# EELU Operating Systems Project 12/2026

# Chat App – Socket & Shared Memory Documentation

**Team Members:**

Ahmed Gamal Moawad El-Bardisi

Mohannad Khaled

Omar Khaled

Fathi Ahmed

Mohab Waleed

**PROJECT TITLE :** Chat-app-Socket-SharedMemory

**SUMMARY:** A high-performance, single-executable Windows desktop application for secure messaging via LAN (Socket Mode) or Local Inter-Process Communication (Shared Memory Mode).

## TABLE OF CONTENTS

## 1. QUICK START

To get up and running immediately:

CLONE: Retrieve the repository.

```
git clone https://github.com/AhmedGamal17-9/Chat-app-Socket-SharedMemory.git
```

PREPARE: Ensure MSYS2 UCRT64 toolchain is installed.

BUILD: Run the build script in the project root.

```
build.bat
```

RUN: Execute the binary.

```
ChatSuite.exe
```

## 2. PREREQUISITES

Ensure your environment meets the following specifications before deployment or development.

OPERATING SYSTEM:

Windows 10 or Windows 11 (64-bit architecture is recommended).

TOOLCHAIN:

MSYS2 (Win64) with UCRT64 environment.

REQUIRED PACKAGES (Pacman):

mingw-w64-ucrt-x86_64-gcc (C++ Compiler & Linker)

mingw-w64-ucrt-x86_64-make (Optional, for Makefile users)

CRITICAL REQUIREMENT:

You MUST add a "Folder Exclusion" in Windows Defender (or your primary Antivirus) for the project directory. The application performs raw socket operations and direct memory manipulation which may trigger heuristic security warnings. See Section 12 for details.

## 3. DEVELOPMENT SETUP

Follow these steps to configure a robust development environment.

INSTALL MSYS2:

Download from

https://www.msys2.org/](https://www.msys2.org/

and install to default location.

INITIALIZE ENVIRONMENT:

Open the "MSYS2 UCRT64" terminal. Update the package database:

```
pacman –Syu
```

INSTALL COMPILER:

Install the full GCC toolchain for UCRT64:

```
pacman -S mingw-w64-ucrt-x86_64-gcc
```

NAVIGATE TO SOURCE:

```
cd /c/Projects/Chat-app-Socket-SharedMemory
```

(Adjust path according to your local clone location).

## 4. BUILD INSTRUCTIONS

We provide two methods for building the application: an automated script and manual compilation.

OPTION A: AUTOMATED BUILD (Recommended)

1. Locate build.bat in the project root.
2. double-click on build.bat

OPTION B: MANUAL COMPILATION (MSYS2 CLI)

For advanced control or CI/CD pipelines, use the direct compiler command:

```
g++ -std=c++17 -municode -mwindows src/*.cpp -o ChatSuite.exe -lws2_32 -lgdi32 -lcomctl32 -
static-libgcc -static-libstdc++
```

BUILD FLAGS EXPLAINED:

mwindows: Creates a GUI application (no console window).

municode: Enables full Unicode support (essential for modern Windows APIs).

static-lib : Statically links libraries for portability (no DLLs needed on client).

## 5. RUNTIME USAGE

The application operates in two distinct modes.

Ensure you select the correct mode for your network environment.

### MODE 1: SOCKET MODE (LAN Communication)

Ideal for chatting between different computers on the same Local Area Network.

A. SERVER SETUP (Host):

1. Run ChatSuite.exe.
2. Select "Socket Mode".
3. Role: Select "Server".
4. Port: Enter a valid port (default 8888).
5. Click "Start".
6. Status Check: Ensure log says Server listening on 0.0.0.0:8888.

B. CLIENT SETUP (Remote User):

1. Run ChatSuite.exe.
2. Select "Socket Mode".
3. Role: Select "Client".
4. IP Address: Enter the Server’s Local IP (e.g., 192.168.1.50).
5. Port: Enter the same port (8888).
6. Click "Connect".

## MODE 2: SHARED MEMORY MODE (Local IPC)

Ideal for secure, offline chat between two users on the SAME physical machine (e.g., Guest vs Admin).

1. User A:

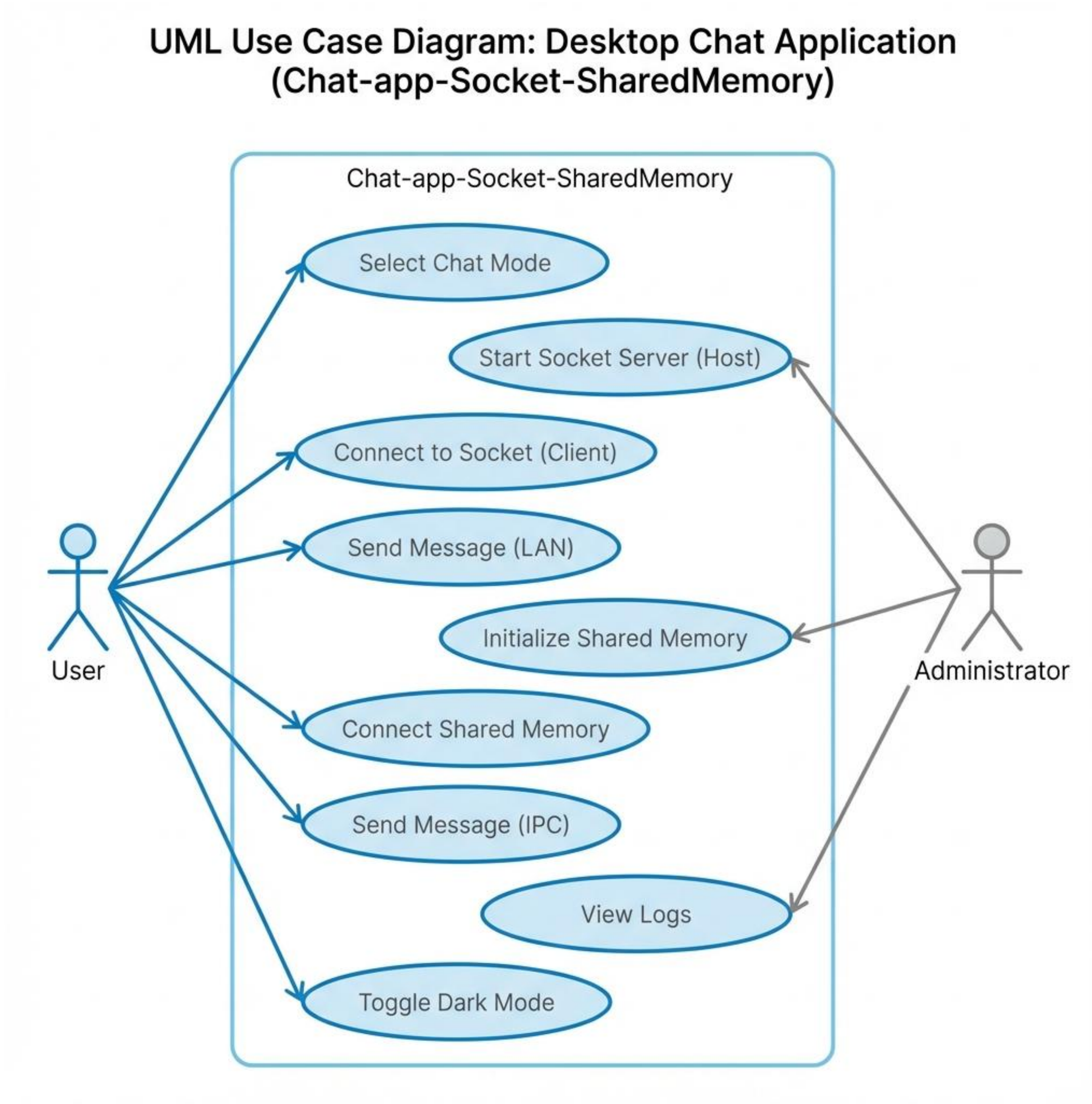
   - Open ChatSuite.exe -> "Shared Memory Mode".
   - Click "Start/Connect".
   - Status: Created new mapping.

2. User B:

   - Open ChatSuite.exe -> "Shared Memory Mode".
   - Click "Start/Connect".
   - Status: Connected to existing mapping.

## 6. USE CASES

Real-world scenarios demonstrating practical application usage.


UML Use Case Diagram: Desktop Chat Application
(Chat-app-Socket-SharedMemory)
Chat-app-Socket-SharedMemory
Select Chat Mode
Start Socket Server (Host)
Connect to Socket (Client)
Send Message (LAN)
Initialize Shared Memory
Connect Shared Memory
Send Message (IPC)
View Logs
Toggle Dark Mode
User
Administrator

## SCENARIO 1: Small Office Team Coordination (Socket Mode)

Context: A small dev team needs a quick communication channel without relying on external cloud services (Slack/Discord) for security.

- Setup: The Team Lead starts the "Server" on their machine (IP: 10.0.0.5).

- Action: Three other developers start "Client" mode and connect to 10.0.0.5.

- Result: Instant, low-latency, private LAN messaging for sensitive code discussion.

## SCENARIO 2: Secure Air-Gapped Communication (Shared Memory Mode)

Context: A classified workstation with no internet access requires communication between two separate secure user sessions or processes.

- Setup: The machine acts as a secure terminal. Both users launch the application locally.

- Action: Using Shared Memory, text data is passed directly through RAM without ever touching the network stack.

- Result: Zero network footprint. The data never leaves the volatile memory of the specific machine.

## SCENARIO 3: Network Diagnosis & Troubleshooting (Logging Usage)

Context: An IT admin is debugging why a legacy machine cannot connect to the chat server.

- Setup: Admin launches the application in Socket Mode on the problematic machine.

- Action: Attempts to connect. Checks the on-screen "Log" window.

- Observation: Log shows [Error] Winsock Connect Failed: 10060.

- Resolution: Admin identifies 10060 as a connection timeout, points to a Firewall block, and creates an exception for the specific port.

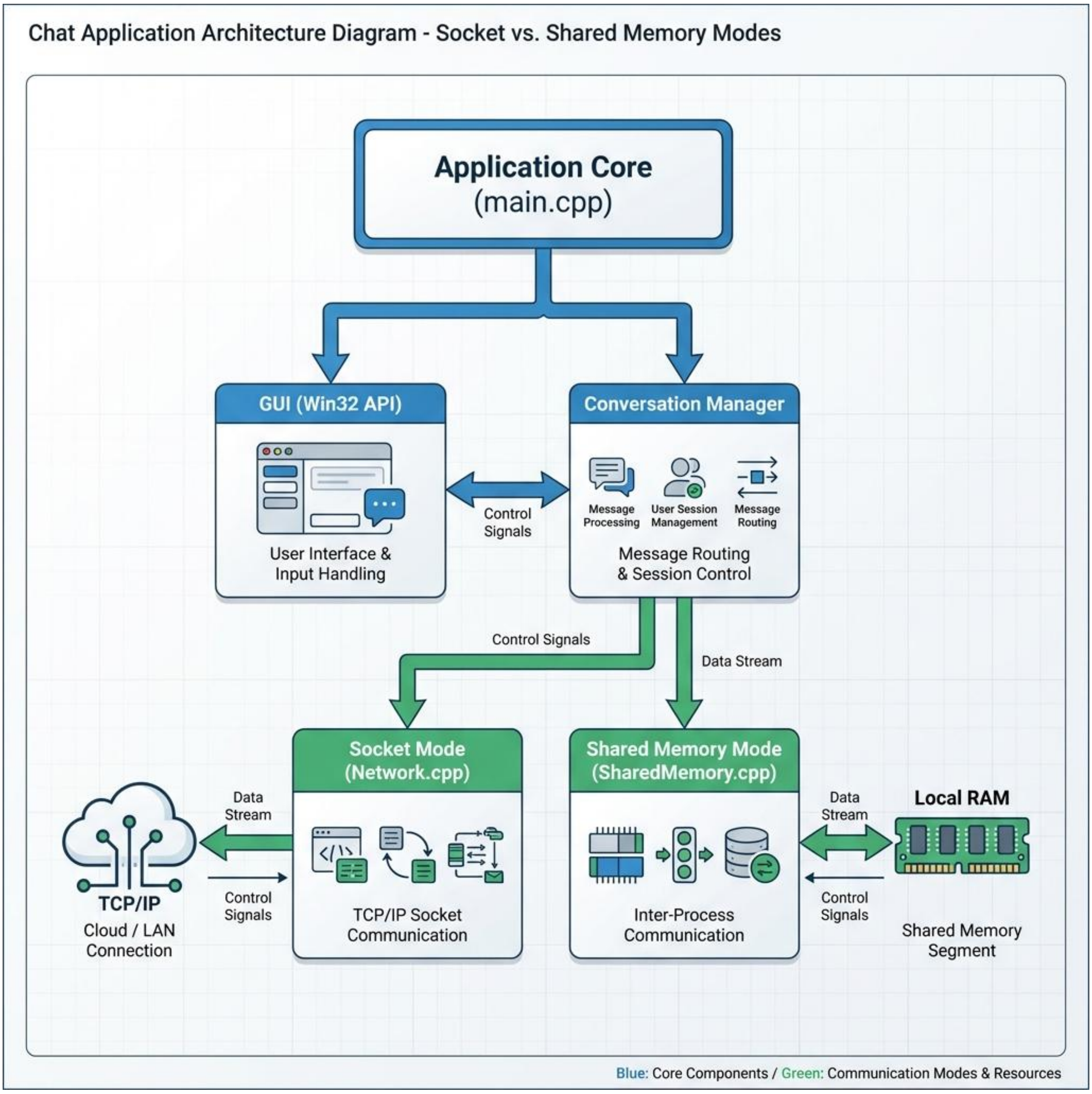
## 7. PROJECT STRUCTURE

A clean, modular structure ensures maintainability.

```
Chat-app-Socket-SharedMemory /
├── build.bat                # Windows Batch build script
├── ChatSuite.exe            # Final production binary
├── README.md                # Markdown overview
├── Documentation/
│   └── Documentation.txt    # This detailed reference file
└── src/
    ├── main.cpp             # Application entry, message loop
    ├── Gui.h / .cpp         # Win32 API rendering & event handling
    ├── Network.h / .cpp     # Winsock2 TCP abstractions
    ├── SharedMemory.h / .cpp # Named Shared Memory implementation
    └── Logger.h             # Thread-safe logging singleton
```

## 8. ARCHITECTURE OVERVIEW

The system follows a modular, event-driven architecture separating UI, Logic, and Data layers.


Chat Application Architecture Diagram - Socket vs. Shared Memory Modes
Application Core (main.cpp)
GUI (Win32 API)
User Interface & Input Handling
Control Signals
Conversation Manager
Message Processing
User Session Management
Message Routing
Message Routing & Session Control
Control Signals
Data Stream
Socket Mode (Network.cpp)
TCP/IP Socket Communication
Shared Memory Mode (SharedMemory.cpp)
Inter-Process Communication
Data Stream
Control Signals
TCP/IP
Cloud / LAN Connection
Data Stream
Control Signals
Local RAM
Shared Memory Segment
Blue: Core Components / Green: Communication Modes & Resources

## 9. CONFIGURATION & CLI OPTIONS

CURRENT STATUS:

The application is designed as a strict GUI-first utility. No CLI arguments are parsed at startup.

RUNTIME CONFIGURATION:

All settings (IP, Port, Mode) are input dynamically via the UI form fields at launch.

FUTURE ROADMAP (Config.ini):

Support for config.ini is planned for version 1.1:

```
[Network]

DefaultPort=8888

AutoConnect=false
```

## 10. LOGGING & DEBUGGING

LOGGING SYSTEM:

- Type: Ephemeral In-Memory Logging.
- Visibility: Logs are rendered in the lower "Status/Log" pane of the GUI.
- Persistence: None (Logs are cleared on exit).

SAMPLE OUTPUT:

```
 [14:20:00] [System] Application initialized.
[14:20:05] [Network] Socket created successfully.
[14:20:12] [Error] Connection Refused (Is the server running?).
```

DEBUGGING TIPS (Developer):

- Use GDB via MSYS2 for runtime inspection:

```
gdb ChatSuite.exe
```

- If the UI freezes, checking the infinite loop in Network.cpp (listener thread) is the recommended first step.

## 11. TESTING

RECOMMENDED MANUAL TEST PROTOCOLS:

TEST A: LOOPBACK SOCKET TEST

1. Target: Verify TCP stack integrity.
2. Action: Launch Server on localhost (127.0.0.1). Launch Client on 127.0.0.1.
3. Validation: Send message "SYN". Confirm receipt of "SYN" on Server instantly.

TEST B: CONCURRENCY STRESS TEST (Shared Memory)

1. Target: Verify mutex locking mechanism.
2. Action: Open two instances. Spam send button on Instance A while typing on Instance B.
3. Validation: No crashes, no garbled text. The Semaphore should block writes until the buffer is clear.

## 12. SECURITY & ANTIVIRUS NOTES

IMPORTANT SECURITY NOTICE:

FALSE POSITIVES:

- This application utilizes CreateFileMapping and raw socket calls.
- Heuristic engines (Windows Defender, Avast) may flag this as "Generic.Malware" due to lack of a digital signature.
- ACTION: Add an EXCLUSION for the build folder.

  Windows Settings -> Update & Security -> Windows Security

  Virus & threat protection -> Manage Settings -> Exclusions -> Add Folder.

NETWORK ENCRYPTION:

The current version sends data in PLAINTEXT.

- Do NOT send passwords, credit card data, or PII over public networks.

- Use only on trusted LANs or VPN-secured networks.

## 13. RELEASE & DISTRIBUTION

To prepare a "Production Release":

1. STATIC LINKING (Mandatory):

   Ensure

   ```
   static-libgcc -static-libstdc++
   ```

   are in your build command. This prevents "Missing .dll" errors on end-user machines.

2. COMPRESSION:

   Zip the ChatSuite.exe.

   The file size will likely be around 2MB due to static linking.

3. SIGNING (Optional):

   For professional distribution, sign the binary with a Code Signing Certificate (e.g., Sectigo, DigiCert) to remove the "Unknown Publisher" warning.

## 14. CONTRIBUTION GUIDE

We welcome contributions! Please adhere to professional standards.

BRANCHING STRATEGY:

- main : Stable releases.
- dev : Integration branch for ongoing work.
- feat/name : New features (e.g., feat/dark-mode).
- fix/name : Bug fixes (e.g., fix/socket-timeout).

COMMIT CONVENTIONS (Semantic):

- feat: (new feature for the user, not a new feature for build script)
- fix: (bug fix for the user, not a fix to a build script)
- docs: (changes to the documentation)
- style: (formatting, missing semi colons, etc; no production code change)
- refactor: (refactoring production code, eg. renaming a variable)

PULL REQUEST REQUIREMENTS:

- Code must compile with -Wall (warnings are acceptable but should be minimized).
- All Manual Tests (Section 11) must pass.

## 15. TROUBLESHOOTING

| Issue | Potential Cause | Solution |
|---|---|---|
| g++ not found | MSYS2 path missing. | Run command in specific MSYS2 UCRT64 shell, or add C:\msys64\ucrt64\bin to System PATH. |
| Connection Failed | Firewall block. | Allow ChatSuite.exe through Windows Firewall (Private & Public). |
| Access Denied | Shared Memory conflict. | Ensure no "zombie" processes are running. Terminate all ChatSuite.exe in Task Manager. |
| Start" Click Ignored | Invalid Input | Ensure Port is a number (1024-65535) and IP is valid format. |

## 16. FAQ

Q: Is internet chat supported without VPN?

A: Generally, no. Socket Mode requires a direct line of sight. For internet chat, you must forward ports on your router, which exposes you to security risks.

Q: Why is the file size larger than expected (~2MB)?

A: We prioritize portability. By statically linking standard libraries, users don't need to install MinGW or C++ Runtime Redistributables.

Q: Can I transfer files?

A: Currently, no. We support text-based payloads only to keep the packet structure simple.

## 17. CHANGELOG

v1.0.0 – Initial Release [2024-12-17]

- [Feature] Dual Mode Support: Socket (TCP) and Shared Memory (IPC).

- [Feature] Native GUI: Pure Win32 API implementation.

- [Chore] Project Docs: Comprehensive documentation and build tools added.

## 18. LICENSE

## 19. APPENDIX: FUTURE ENHANCEMENTS

Planned features for upcoming sprints:

- [ ] Async I/O (Overlapped Sockets) to prevent UI blocking.

- [ ] Config File Loader (config.ini) for persistent settings.

- [ ] Tiny-AES Encryption for payload security.

- [ ] System Tray integration for background operation.