabilities could be compromised. SwiftUI heralded the era of declarative programming, where the system autonomously renders based on screen size, orientation, and other factors, dramatically boosting development efficiency.

SwiftUI also boasts the feature of chain calling. Developers can continuously invoke functions in a chain-like manner, adding a range of custom attributes. This streamlines the coding process, circumvents repetitive tasks, and allows for a meticulous arrangement of page element details.

3.4.3 Development Environment Overview

This project employs XCode as the integrated development environment for coding in Swift. As Apple's official IDE, XCode enables the creation of applications for all Apple devices. XCode provides an image-based representation of a program, allowing developers to preview the actual UI effects without the need to compile and run the program - a highly convenient feature. This circumvents the traditional iterative process of compiling, running, waiting, and finally viewing the resultant output.

Figure 27 illustrates the development environment of XCode. As can be seen from the image, within

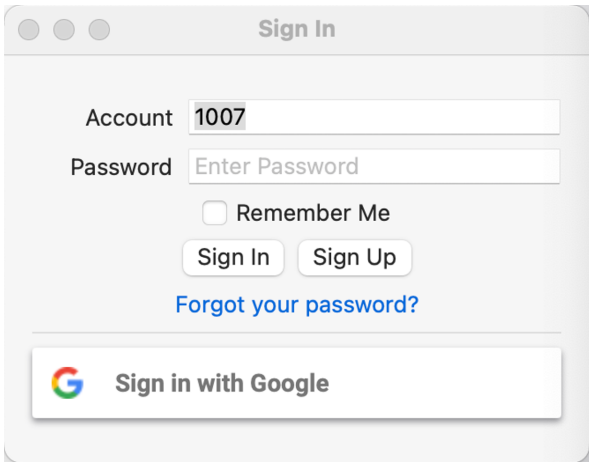


Figure 28: macOS Design: Login View

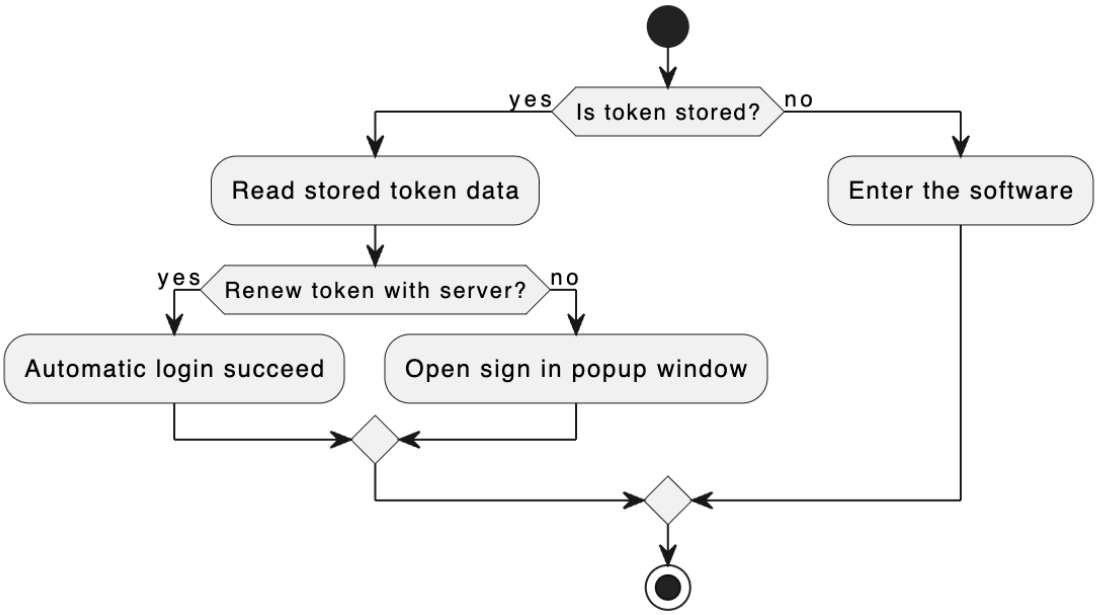


Figure 29: macOS Design: Activitiy Diagram of Auto Login

XCode, the addition of a preview function allows for real-time results to be displayed. This feature is not available in Windows-based development. Compared to the Windows environment, where the software's actual effects can only be checked after each compilation and launch, XCode's direct preview feature significantly saves developers a considerable amount of time.

3.4.4 Login View

During the software's startup process, the initial interface users see is a login page, shown in figure 28, equipped with Account and Password fields for users to log in using their credentials. Additionally, users can opt to remember their account details by selecting the "Remember Me" checkbox. If a user forgets their password, they can choose to reset it via the "Forgot Your Password" option on the login page.

Furthermore, when a user does not have an account, they can register via the "Sign Up" button. Even if a user does not have an AirX account, they can opt to log in using an existing Google account.

Figure 29 illustrates the activity diagram of the auto-login process of macOS client.

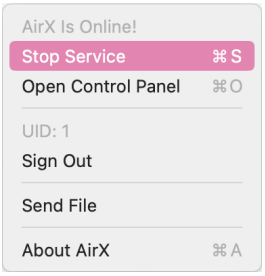


Figure 30: macOS Design: Tray Menu

3.4.5 Tray Menu

As previously mentioned, AirX's goal is to align with the habits of macOS users and to match Apple's design aesthetics. Therefore, once a user successfully logs into AirX, the software does not directly display the interface but minimizes automatically, showing a concise icon on the upper right Menu Bar. This is in line with other software that do not rely on a Control Panel; specific menu options are only displayed when the user clicks on the icon. As illustrated in the screenshot, the menu allows users to start or stop services, open the Control Panel. The user's logged-in UID is also displayed, and users can select "Sign Out" directly from the menu.

The button to send files is also built into the menu, which will be covered later. Users can see basic information about the software, including the front-end version and library version of the mac client, author information, and more through "About AirX". Users can use "Exit" to log out of the AirX service. When a user logs out, the software will automatically shut down the service and exit.

Figure 30 presents the tray menu which encompasses the following elements:

- **Online Status:** This signifies whether the core library is currently active and operational.
- **Open Control Panel:** This option facilitates the opening of the control panel window.
- **Sign In / Sign Out:** This selection triggers the opening of the sign-in dialog or enables the current user to sign out.
- **Send File:** This functionality initiates the file picker dialog and permits the transmission of the selected file to a chosen peer.
- **About AirX and Exit:** This choice prompts the display of the about dialog or facilitates the closing of the application.

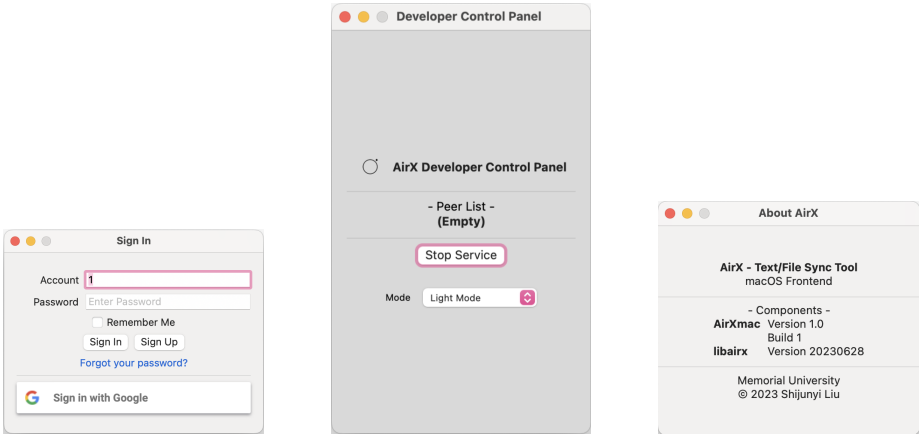
3.4.6 Continue With Google

Figure 31 (a) presents the sign-in page, permitting users to log in via their AirX account or proceed with their Google account. For logins using Google, the client employs OAuth 2.0 to authenticate the user's identity and acquire their email address.

3.4.7 Control Panel

Figure 31 (b) illustrates the control panel window. The window currently supports listing of online peers, the ability to start or stop the core library, and the option to switch between light and dark modes.

Users can choose to open the Control Panel from the aforementioned menu, as shown in the figure.



(a) Sign In Dialog (b) Control Panel (c) About Dialog

Figure 31: macOS Design: Sign In and Control Panel

The Control Panel displays the AirX icon and the label "AirX Developer Control Panel". Simultaneously, below, it shows a "Peer List" of users who have opened the software service concurrently with this machine within the current LAN. If only this machine is running the service in the current LAN, it shows "Empty". Users can choose to stop the service from this interface and switch the software's color mode. The currently built-in modes are "Light Mode" and "Dark Mode".

3.4.8 About Dialog

Figure 31 (c) demonstrates the about dialog.

3.4.9 SwiftUI: Code as Data

SwiftUI is a declarative framework designed for crafting user interfaces across Apple platforms. Its philosophy closely mirrors that of React, allowing developers to conceptualize their code as data [6, 10].

Consider a basic view exhibiting a piece of text and an input box: **As the user provides input, the text automatically updates.** In the case of SwiftUI, the code, as demonstrated in listing 1, is concise, straightforward, and easily comprehensible. Developers can anticipate the outcome without needing to execute the code.

Listing 1: SwiftUI Example for Code as Data

```
1 struct ContentView: View {
2     @State var name: String = "ENGI-981B"
3
4     var body: some View {
5         // Code as data
6         Text("Hello, \(name)!")
7         TextField("Enter Your Name: ", text: $name)
8     }
9 }
```

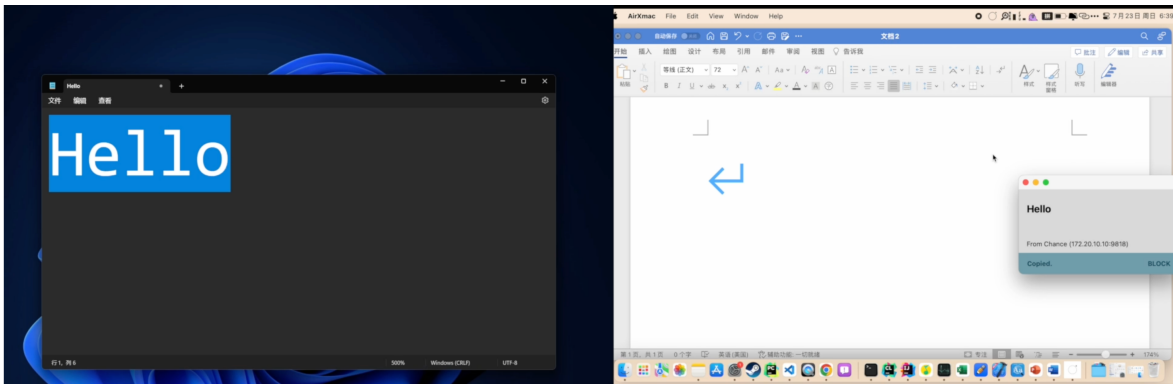


Figure 32: macOS and Windows Text Transfer Demonstration

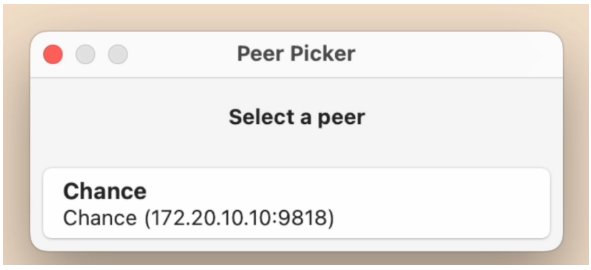


Figure 33: macOS Select Peer Dialog

3.4.10 Text Transfer Demostation

Figure 32 shows a text transfer demonstration. The left side is the Windows client, and the right side is the macOS client. Both have started the AirX service. When we input and copy text under any circumstances on the Win client (including but not limited to Edge browser, Chrome browser, Word, Txt notepad, etc.), users in the current LAN can receive a pop-up reminder. The pop-up will display the content of the transferred text and the sender's information, and the transferred text will be automatically copied.

3.4.11 File Transfer Demostation

File transfer is one of the most important features of AirX. Due to AirX adhering to the design style of macOS, users need to click on the Menu Bar in the upper right corner, then click on file transfer to directly carry out file transfer within the LAN. We have avoided the redundant Control Panel to enhance the productivity experience for macOS users.

When the user clicks the file transfer button, the software will ask the user to first select the file to be transferred. After the user selects the file, the system will pop up a Peer Picker List, illustrated by figure 33, showing the users and their IP addresses who have started the service in the current LAN.

Once the recipient accepts the file, similar to text transmission, the recipient will receive a pop-up, informing the user of the received file name, file size, sender's name, and IP address. The pop-up will also display the file's receiving progress in real time. Users can choose to click STOP to stop receiving. As with text transfer, users can choose to click BLOCK to block the sender to prevent harassment. The example is a test file named "981B_test" with a size of 100MB, sent from another MacBook.

After the file transfer is completed, the STOP button will change to "OPEN FOLDER", which makes it convenient for users to directly open the file's location.

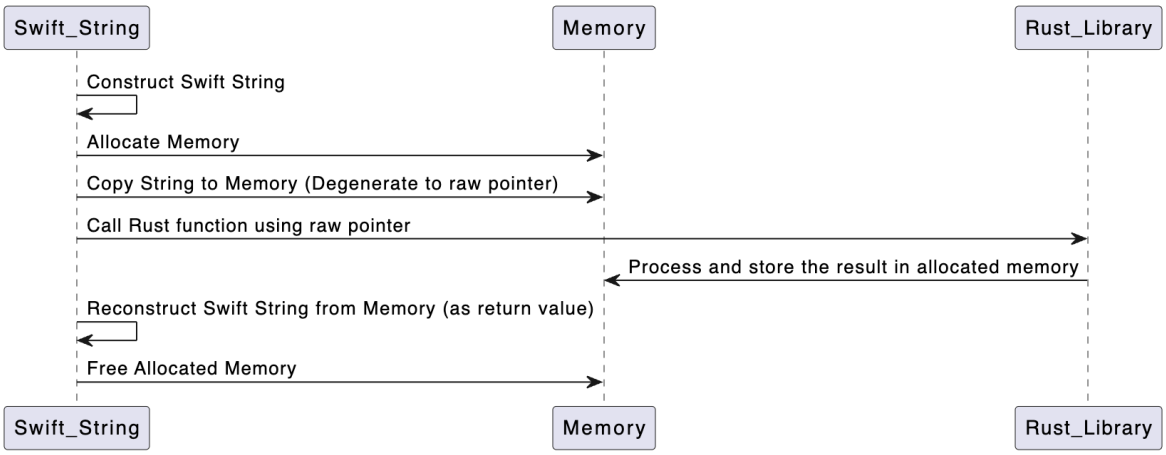


Figure 34: Sequence Diagram of the Process of Calling libairx from Swift

3.4.12 Connection with the Shared Library libairx

The C binding header "AirXBridge.h" allows Swift to invoke libairx, which was written in Rust and suitable for critical tasks such as system-level operations.

Rust provides a set of C-compatible external function interfaces, and Swift can interact with these interfaces through a C bridging header with proper data type conversions.

3.4.13 Implementation of UnsafeString

The author maintained an extension to Swift's builtin String object to facilitate the conversion between raw pointer and String object. As figure 34 shows, the process of calling libairx from Swift is as follows:

- Construct Swift String object,
- Allocate memory for the buffer and copy the String object to the buffer,
- Call the C interface of libairx,
- Construct another Swift String from the buffer as the return value, and
- Release the memory of the buffer.

This is likely to produce memory leak if the memory is not released properly. Therefore, an extension to Swift's builtin String object is implemented to automate the process of memory allocation and release.

3.4.14 HTTP Requests based on Alamofire

Alamofire is a Swift-based HTTP networking library, which is widely used in the iOS and macOS development communities.

Alamofire provides developers with an easy-to-use, yet powerful interface for making network requests, processing responses, and handling networking tasks such as file downloads and uploads. This library excels in its simplicity and efficiency, and it has proven to be an invaluable tool for developers seeking to interact with HTTP APIs, whether to fetch data, post data, or handle JavaScript Object Notation (JSON) responses.

In AirX, we utilize Alamofire for handling all network communication. Whether it's user authentication or session management, Alamofire provides a robust and reliable solution. Its swift-based nature allows for seamless integration with existing codebase, enhancing the overall quality and performance of AirX.

The library's built-in features for error handling, response validation, and automatic JSON decoding significantly reduce the complexity of network-related code and improve its maintainability.

3.4.15 WebSocket Connection based on Starscream

Starscream is a powerful WebSocket client library for iOS and macOS, written in Swift. WebSockets provide a full-duplex communication channel that operates through a single socket over the web, which is extremely beneficial in scenarios where real-time data transfer is essential.

In AirX, we use Starscream for managing WebSocket connections due to its ease of use, high reliability, and extensive feature set. The library supports both WebSocket (ws://) and Secure WebSocket (wss://) connections and offers a simple, delegate-based API which allows us to handle events such as connection opening, closing, and receiving messages.

One of the main uses of Starscream in AirX is to maintain a constant and real-time connection between the client and libairx. This is crucial for AirX because it allows for immediate updates and responses, improving the overall user experience. For example, if a user initiates a file transfer, the software needs to be able to instantly notify the receiving user and handle the incoming data in real-time.

Additionally, Starscream also supports protocol upgrading, meaning that the client can start by making an HTTP/HTTPS connection, and then switch to using WebSockets if both ends of the connection support it. This is extremely useful for dealing with network environments where WebSockets might not initially be available.

3.5 Android Client

Currently, Android client is still working in progress and is not ready for release due to time and Android permission issue. However, the user interface based on React Native has already been implemented.

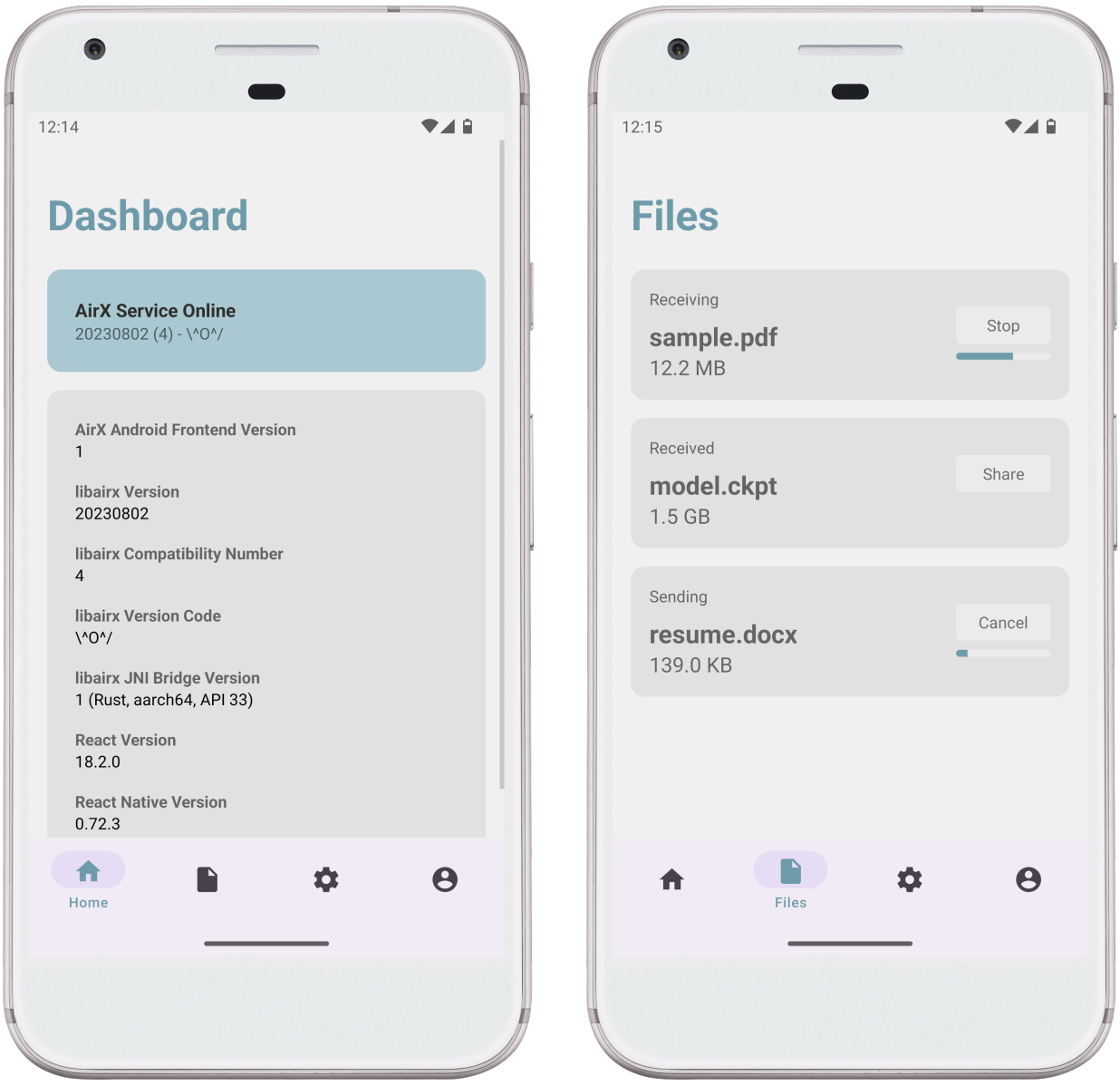
3.5.1 Material 3: A New Design System for Android

Material Design, a design system crafted and maintained by the creative minds at Google, has been updated to its latest iteration, Material 3. This new version facilitates highly personalized, adaptive, and communicative user experiences. Features ranging from dynamic color schemes and improved accessibility options to the groundwork for expansive screen layouts and design tokens all come together to form the innovative landscape of Material 3.

Figure 35 (a), (b) and figure 36 (a), (b) shows the dashboard, files transfer, preferences and my account page of Android client, respectively. The user interface of Android client is designed based on Material 3.

3.5.2 React Native, a Cross-Platform Framework

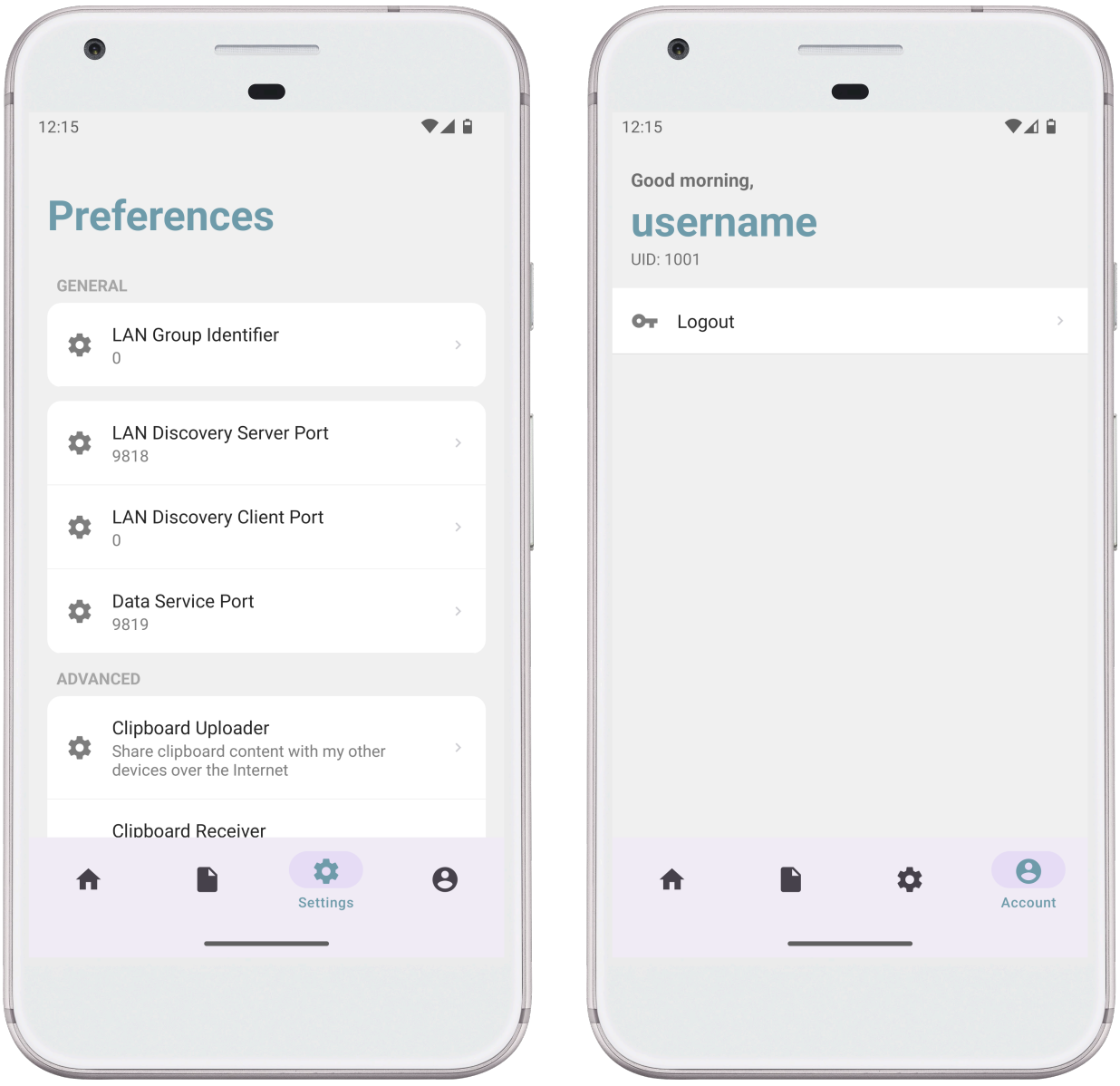
React Native is a popular open-source framework that allows developers to build mobile applications using JavaScript and React. By leveraging the power of React and JavaScript, it offers a platform-



(a) Android Client: Dashboard Page

(b) Android Client: Files Transfer Page

Figure 35: Android Client: Dashboard and Files Transfer Page



(a) Android Client: Preferences Page

(b) Android Client: My Account Page

Figure 36: Android Client: Preferences and My Account Page

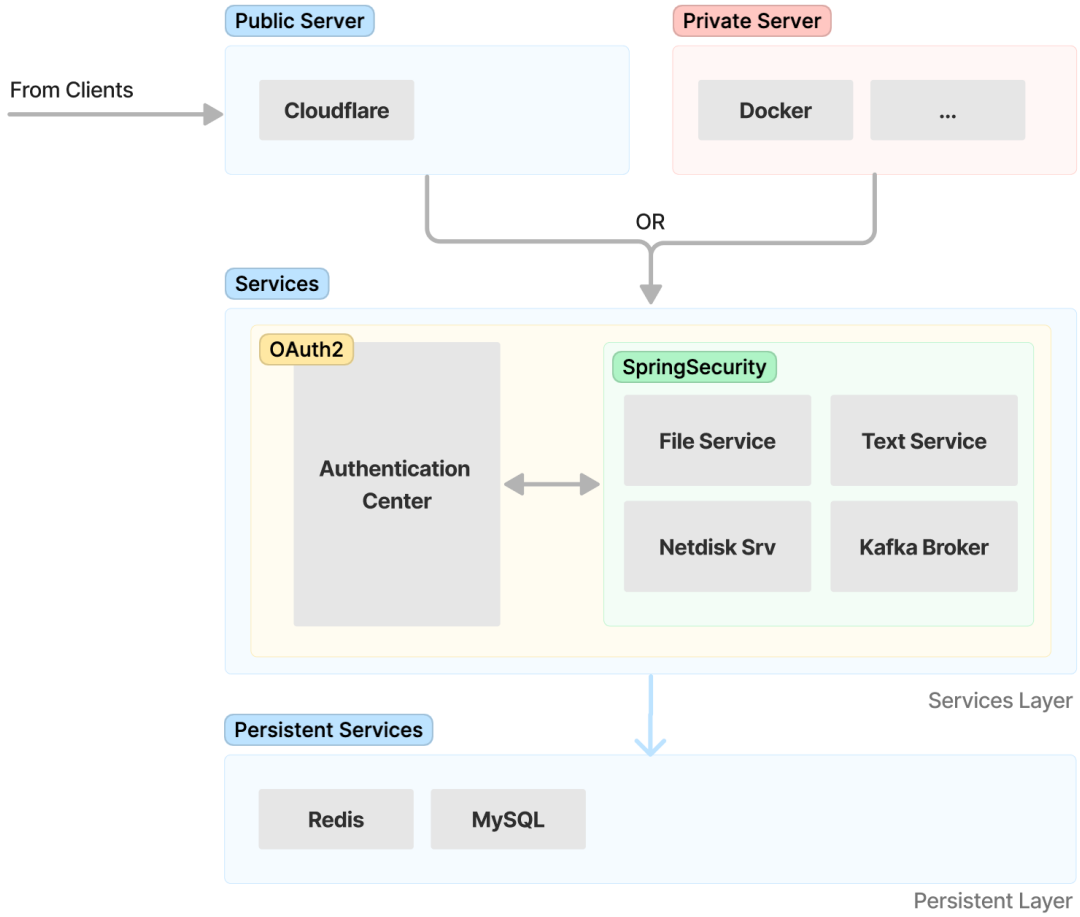


Figure 37: AirX Backend Server: Full Architecture

agnostic solution for developing high-performance applications on both Android and iOS.

React Native works by providing JavaScript interfaces for platform-specific APIs, enabling your application to use native device capabilities beyond what is available to pure web applications. This means that apps built with React Native perform as well, if not better, than natively developed applications, providing users with seamless and high-quality experiences.

Moreover, React Native's component-based structure enables developers to build apps with a more intuitive and streamlined coding process, and its "hot-reloading" feature allows changes to take effect in the app while it's running, which significantly improves the development efficiency.

In the case of the Android client, React Native was chosen for its ability to deliver a robust, high-performing application while significantly reducing the development time and effort involved compared to native development. Furthermore, its vast community support provides a wealth of resources and third-party plugins, ensuring that the development process can quickly adapt to any new requirements or challenges.

3.6 Backend Server

Figure 37 shows the full architecture of the backend server of AirX. The backend server is responsible for managing the data flow between different devices, ensuring efficient communication between devices, and maintaining the overall performance and reliability of the system.

3.7 Authentication Server

Figure 38 illustrates the procedure of enhancing user login security. This can be achieved by initiating a login request from the frontend web page to the backend server, and by validating an encrypted username and password, as depicted in Figure 39. The incorporation of a token system facilitates stateless authentication and continual confirmation of the user's identity during subsequent business interactions. This process amalgamates password hash storage and verification, managing user records in the database, and generating and validating tokens to augment both security and user experience. However, it is crucial to be mindful of choosing and configuring the password hashing algorithm and managing the validity and update mechanism of the token. This attention to detail is needed to maintain the security and dependability of the overall login process [11].

The specific implementation can be divided into the following steps:

3.7.1 Security in MySQL-based Authentication Process

When a user inputs their username and password on the login page of a frontend webpage, the webpage packages the username and password in a login request and sends it to the backend server. The Authorization Services module on the backend server receives the login request, extracts the username and password from the request, and establishes a connection with the MySQL database. Using the provided username, the Authorization Services module queries the MySQL database for the matching user record. The database accepts the query request, searches using the provided username as a condition, finds the corresponding user record in the user table, and retrieves the stored password hash and related information.

The backend server, having received the user records returned by the database, extracts the stored password hashes. The Authorization Service Module performs a hash of the password entered by the user and compares it with the password hash stored in the database. If the password hash values match, the user authentication is successful, verifying the username and password as valid. The Authorization Service Module then returns the authentication result to the frontend webpage [12].

3.7.2 Token-Based Authentication and Secure Transport

Following a successful user authentication in step 1, the Authorization Services module generates a token containing the user's identity information. This token is sent to the frontend web page as a response to a successful login. The token is received and stored locally by the frontend web page for use in subsequent requests.

3.7.3 Token-Based Backend Request Processing

The frontend web page appends the token to subsequent requests as identity credentials, for instance, by adding an Authorization field to the request header or by carrying the token in the request parameters. The business service module in the backend server receives the request, verifies and parses the token. By confirming the signature and integrity of the token, the backend server can authenticate the token and extract the user's identity information from it. The backend server processes and responds based on the user's identity information and the specific business request.

The introduction of the token mechanism in step 2 enables stateless authentication, avoiding the overhead of storing session information on the backend server. The frontend web page can continuously

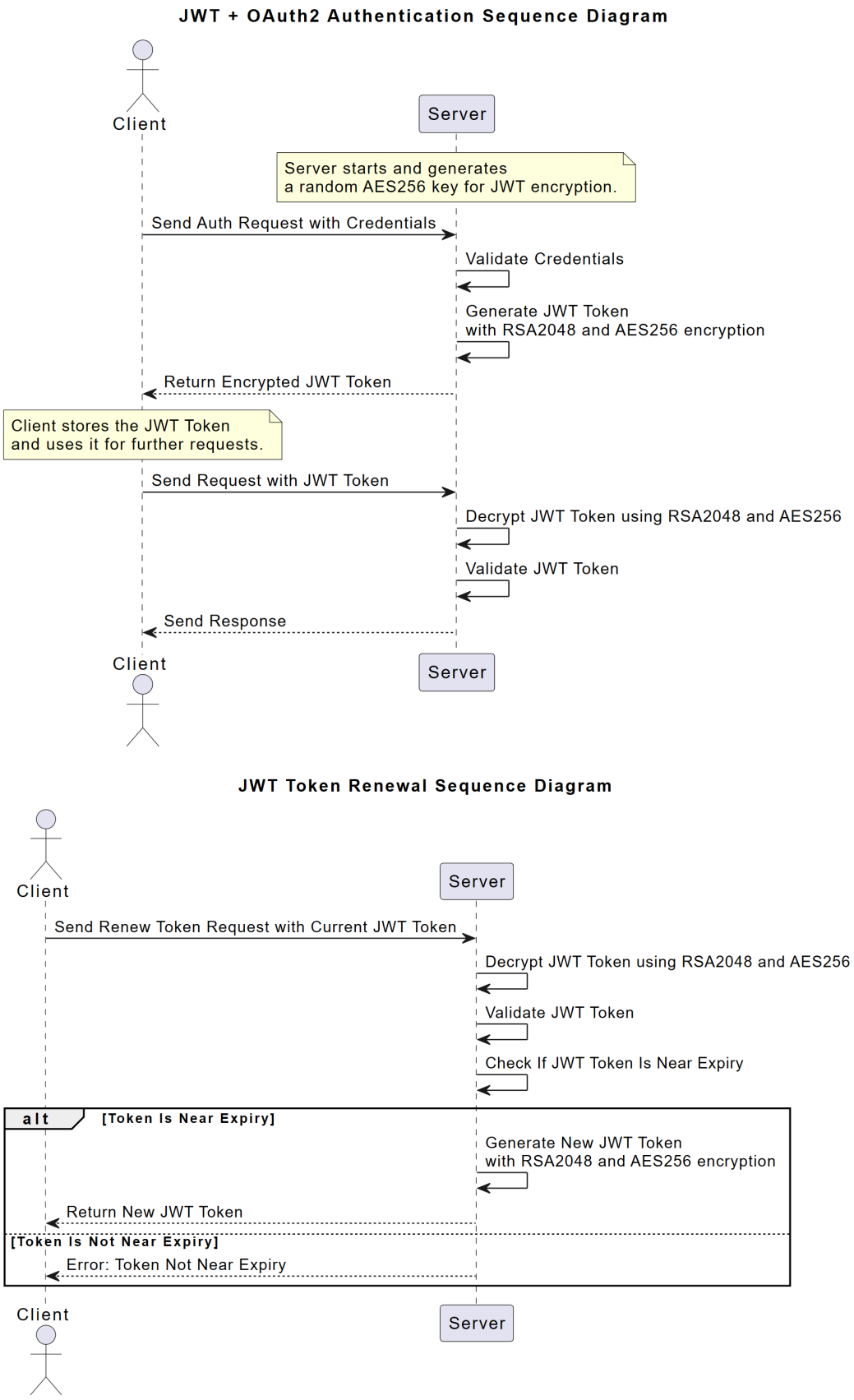


Figure 38: AirX Authentication Server: Authentication and Token Renewal

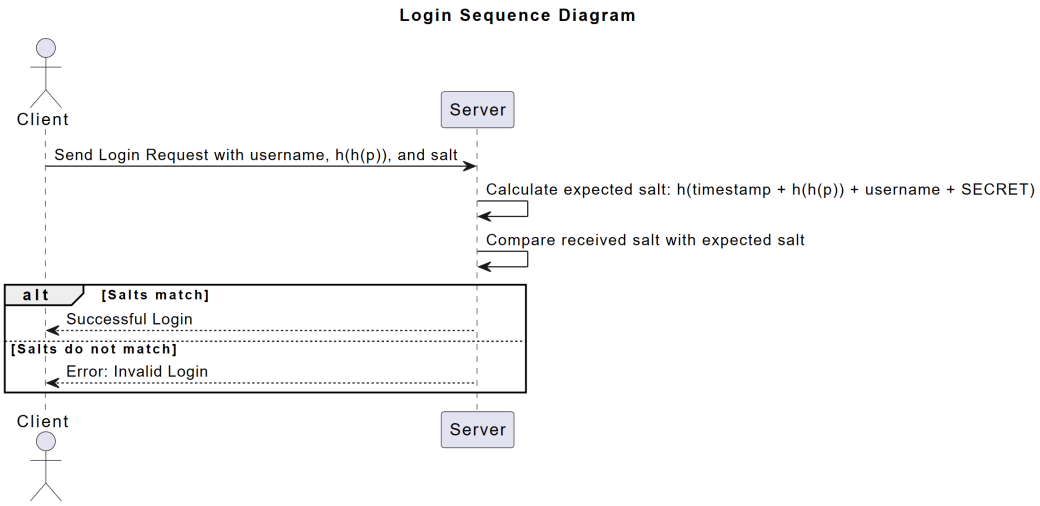


Figure 39: AirX Authentication Server: Login Sequence Diagram

validate the user's identity in subsequent requests by carrying the token, eliminating the need for re-authentication for each request. Concurrently, the authenticity and integrity of the token are ensured by signing and verifying the token, preventing forgery and tampering. This token-based authentication mechanism is particularly useful in distributed systems and cross-domain applications, offering superior security and scalability.

3.8 Authentication Technical Details

3.8.1 Cryptographic Salt

The incorporation of salt in user login authentication is crucial to amplify password security, protect user accounts, and deter replay attacks. By employing a randomly generated salt value combined with the user's password for a hash operation, common password cracking attacks can be efficiently mitigated. Salt technology is a staple in modern authentication systems and, when used alongside other security measures such as choosing the correct hash algorithm and encrypted communications, can offer an enhanced level of password protection and user authentication. Salt techniques increase password complexity by appending additional random data and hashing encryption, preventing an attacker from directly accessing a user's plaintext password. With the combination of the salt value and the password, and its encryption using a hash function, even if an attacker gains a secondary hash value, they cannot restore the user's real password as the salt value is randomly generated.

Specific Process of Salt Generation

- **Generating the salt:** The server side generates a unique salt value for each user during user registration. The salt could be a randomly generated string or could be generated based on specific user information. The salt should be generated randomly and uniquely to ensure that each user possesses a different salt value.
- **Combining salt values with passwords:** During user registration, the salt value is merged with the password inputted by the user to form an extended string. The salt value can be added to the beginning or end of the password, or interlaced with the password.
- **Salted password hash:** The merged string undergoes a hash with SHA-256. The resulting hash operation becomes the password hash stored in the database.

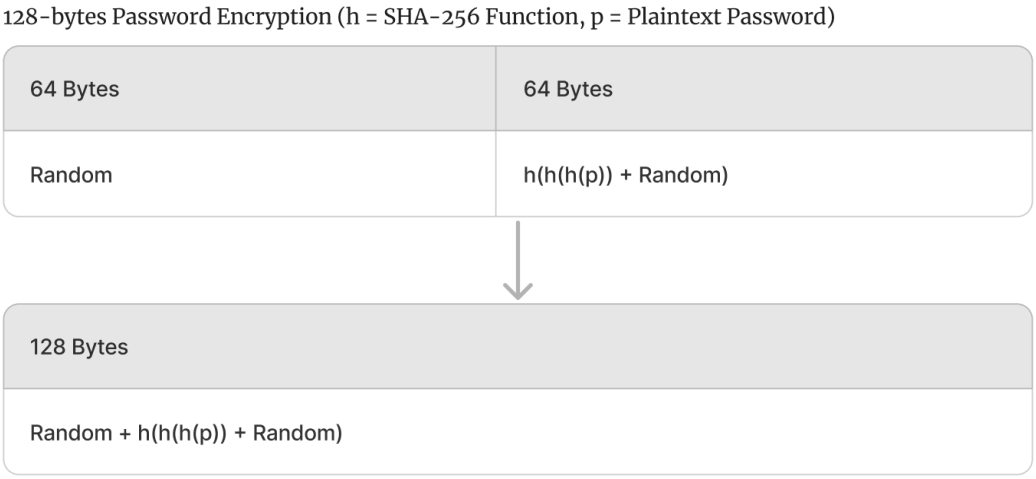


Figure 40: 128-byte Password Encryption: Verification Possible, Decryption Impossible

- **Storing the salt and password hash:** The created salt and password hash are stored together in the user record in the database. To verify a user's identity, the salt value of the corresponding user is retrieved, and the same merge and hash operation is performed on the entered password and salt value. The result is then compared with the stored password hash value in the database.

The Importance of Salt Incorporation

- **Boosting password complexity and security:** Integrating a salt value with a user's password enhances the password's length and intricacy. This increased complexity makes the password more resistant to guessing or brute force attacks. Salt usage protects against prevalent attacks, such as rainbow table attacks. If two users have identical passwords, the merged string and hash value differ due to the unique salt value, further challenging an attacker's password cracking attempts.
- **Mitigating password reuse:** Users frequently use the same password across multiple platforms. Employing salt values ensures that even if users reuse passwords across systems, the resulting hashes will differ due to the unique salt values, mitigating the risks associated with password reuse.
- **Augmenting password confidentiality:** The salt value, stored as supplementary information on the server side, prevents attackers from directly accessing the original user password. Even in the event of a database breach, the salt value provides an extra layer of protection, reducing the likelihood of password compromise.

3.8.2 Password Storage

The employment of random numbers and hash functions is a common practice to secure passwords. Figure 40 and the ensuing description detail the specific implementation steps for password storage:

- **Random number generation:** A random number is generated during the user registration phase. This random number is typically 64 bits long and can be produced by a secure pseudo-random number generator.
- **Hash function nesting:** Applying the formula

$$\text{password} = r \ \& \ h(h(h(s)) \ \& \ r)$$

where "&" signifies the string concatenation operation, "r" denotes the random number, "s" represents the user-entered password, and "h" is the SHA-256 hash function. This formula introduces hash function nesting, meaning the password undergoes multiple rounds of hashing. The objective of nested hashes is to elevate the complexity and security of the password, rendering it harder to crack.

- **Password feature elimination:** Appending a random number before and after the password effectively eliminates the characteristics of the user's password. This prevents attackers from cracking the password using pre-calculated rainbow tables, for instance.
- **Adherence to Kerckhoffs Principle:** The utilization of random numbers and hash functions to bolster password security aligns with the Kerckhoffs Principle. According to this principle, a system's security should rely solely on the confidentiality of the key (random number), and not on the confidentiality of the algorithm. Even if an attacker has knowledge of the password storage algorithm and the used hash function, they must still crack the random numbers to access the actual password.

With the described steps, the password storage formula provides a highly secure method of storing user passwords. It amalgamates techniques such as random number generation, hash function nesting, and password feature elimination, thereby making passwords more resistant to cracking. Furthermore, this approach conforms to the Kerckhoffs principle, placing the system's security on the confidentiality of the key, enhancing the dependability and security of the password storage.

3.8.3 Token Technology

The server returns a token, which contains key information and serves as the user's sole proof of identity for subsequent interactions. Essentially, this token is an AES-256 encrypted JWT with an RSA-2048 digital signature and includes the user's name among other details. The token also features an important field indicating its expiration time, after which it cannot serve as identity proof. Each token possesses a fixed validity of 3600 seconds [13].

Detailed Explanation of Token

- **Token Structure:** The token in focus is a JWT (JSON Web Token), structured into three components: a header, a payload, and a signature.
- **Encryption and Signature:** The JWT employs the AES-256 encryption algorithm for encrypting the payload, and the RSA-2048 digital signature algorithm is used for signing the encrypted JWT. This dual-level application ensures the token's confidentiality and integrity.
- **Payload Information:** The payload part of the token encapsulates information pertinent to the user, such as the username, role, permissions, and so forth. This information is vital during identity verification and can be utilized for authorization and recognition purposes.
- **Expiry Field:** The token also embeds a critical field known as "exp" (expires) field, indicating the token's expiration time. Every token has a pre-set expiration time of 3600 seconds (1 hour). Post this duration, the token becomes invalid for authenticating the user's identity, necessitating the generation of a new authenticated token.

- **Enhanced Security:** JWTs, through their usage of encryption and signature technologies, safeguard the confidentiality and integrity of payload information against tampering and falsification. Reduced Server Load: JWTs, being self-contained, eliminate the need for servers to maintain session state on the backend, thereby reducing server load. JWTs can easily adapt to distributed systems as each server can independently verify and interpret the JWT without sharing session state.

Token-based Innovations and Improvements

- **JWT Limitations:** The revocation of a JWT once it has been issued is challenging. Ordinarily, it must be left to expire, posing potential security risks. To mitigate this, AES encryption is employed. With AES encryption, a 16-bit Initialization Vector (IV) matrix is required. Each time the server is restarted, this IV matrix changes randomly. Consequently, the previous JWT cannot be decrypted, rendering the old JWT ineffective. This procedure effectively serves as a revocation mechanism.

- **AES encryption for tokens:** The content of a JWT is encoded using base64, making it publicly accessible. Any tampering with the JWT content will result in the digital signature not aligning with the original data. The JWT digital signature, created using RSA-2048, acts as a unique stamp. To bolster user privacy and overall system security, AES-256 encryption is implemented. This additional layer ensures that only the server can decrypt and validate the original JWT.

In the area of user authorization and login authentication, integrating AES encryption with Tokens can significantly augment both user privacy and system security. A comprehensive analysis of this method is provided in the following sections.

Given that the payload of the JWT is Base64 encoded and accessible publicly, it implies that the information it contains is readable by anyone. While the digital signature of the JWT assures the data's integrity, it does not inhibit the potential leaking of information. To counteract this issue, AES-256 encryption is introduced to eliminate the risk of JWT content being viewable, by adding an AES-256 encryption layer to the JWT. AES is a symmetric encryption technique that utilizes the same key for both encryption and decryption processes. The server solely possesses this key and is thus the only entity that can decrypt and gain access to the original JWT content.

The execution of AES encryption necessitates a 16-bit initialization vector (IV) matrix. The IV matrix undergoes random alterations whenever the server is rebooted, leading to the inability of the new IV matrix to decrypt the preceding ciphertext. As a result, the previous JWT certificate becomes ineffective post a server reboot, serving as a form of revocation to some extent. Traditional JWTs pose considerable difficulties to be revoked once issued and are often left to naturally expire. By incorporating AES encryption, decryption demands both the server's key and the accurate IV matrix, ensuring that even with an old JWT certificate, decryption and verification cannot be achieved. This indirectly mitigates the issue of non-revocation.

3.9 Data Service: Device Registration

Device registration is a part of text synchronization over the Internet. This process is shown in figure 41: the clients connect to server through WebSocket and send their logging user Unique Identifier (UID) to the server.

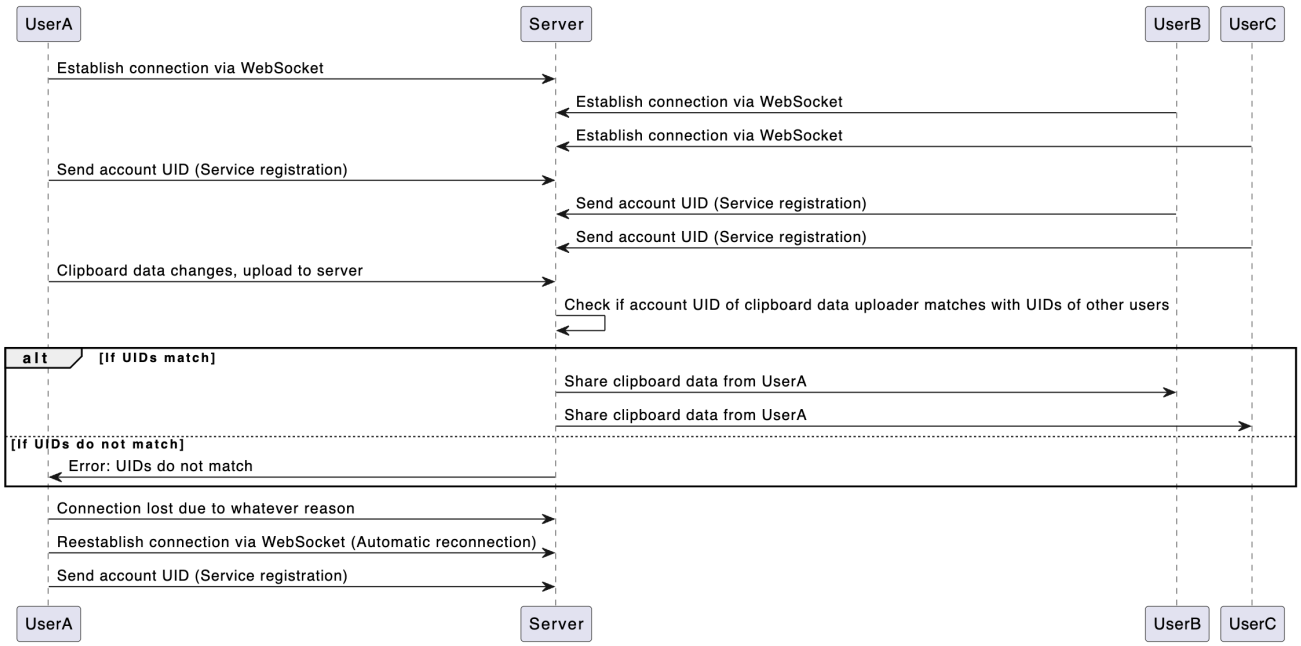


Figure 41: AirX Data Service: Device Registration Sequence Diagram

3.10 Data Service: Data Distribution

Except for device registration, figure 41 also briefly illustrated the data distribution process. This process is demonstrated in detail in figure 42: Once a server receives the clipboard data from a client, the server itself, as a Kafka producer, sends the user's UID and the clipboard data to the fixed topic in the Kafka cluster. All subscribers, including the producer server itself, checks all registered devices and sends the clipboard data to the corresponding device who has the same UID as the clipboard data.

3.11 Cloud Storage Service: Instant File Upload

As previously mentioned in the literature review section, AirX's cloud service has implemented instant file upload mechanism. This process is shown in figure 43: the client sends a file upload request to the server, without the file itself. If the server already has the file, the transmission will be skipped.

3.12 Database Design

The core foundation of any data-driven application is its underlying database design. An efficient and robust database design allows the application to effectively manage, store, and retrieve data, thereby driving the application's performance and scalability. AirX is no exception to this. Its database design, as depicted in Figure 44, plays a crucial role in its overall functioning. The design consists of four primary components:

- **file_store**: The file_store table serves as a direct mapping to the physical files. Each entry corresponds to an individual file and stores crucial metadata related to that file. Metadata could include attributes such as the file name, size, format, creation date, hash value, and more.
- **file**: This table establishes the relationship between the files and the users who own them. It is instrumental in managing user-specific data and enforcing user access controls. For each file, it links the file to the user who owns it, ensuring that users can only access and perform operations on files they own.

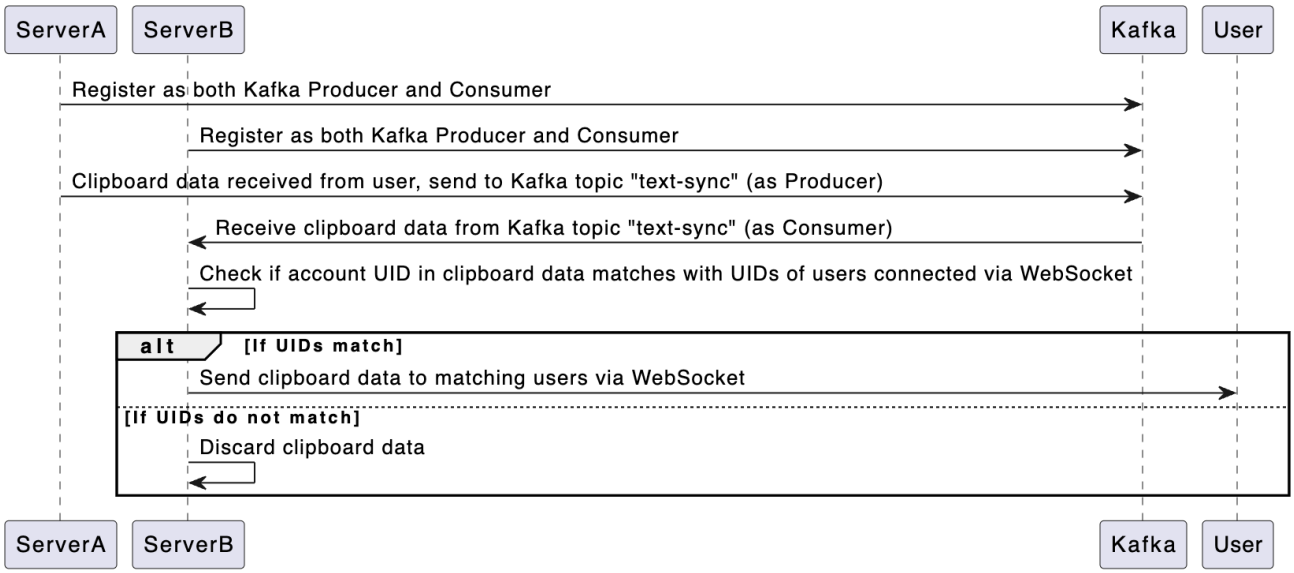


Figure 42: AirX Data Service: Data Distribution Sequence Diagram

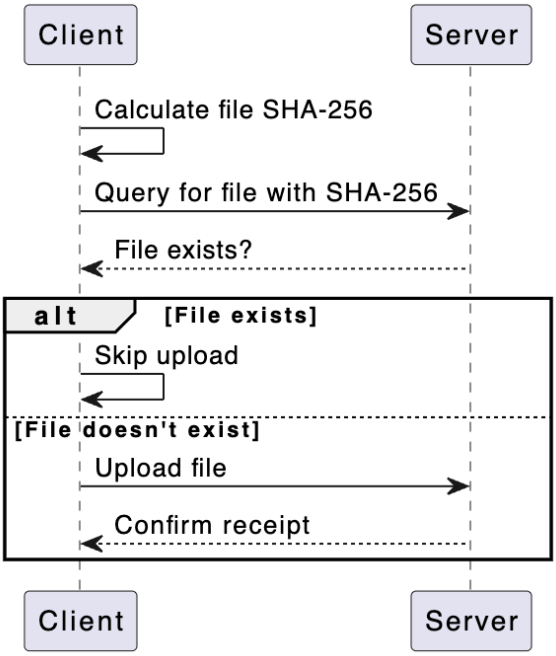


Figure 43: AirX Cloud Storage Service: Instant File Upload Sequence Diagram

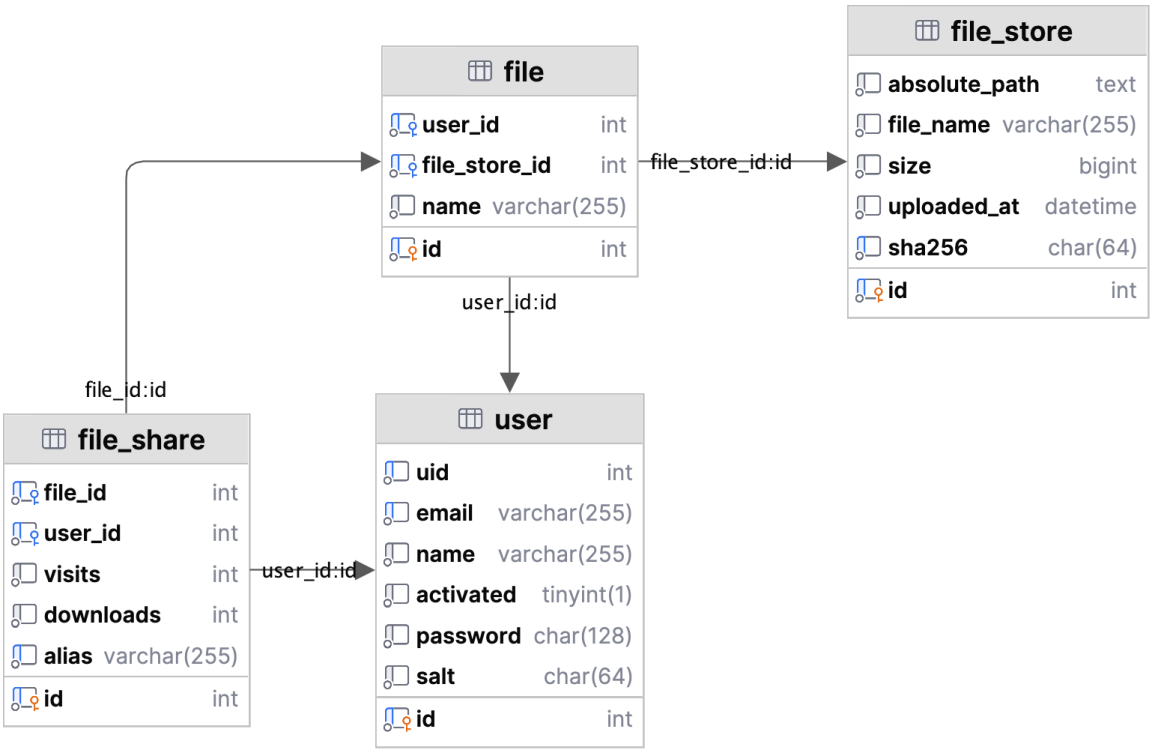


Figure 44: AirX Database Design

- **file_share**: This table is responsible for storing the share links for files. Each entry represents a unique share link, which can be used by other users to access the file. This table plays a vital role in facilitating the sharing of files between users, a core functionality of AirX.
- **user**: This table stores all the necessary information about the users. Information stored includes user details like user UID, 128-bit hashed password, email address, and other pertinent details. It helps manage user authentication and personalization, contributing significantly to the user experience.

Each component serves a distinct role, and together, they create a system that facilitates efficient and secure file sharing. The database design is critical to the successful operation of AirX and forms the basis for many of its key features. Future work on AirX will include updates and improvements to this database design to accommodate new features and ensure AirX continues to operate efficiently and effectively.

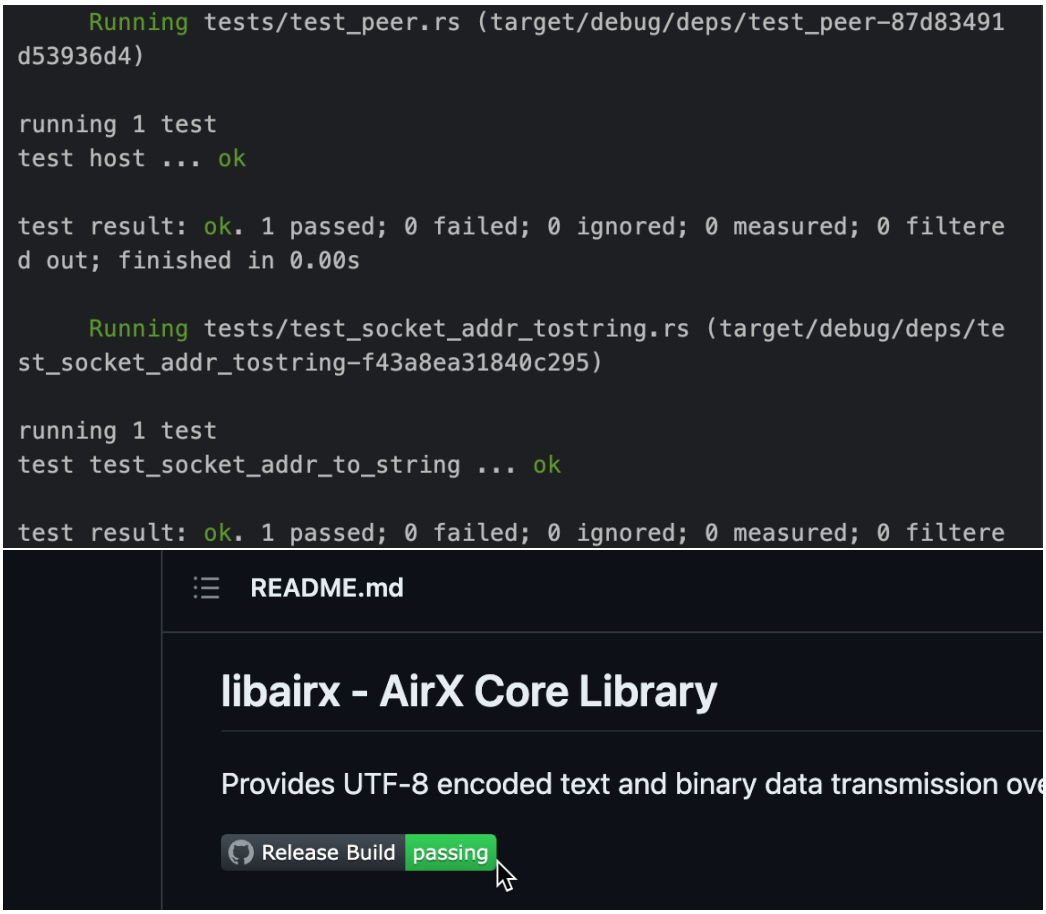


Figure 45: Unit Test for libairx

4 Testing and Verification

AirX has 5 components that need to be tested:

- lixairx,
- backend server,
- Windows client,
- macOS client, and
- Android client.

This section will discuss the testing and verification of each component.

4.1 libairx: Automated Unit Test

The unit test cases for libairx were written along with the implementation of the library. The testcases covered serialization and deserialization of all packets and the correctness of utility functions. Figure 45 shows the result of the unit test and automated test implemented in GitHub Actions.

4.2 Backend: SwaggerUI Testing

The backend server provides a SwaggerUI interface for testing the API of the server. Figure 46 shows the SwaggerUI interface of the backend server.

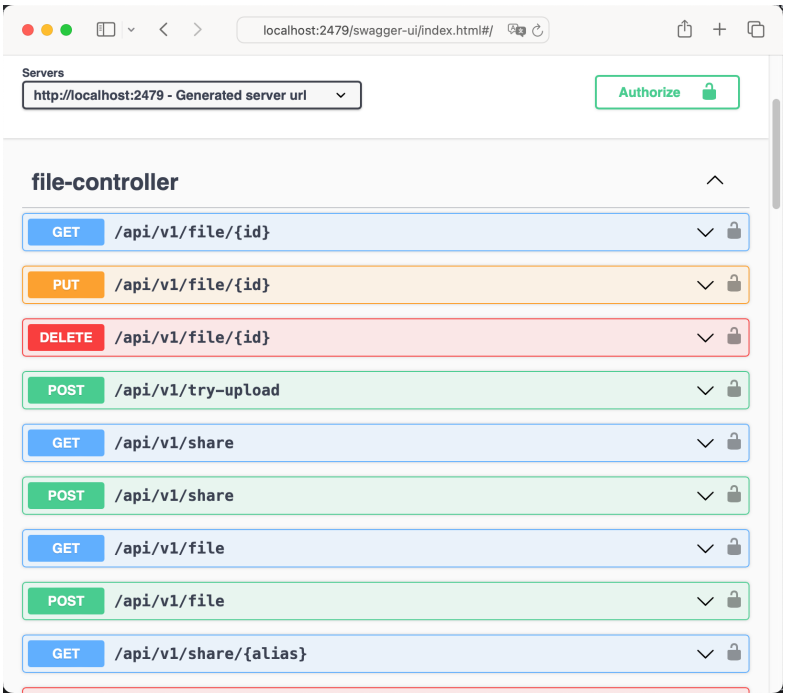


Figure 46: SwaggerUI Application Programming Interface (API) Testing for backend

4.3 Frontend: Black Box Testing

Except for Android client that is currently under development, both macOS and Windows cilent have been fully tested by the team members. The process is based on the black box testing method with the following test cases:

- **Basic Functionality**: The basic functionality of AirX includes clipboard sharing and file sharing. The test cases for basic functionality are:
- **Cold Start Time**: The cold start time of AirX is the time from the moment the user clicks on the application icon to the moment the application is fully loaded. We require the cold start time to be less than 2 seconds for all platforms.
- **Memory Leak Check**: The memory leak check is performed by running the application for 24 hours and checking the memory usage of the application. We require the delta of memory usage to be less than 100 MB compared to the initial state.

4.4 Frontend: AI-Assisted Code Review

The frontend codes mainly include UI manipulation and network communication. The UI manipulation is implemented using corresponding frameworks and libraries, which are already robust and well-tested. Therefore, the main focus of the code review is on the network communication part.

We used the AI-assisted apporach to statically analyze the network communication code of AirX to find potential exploits and vulnerabilities.

5 Achievements and Experiences

AirX is a cross-platform text and file sharing system that pushes the boundaries of interoperability. Not limited by device types, it enables users to effortlessly transfer text and files between their devices, thus creating a cohesive digital environment.

5.1 Blurring the Lines Between Physical Devices

AirX broke the constraints of device-specific boundaries by providing a seamless file and text sharing experience across multiple platforms and creating a singular digital landscape where all your devices can interact with each other freely. This approach eliminates the need for repetitive processes or multiple application usage, saving valuable time and increasing productivity.

5.2 Seamless Cooperation with Apple AirDrop

Recognizing the efficiency of Apple AirDrop's automatic clipboard synchronization among Apple devices, AirX takes it a step further. It can sense data synchronized by AirDrop, further extending this synchronization to other platforms. This innovation signifies a breakthrough in cross-platform file and data sharing, allowing users to benefit from the advantages of both systems. It's a testament to the technological fusion that AirX represents, bridging the gap between various device ecosystems.

5.3 Development Experiences

The development of AirX is an ongoing process of innovation, collaboration, and testing. Leveraging the strengths of various technologies, it was built to adapt and grow with the changing technological landscape. This application's development journey involves multiple iterations, constant refinement, and a profound understanding of user needs, resulting in a robust and intuitive system that provides a truly seamless data sharing experience.

6 Problems and Solutions

6.1 Limitations of UDP Broadcast-based Discovery

Not all routers actively forward UDP broadcast packets. This is especially true for routers that are located in public areas or configured to isolate clients from each other. For example, the public wireless network of the Memorial University of Newfoundland drops all UDP broadcast packets, which prevents the discovery service of libairx from working.

To workaround this issue, we would suggest users to log in to their AirX accounts to enable clipboard sharing over the Internet.

6.2 WebSocket Source Address Hidden by Reverse Proxy

The backend server rely on the original address that stored in WebSocket sessions to communicate with clients. However, the originating address of WebSocket connections is hidden by the reverse proxy.

This issue was solved by adding a custom header to WebSocket handshake requests to store the original address.

6.3 False Positive CORS Error from Google Chrome

Google Chrome reports all non-200 HTTP responses whose `Access-Control-Allow-Origin` is not set as CORS errors. For example, if the server returns a 401 Unauthorized error, Google Chrome will report a CORS error instead, which is misleading and confusing.

The problem was addressed by forcing the server to attach permissive CORS headers to all responses, regardless of the status code.

6.4 libairx Android Architecture Cross-Comilation Issues

From version r23, the Android NDK no longer ship with libgcc, which is required by Rust to cross-compile libairx for Android. Instead, libunwind is suggested as a replacement.

However, the Rust-side toolchain had not follow this change yet, which results in a compilation error. The problem was solved by a patch to Android NDK to force libunwind to link.

6.5 WinUI 3 Functionality and Performance Issues

Despite the three-year tenure of WinUI 3, there are still fundamental user interface (UI) operations that are not directly supported. Instances of such include:

- **UI Platform components:** Modification of UI location or window visibility necessitates the use of the Win32 API.
- **Window startup placement:** This property of the Window class is utilized to determine the initial position of a window.
- **Hide control box:** This capability permits the developer to conceal the control box of a window, which typically contains the minimize, maximize, and close buttons.
- **Hide a window:** This permits the developer to conceal a window without terminating it.

There are also several issues that presently lack solutions:

- **Performance decline:** WinUI 3 experiences a notable decrease in performance when the window is displayed on screens with a vertical refresh rate exceeding 60Hz.
- **Tree-shaking issue:** The current Windows App SDK does not support tree-shaking of WinUI 3, resulting in a final binary size that is significantly larger. An empty project with a single blank window can have a final package size exceeding 200 MB, roughly 250% of that of Electron.js.

The following are some of the shortcomings identified during the development process:

- **Subpar development experience:** At present, there is no designer available, forcing developers to directly write to a XAML file. Consequently, it is critical for developers to have a certain degree of mental rendering ability.
- **Weak ecosystem:** Given the relatively short official release time of WinUI 3, there are few third-party controls available. Developers often find themselves needing to write controls manually.
- **Limited interoperability:** Despite the indisputable performance of .NET 6, WinUI is a Native framework, and control dependency property read-write operations need CsWinRT for interoperability. This is followed by a slow compilation process, which has been tested to be significantly slower than WPF.

It is hopeful that future version updates will see Microsoft incorporate these related UI operations into WinUI 3, thereby offering enhanced benefits to developers.

6.6 Windows App SDK MSIX Packaging Issues

The current Windows App SDK does not support portable MSIX packaging, making it impossible to package the application as a single executable (exe) file. Instead, only MSIX bundle packaging is supported, which requires the user to install the application through the Microsoft Store.

There is no feasible solution to this problem at present. However, Microsoft has stated that they are working on a solution to this problem, and it is expected to be available in the next version of Windows App SDK.

6.7 React Native Java Bridge Performance Issues

The React Native Java Bridge is a mechanism that enables communication between JavaScript and native code. As previously mentioned in the literature review section, the Java Bridge is a performance bottleneck for React Native applications.

A potential solution to this problem is to reduce the number of interactions between JavaScript and native code. This can be achieved by reducing the frequency of progress updates for file transfer.

6.8 Android Security Measures

As Google is enforcing a series of security measures on Android, it is impossible for normal applications to read the clipboard in the background.

There is no bypass for this security measure at present. However, for advanced users who have rooted their Android devices, AirX can function normally if it is installed as a system application.

From Android 10 onwards, all changes made to the system partition are temporary and Magisk is widely used to safely install system extensions. Therefore a Magisk module could be developed to address the permission issue.

7 Future Work

This section will discuss the upcoming work and potential upgrades planned for AirX, which could greatly enhance its performance and usability.

7.1 Magisk-based Android Client

As previously mentioned in the "Problems and Solutions" section, we need a Magisk module for AirX to work properly on current Android devices. Magisk helps install AirX as a system application, thus granting the necessary permissions to read the clipboard in the background.

Future work will focus on thorough testing and refining of this Magisk-based client. We will pay special attention to making it compatible with a wide range of Android versions and devices, simplifying the installation process, and ensuring top-notch data security and privacy for users.

7.2 Linux Client

At the same time, we are also developing a Linux client using Rust. Currently, the Linux client is expected to launch with only Command Line Interface (CLI) support. It is being designed to work with a particular set of Linux utilities such as Vi Improved (Vim), Neovim, and GNU Nano.

Our goal is to extend the convenience of file and text sharing provided by AirX to Linux users. Work will involve optimizing the CLI support, ensuring smooth interaction with preferred Linux utilities, and establishing stable and reliable operation.

7.3 File Preview

In the future, we aim to enhance the user experience by introducing a file preview feature in AirX. This feature would allow users to get a quick look at the contents of received files and shared links. For example, they could see the first page of a PDF file or the opening frame of a video file.

Work in this area will involve extending the preview feature to support a wide range of file formats, as well as tackling any potential performance issues.

7.4 "Ctrl + C + C" Mode

We are also considering the addition of a "Ctrl + C + C" mode, which extends the standard "Ctrl + C (or Command + C)" copy shortcut. This feature would allow users to share their copied data with all connected devices by pressing an extra "C" key.

The intention behind this mode is to bypass the need to manually open the AirX client to share data. Enhancements will involve improving the response time of this key combination and ensuring support on all platforms.

7.5 UI Improvements

We also plan to improve AirX's user interface by adding more animations and effects. For instance, the "Copy" button could be animated to visually indicate when data has been successfully copied.

Our focus will be on making the UI more intuitive, pleasing to the eye, and interactive. We will also work on optimizing the performance of the animations to ensure smooth operation and prevent potential delays or glitches.

7.6 iOS and iPadOS Client

While iOS and iPadOS applications have restrictions on reading the clipboard in the background, we still plan to develop a version of AirX for these platforms. Even if it only supports manual file transmission initially, it will broaden the user base of AirX.

In the future, we aim to overcome these limitations and offer a fully functional version of AirX to iOS and iPadOS users, all while adhering to the guidelines and rules of the platform.

8 Open Source and Private Server Deployment

The AirX series are all open-source under the MIT license. The source code is hosted on GitHub and can be accessed at:

- **libairx** - https://github.com/hatsune-miku/libairx
- **Backend** - https://github.com/hatsune-miku/airx-backend
- **Cloud Storage Web Frontend** - https://github.com/hatsune-miku/airx-cloud
- **Windows Frontend** - https://github.com/hatsune-miku/AirX-win
- **macOS Frontend** - https://github.com/Lsjy44/airX_mac.git
- **Android Frontend** - https://github.com/hatsune-miku/airx4a

The private server deployment is documented in the README of the backend repository.

References

[1] J. Wu and J. Dong, "A simple service discovery and configuration protocol for embedded devices," in *2006 International Conference on Communication Technology*, pp. 1–3, 2006.

[2] R. Wittmann, *Multicast communication protocols and applications*. Morgan Kaufmann series in networking, San Francisco, CA: Morgan Kaufmann, 2001.

[3] P. C. Reddy, "Tcp over ieee 802.11," *ArXiv*, vol. abs/1202.5941, 2012.

[4] W. Feng and P. Tinnakornsrisuphap, "The failure of tcp in high-performance computational grids," in *SC '00: Proceedings of the 2000 ACM/IEEE Conference on Supercomputing*, pp. 37–37, 2000.

[5] M. Pagani and M. Plogas, "Modernizing your windows applications with the windows app sdk and winui: Expand your desktop apps to support new features and deliver an integrated windows 11 experience," in *Modernizing Your Windows Applications with the Windows App SDK and WinUI: Expand your desktop apps to support new features and deliver an integrated Windows 11 experience*, 2022.

[6] P. Wiertel and M. Skublewska-Paszkowska, "Comparative analysis of uikit and swiftui frameworks in ios system," *Journal of Computer Sciences Institute*, vol. 20, pp. 170–174, Sep 2021.

[7] O. Gill, "Using react native for mobile software development," in *Using React Native for mobile software development*, 2018.

[8] M. Haekal and Eliyani, "Token-based authentication using json web token on sikasir restful web service," in *2016 International Conference on Informatics and Computing (ICIC)*, pp. 175–179, 2016.

[9] G. Wang, J. Koshy, S. Subramanian, K. Paramasivam, M. Zadeh, N. Narkhede, J. Rao, J. Kreps, and J. Stein, "Building a replicated logging system with apache kafka," *Proc. VLDB Endow.*, vol. 8, pp. 1654–1655, aug 2015.

[10] AppleInc., "Swiftui documentation." https://developer.apple.com/xcode/swiftui/, 2023. Accessed: 2023-07-06.

[11] N. Hossain, M. A. Hossain, M. Z. Hossain, M. H. I. Sohag, and S. Rahman, "Oauth-sso: A framework to secure the oauth-based sso service for packaged web applications," in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pp. 1575–1578, 2018.

[12] A. K. Zaki and I. M., "A novel redis security extension for nosql database using authentication and encryption," in *2015 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, pp. 1–6, 2015.

[13] L. B. Martinkauppi, Q. He, and D. Ilie, "On the design and performance of chinese oscca-approved cryptographic algorithms," in *2020 13th International Conference on Communications (COMM)*, pp. 119–124, 2020.