Project Schematic - By: LootSpam
Date: March 3, 2024
Website: www.nosignup.trade (Bottom-Left, Monero Button, Under Construction)

Objective

Develop a cross-platform, open-source Monero trading application using Flutter, designed to run natively on both Android and Windows from a single codebase.

The app will offer two installation options:

- Android Version – A fully self-contained APK that operates independently.
- Windows Version – A fully self-contained EXE that runs as a native desktop application.

Both versions must support full trading functionality without requiring Google services, proprietary APIs, or centralized servers. The app must operate entirely peer-to-peer, leveraging Monero nodes and the Haveno Daemon.

Requirements

Platform Compatibility & Open-Source Compliance

- **Android:** Fully self-buildable APK, compliant with Play Store & F-Droid.
- **Windows:** Fully self-buildable EXE using Flutter's desktop framework.
- **No Proprietary Dependencies:**
  - No Google Play Services or proprietary libraries.
  - Fully functional on de-Googled Android devices.
  - Must comply with F-Droid's self-buildability standards.
  - Haveno Daemon is started dynamically using Flutter's process management.
  - The app connects to Haveno Daemon and Monero Node using gRPC API calls.

Privacy & Security

- No telemetry, tracking, or analytics.
- No user data collection, storage, or transmission.
- End-to-end encrypted chat for secure trading communication.
- Tor integration for anonymous trading (optional).

Installation, Dependencies & Execution

Single Installation Per Platform

The app must be compilable and installable to run natively on both platforms:

- **Android:** Runs as a fully self-contained APK.
- **Windows:** Runs as a fully self-contained EXE.

First-Launch Dependency Check

- **Dependency:** Haveno Daemon (`daemon.jar`)
- **Dependency:** Termux (for Android)
- **Dependency:** Java (OpenJDK 21, if not already installed)
- If any dependency is missing → Prompts the user to approve a secure download before installing.

- Haveno Daemon Mirror: https://www.nosignup.trade/monerodaemon/daemon.jar

Once all dependencies are installed, the app executes the Haveno Daemon as a background process, and Flutter's gRPC service automatically detects and connects to it. The Daemon uses gRPC calls to communicate with the device it is on, either Android or Windows.

Security & Compliance

- F-Droid Compliance: No prepackaged binaries. APK must download dependencies **post-installation**.
- Integrity Check: Downloads are verified via cryptographic checksums before execution.
- Execution Controls:
  - Requires explicit user approval before setting binaries as executable.
  - If download/installation fails, the app automatically retries.
  - After multiple failures, the user may manually enter an alternative download URL.

Fully Offline & Peer-to-Peer Execution

- The app runs Haveno Daemon locally on both Android and Windows.
- Connects directly to Monero peer-to-peer nodes—no third-party servers.
- Ensures trading, chat, and order book fetching work seamlessly on both platforms.

Open-Source Compliance & Self-Sufficiency

The app must be fully independent and community-driven, ensuring transparency, privacy, and FOSS compliance.

Fully Open-Source:

- The complete source code will be hosted on GitHub.
- The APK & EXE must be self-buildable from source to meet F-Droid and open-source standards.
- Weekly/bi-weekly source code deliveries are mandatory for tracking progress.

No Centralized Dependencies:

- The app will not rely on cloud services or proprietary APIs.
- All functionality is performed locally using Monero nodes and the Haveno Daemon.

Community Verification & Transparency:

- Code must be structured for easy peer review and contributor onboarding.
- Build instructions and reproducibility checks will be documented to facilitate community auditing.

Features

1. Order Book Fetching

- Retrieves live buy/sell offers from the Haveno Daemon.
- Automatically sorts offers by price and availability.
- Users can filter specific trade pairs for better visibility.

2. Managing Trade Offers

- Users can create, modify, and cancel trade offers.
- Trade offers use gRPC API calls for secure, peer-to-peer transactions.
- Minimum trade amounts can be set.
- Optional Tor integration allows anonymous trading.

3. Real-Time Encrypted Chat

- End-to-end encrypted messaging using public-key cryptography.
- No centralized storage—messages exist only during a session.
- No file sharing or voice messages for security.
- Secure data synchronization across active sessions.
- Uses JWT-based authentication for secure access.
- Optional Tor integration for enhanced privacy.

4. Monero Node Connectivity

- The app automatically detects and connects to an available Monero node.
- Users may manually enter a custom Monero node URL for better reliability.

5. Tor-Based Anonymity

- Users can enable Tor routing through the Haveno Daemon.
- The daemon will confirm whether Tor is running before routing transactions.
- On Android, Orbot integration is required for (optional) Tor-enabled sessions.

Haveno Integration in Flutter

- The app uses **Flutter's gRPC package** to communicate with Haveno Daemon.
- All API calls are handled using Dart-based gRPC services.
- Haveno Daemon (daemon.jar) is executed as a background process on Android & Windows.
- On Windows: The user is prompted to install OpenJDK if Java is missing.
- On Android: The user is prompted to install Termux & OpenJDK to execute the daemon.
- Once started, Flutter automatically detects and connects to the daemon via gRPC (localhost:9999).

Key Resources:

Haveno Dart Client Repository, GitHub: Haveno-Dart:

(https://github.com/haveno-dex/haveno-dart)

API Documentation, Haveno API:
(https://pub.dev/documentation/haveno/latest/)

Haveno TypeScript Repository, GitHub: Haveno-TS:
(https://github.com/haveno-dex/haveno-ts)

Command Line Emulator, Termux (Required for Android Java Execution):
(https://f-droid.org/en/packages/com.termux/)

Java SDK Sources (OpenJDK 21 for Haveno Daemon):
(https://adoptium.net/temurin/releases/)

Development, Testing & Distribution

The app will be developed using Flutter and undergo rigorous security audits to ensure:

- Ensure Haveno Daemon downloads dynamically and runs via gRPC API.
- Monero privacy compliance for secure transactions.
- F-Droid and Play Store compliance for seamless distribution.
- Tor routing functionality (if enabled).

Once development and testing are complete, (Nosignup.Trade) will host:

- Compiled app and dependency downloads (i.e. a secure mirror to the pre-compiled daemon).
- Project documentation and checksums for all pre-compiled downloads.
- Source code repository links.

Project Timeline, Expectations & Roadmap

Phase 1: Initial Setup & Planning **(✓ Completed! ✓)**

- A full schematic outlining the app's structure, data flow, and API interactions.
- Definitions and pseudocode for project structure, UI design, and authentication system.

Phase 2: Core Feature Integration

- Implement Haveno Daemon, Monero Node, and order book functionality.

Phase 3: UI/UX Development

- Finalize the Flutter-based UI.
- Ensure seamless Android & Windows adaptations.

Phase 4: Security & Testing

- Perform cross-platform testing.
- Verify Tor integration and Monero compliance.

Phase 5: Deployment & Distribution

- The total project budget is fixed and covers development, launch, and website integration.
- The final app will be fully open-source after release.
- Source code provided weekly to bi-weekly for review & reporting.
- Prepare for F-Droid release.
- Integrate with the project website.
- Final bug fixes and optimizations.

Key Pseudocode Implementations

Note: This code is not perfect; please revise and validate before attempting to implement it in its current state.

System Setup & Dependency Management

1. Check & Install Haveno Daemon and Dependencies

```
function ensureDaemonInstalled() {
   /**
    * Ensures the Haveno Daemon and required dependencies are installed on the system.
    * If missing, downloads it from the official source.
    *
    * @returns {string} Path to daemon.jar
    * @throws {Error} If installation fails.
    */

   let daemonPath = (isAndroid())
      ? "/data/data/com.haveno/files/daemon.jar"
      : "C:/Program Files/Haveno/daemon.jar";

   if (fileExists(daemonPath)) {
      console.log("Haveno Daemon is already installed.");
      return daemonPath;
   }

   console.log("Haveno Daemon not found. Checking dependencies...");

   let systemArch = getSystemArchitecture();
   console.log("Detected System Architecture: " + systemArch);

   if (isAndroid()) {
      if (!isTermuxInstalled()) {
         throw new Error("Termux is required to execute the daemon on Android. Download for " + systemArch + ":
https://f-droid.org/en/packages/com.termux/");
      }
      if (!isJavaInstalled()) {
         alertUser("Java (OpenJDK 21) is required. Install via: pkg install openjdk-21 (Detected: " + systemArch + ")");
```

```
            throw new Error("Java not installed.");
        }
    } else if (!isJavaInstalled()) {
        alertUser("Java (OpenJDK 21) is required on Windows. Install from: https://adoptium.net/temurin/releases/?arch=" + systemArch);
        throw new Error("Java not installed.");
    }

    console.log("Downloading Haveno Daemon...");

    let downloadURL = "https://www.nosignup.trade/monerodaemon/daemon.jar";
    let retries = 3;

    while (retries > 0) {
        if (downloadFile(downloadURL, daemonPath)) {
            console.log("Haveno Daemon successfully installed.");
            if (isAndroid()) {
                runShellCommand(["chmod", "+x", daemonPath]); // ensure execution permissions on android
            }
            return daemonPath;
        }
        retries--;
        console.log("Download failed, retrying... (" + retries + " attempts left)");
    }

    throw new Error("Failed to download Haveno Daemon after multiple attempts.");
}

/**
 * Detects the system architecture (x86, x86_64, arm64, etc.)
 *
 * @returns {string} The system architecture identifier.
 */
function getSystemArchitecture() {
    try {
        let arch = runShellCommand(["uname", "-m"]).trim(); // For Linux/Android
        if (!arch) {
            arch = runShellCommand(["wmic", "OS", "get", "OSArchitecture"]).trim(); // Windows
        }
        return arch || "unknown";
    } catch (e) {
        return "unknown";
    }
}

/**
 * Checks if Java is installed.
 *
 * @returns {boolean} True if Java is installed, otherwise false.
 */
```

```
function isJavaInstalled() {
    try {
        let result = runShellCommand(["java", "-version"]);
        return result.includes("openjdk") || result.includes("Java");
    } catch (e) {
        return false;
    }
}

/**
 * Alerts the user with installation instructions for missing dependencies.
 *
 * @param {string} message - The message to display to the user.
 */
function alertUser(message) {
    console.warn(message);
    // This should be replaced with a proper UI alert in Flutter
}
```

2. Start the Haveno Daemon as a Background Process

```
function runDaemon() {
    /**
     * Starts the Haveno Daemon as a background process.
     *
     * @returns {boolean} True if started successfully.
     * @throws {Error} If startup fails.
     */

    if (isDaemonRunning()) {
        console.log("Haveno Daemon is already running.");
        return true;
    }

    if (!isJavaInstalled()) {
        throw new Error("Java (OpenJDK 21) is required to run Haveno Daemon.");
    }

    let daemonPath = ensureDaemonInstalled();
    let command = ["java", "-jar", daemonPath];

    console.log("Launching Haveno Daemon...");
    let [success, output] = runShellCommand(command);

    if (!success) {
        throw new Error("Failed to start Haveno Daemon: " + output);
    }

    console.log("Haveno Daemon successfully started.");
    return true;
```

```
}

/**
 * Checks if Haveno Daemon is already running.
 *
 * @returns {boolean} True if daemon is running, otherwise false.
 */
function isDaemonRunning() {
   try {
      let result = runShellCommand(["pgrep", "-f", "daemon.jar"]);
      return result.trim() !== "";
   } catch (e) {
      return false;
   }
}
```

3. Verify Daemon is Running

```
function verifyDaemon(retries = 3) {
   /**
    * Verifies if the Haveno Daemon is running and reachable.
    *
    * @returns {Object} Daemon status response.
    * @throws {Error} If connection fails or daemon is not running.
    */

   let client;
   while (retries > 0) {
      try {
         client = HavenoClient.connect("localhost", 9999); // gRPC connection

         if (!client.isConnected()) {
            throw new Error("Failed to connect to Haveno Daemon.");
         }

         let status = client.daemonService.getStatus();
         console.log("Haveno Daemon is running:", status);
         return status;

      } catch (error) {
         console.error("Error verifying Haveno Daemon:", error.message);
         retries--;

         if (retries === 0) {
            throw new Error("Haveno Daemon is not running or unreachable.");
         }
      }
   }
}
```

4. Fetch Order Book

```
async function fetchOrderBook(market) {
    /**
     * Retrieves the order book from Haveno Daemon.
     *
     * @param {string} market - The trading market (e.g., "XMR_BTC")
     * @returns {object} Order book data.
     * @throws {Error} If the API call fails.
     */

    if (!isDaemonRunning()) {
        await runDaemon();
    }

    const client = HavenoClient.connect("localhost", 9999);
    if (!client.isConnected()) {
        throw new Error("Connection Error: Failed to connect to Haveno API.");
    }

    try {
        return await client.offerService.getOrderBook(market);
    } catch (error) {
        throw new Error("Error fetching order book: " + error.message);
    }
}
```

5. Create Trade Offer

```
async function createOffer(user, amount, price, useTor) {
    /**
     * Creates a new trade offer.
     *
     * @returns {object} Trade offer response.
     * @throws {Error} If the API call fails.
     */

    if (!isDaemonRunning()) {
        await runDaemon();
    }

    const client = HavenoClient.connect("localhost", 9999);
    if (!client.isConnected()) {
        throw new Error("Failed to connect to Haveno API.");
    }

    const offer = { user, amount, price, useTor };
```

```
    try {
        return await client.offerService.createOffer(offer);
    } catch (error) {
        throw new Error("Error creating trade offer: " + error.message);
    }
}
```

6. Execute Trade

```
async function executeTrade(user, offerId, retries = 3) {
    /**
     * Accepts a trade offer and executes the transaction.
     *
     * @param {string} user - User executing the trade.
     * @param {string} offerId - The trade offer ID to accept.
     * @returns {object} Trade execution result.
     * @throws {Error} If trade execution fails.
     */

    if (!isDaemonRunning()) {
        await runDaemon();
    }

    const client = HavenoClient.connect("localhost", 9999);
    if (!client.isConnected()) {
        throw new Error("Trade execution service unavailable.");
    }

    while (retries > 0) {
        try {
            const trade = await client.tradeService.acceptOffer(user, offerId);

            if (trade.status !== "successful") {
                throw new Error("Trade Execution Failed.");
            }

            return trade;
        } catch (error) {
            console.error(`Trade execution failed: ${error.message}`);
            retries--;

            if (retries === 0) {
                throw new Error("Trade execution failed after multiple attempts.");
            }
        }
    }
}
```

User Authentication & Security

7. Authenticate User with JWT

```
function getWalletBalance() {
   /**
    * Fetches the user's wallet balance.
    *
    * @returns {Object} Available & locked funds.
    * @throws {Error} If connection fails.
    */

   if (!isDaemonRunning()) {
      throw new Error("Haveno Daemon is not running. Please start the daemon first.");
   }

   const client = HavenoClient.connect("localhost", 9999);
   if (!client.isConnected()) {
      throw new Error("Failed to retrieve wallet balance.");
   }

   return client.walletService.getBalance();
}
```

8. Retrieve Wallet Balance

```
function logoutUser() {
   /**
    * Logs out the currently authenticated user.
    *
    * @returns {string} A success message confirming logout.
    * @throws {Error} If logout fails.
    */

   if (!isDaemonRunning()) {
      throw new Error("Haveno Daemon is not running. Unable to process logout.");
   }

   const client = HavenoClient.connect("localhost", 9999);
   if (!client.isConnected()) {
      throw new Error("Logout Service Unavailable");
   }

   try {
      client.accountService.logout();
      console.log("User successfully logged out.");
      return "User Logged Out";
   } catch (error) {
      throw new Error(`Logout failed: ${error.message}`);
   }
}
```

9. Logout User

```
function logoutUser() {
   /**
    * Logs out the currently authenticated user.
    *
    * @returns {string} A success message confirming logout.
    * @throws {Error} If logout fails.
    */

   if (!isDaemonRunning()) {
      throw new Error("Haveno Daemon is not running. Unable to process logout.");
   }

   let client = HavenoClient.connect("localhost", 9999);
   if (!client.isConnected()) {
      throw new Error("Logout Service Unavailable");
   }

   try {
      client.accountService.logout();
      console.log("User successfully logged out.");
      return "User Logged Out";
   } catch (error) {
      throw new Error(`Logout failed: ${error.message}`);
   }
}
```

Encrypted Communication & Tor Integration

10. Secure Real-Time Chat

```
function sendMessage(user, recipient, message, useTor) {
   /**
    * Securely sends an encrypted chat message over the Haveno network.
    *
    * @param {string} user - Sender's identifier.
    * @param {string} recipient - Recipient's identifier.
    * @param {string} message - Plaintext message to be encrypted.
    * @param {boolean} useTor - Whether to route through Tor.
    *
    * @throws {Error} If message sending fails.
    */

   if (!user || !recipient || !message) {
      throw new Error("Invalid chat message parameters!");
   }

   if (!isDaemonRunning()) {
      throw new Error("Haveno Daemon is not running. Unable to send message.");
```

```
    }

    let encryptedMessage;
    try {
        encryptedMessage = encrypt(message, recipient.publicKey);
    } catch (error) {
        throw new Error("Message encryption failed: " + error.message);
    }

    let client = HavenoClient.connect("localhost", 9999);
    if (!client.isConnected()) {
        throw new Error("Chat connection error - Unable to reach Haveno network.");
    }

    let chatMessage = {
        sender: user,
        receiver: recipient,
        message: encryptedMessage,
        timestamp: new Date().toISOString()
    };

    let connection = useTor ? TorProxy.connect(client.chatService) : client.chatService;

    try {
        connection.sendMessage(chatMessage);
        console.log("Message sent successfully.");
    } catch (error) {
        throw new Error(`Failed to send message: ${error.message}`);
    }
}
```

11. Tor Routing Configuration

```
function configureTor(enable) {
    console.log(`Attempting to ${enable ? "enable" : "disable"} Tor routing...`);

    if (!isDaemonRunning()) {
        throw new Error("Haveno Daemon is not running. Unable to configure Tor.");
    }

    let client = HavenoClient.connect("localhost", 9999);
    if (!client.isConnected()) {
        throw new Error("Tor Configuration Failed - Haveno Daemon unreachable.");
    }

    try {
        let currentStatus = client.networkService.getTorStatus();
        if (currentStatus.enabled === enable) {
            console.log(`Tor is already ${enable ? "enabled" : "disabled"}.`);
            return true;
```

```
        }

        let result = client.networkService.configureTor(enable);
        if (result.success) {
            console.log(`Tor routing successfully ${enable ? "enabled" : "disabled"}.`);
            return true;
        } else {
            console.log("Tor configuration attempt failed.");
            return false;
        }
    } catch (error) {
        throw new Error(`Error configuring Tor: ${error.message}`);
    }
}
```

Verification & Connectivity

12. Verify Daemon is Running

```
function verifyDaemon(retries = 3) {
    /**
     * Verifies if the Haveno Daemon is running.
     *
     * @returns {Object} Daemon status response.
     * @throws {Error} If daemon is not running.
     */

    let client;
    while (retries > 0) {
        try {
            client = HavenoClient.connect("localhost", 9999);

            if (!client.isConnected()) {
                throw new Error("Failed to connect to Haveno Daemon.");
            }

            let status = client.daemonService.getStatus();
            console.log("Haveno Daemon is running:", status);
            return status;

        } catch (error) {
            console.error(`Error verifying Haveno Daemon (Retries Left: ${retries - 1}):`, error.message);
            retries--;

            if (retries === 0) {
                throw new Error("Haveno Daemon is not running or unreachable.");
            }
        }
    }
}
```

13. Monero Node Connectivity

```
async function connectToMoneroNode(customNodeURL = "https://node.moneroworld.com:18089", timeout = 5000) {
  /**
   * Connects to a Monero node.
   *
   * @returns {Object} Monero node info.
   * @throws {Error} If connection fails.
   */

  try {
    console.log(`Connecting to Monero Node at ${customNodeURL}...`);

    const controller = new AbortController();
    const timeoutId = setTimeout(() => controller.abort(), timeout);

    const response = await fetch(customNodeURL, {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ method: "get_info" }),
      signal: controller.signal,
    });

    clearTimeout(timeoutId);

    if (!response.ok) {
      throw new Error(`Monero Node Connection Failed: HTTP ${response.status}`);
    }

    return await response.json();
  } catch (error) {
    if (error.name === "AbortError") {
      throw new Error("Monero Node Connection Timeout.");
    }
    throw new Error("Failed to connect to Monero Node.");
  }
}
```