**Technical Specification Document for Tron-Based Trading Platform**

**1. Overview**

**1.1 Project Description**

The project aims to develop a decentralized trading platform on the Tron blockchain. This platform will enable users to trade TRX and TRC20 tokens with low transaction fees, secure fund management, and a user-friendly interface. Integration with Sunpump will enhance liquidity and reduce transaction costs, making the platform attractive to both retail and institutional traders.

**1.2 Goals and Objectives**

* **Primary Goals:**
  + Develop a secure and scalable trading platform on the Tron blockchain.
  + Integrate Sunpump to reduce transaction fees and manage liquidity.
  + Deliver a real-time trading experience with an intuitive user interface.
* **Secondary Goals:**
  + Ensure compatibility with major Tron wallets (TronLink, TronWallet, Math Wallet).
  + Implement advanced security features, including optional two-factor authentication (2FA).
  + Provide responsive customer support and detailed analytics for users.

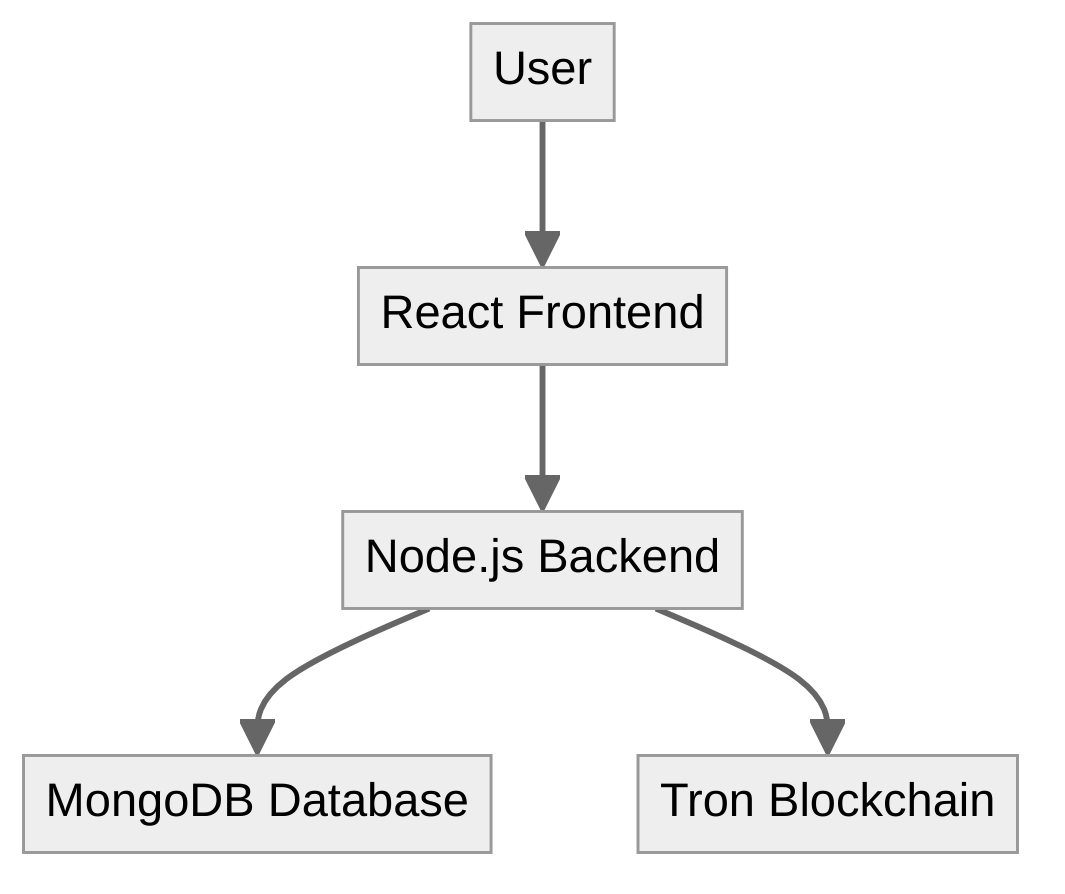
**2. Architecture**

**2.1 System Architecture**

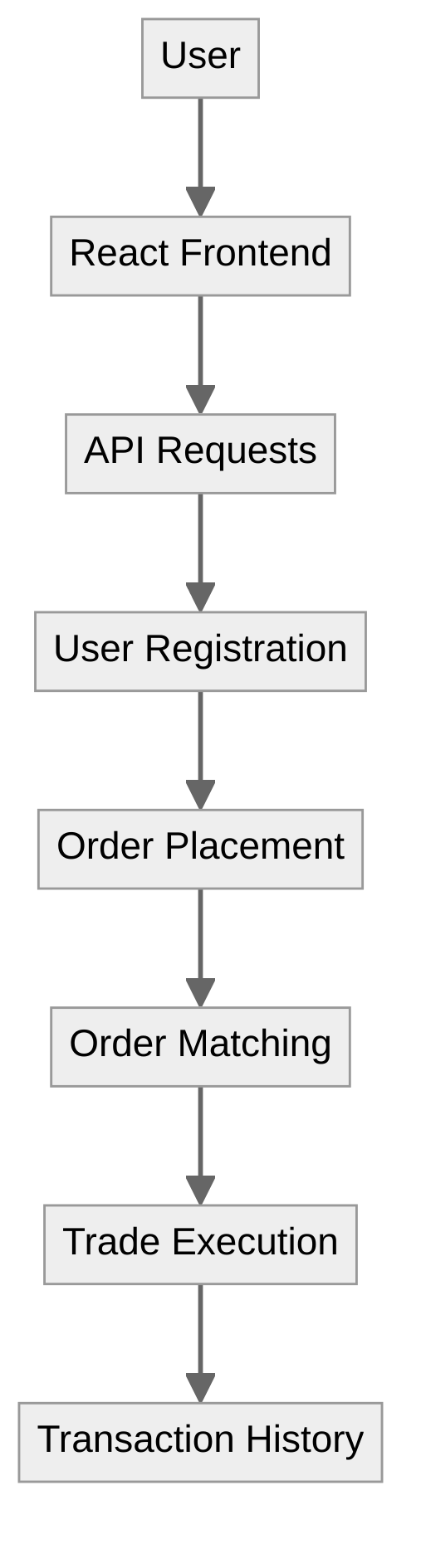
* **Frontend:** The frontend will be developed using React. It will interact with the backend and the Tron blockchain through RESTful APIs and WebSockets for real-time updates.
* **Backend:** The backend will be built using Node.js with Express.js, handling user requests, trade processing, and interactions with the MongoDB database.
* **Blockchain:** Smart contracts deployed on the Tron blockchain will manage trades, token transfers, and Sunpump transactions.
* **Database:** MongoDB will be used to store user data, transaction history, and platform settings.

**Diagrams Needed:**

1. **System Architecture Diagram:**
   * A high-level diagram showing how the frontend, backend, blockchain, and database components interact. This should include:
     + User interaction with the React frontend.
     + API calls from the frontend to the Node.js backend.
     + The backend’s interaction with MongoDB for data storage.
     + The backend’s interaction with the Tron blockchain via TronWeb.



1. **Data Flow Diagram (DFD):**
   * A detailed Level 1 DFD showing the flow of data through processes like user registration, order placement, order matching, trade execution, and transaction history management.

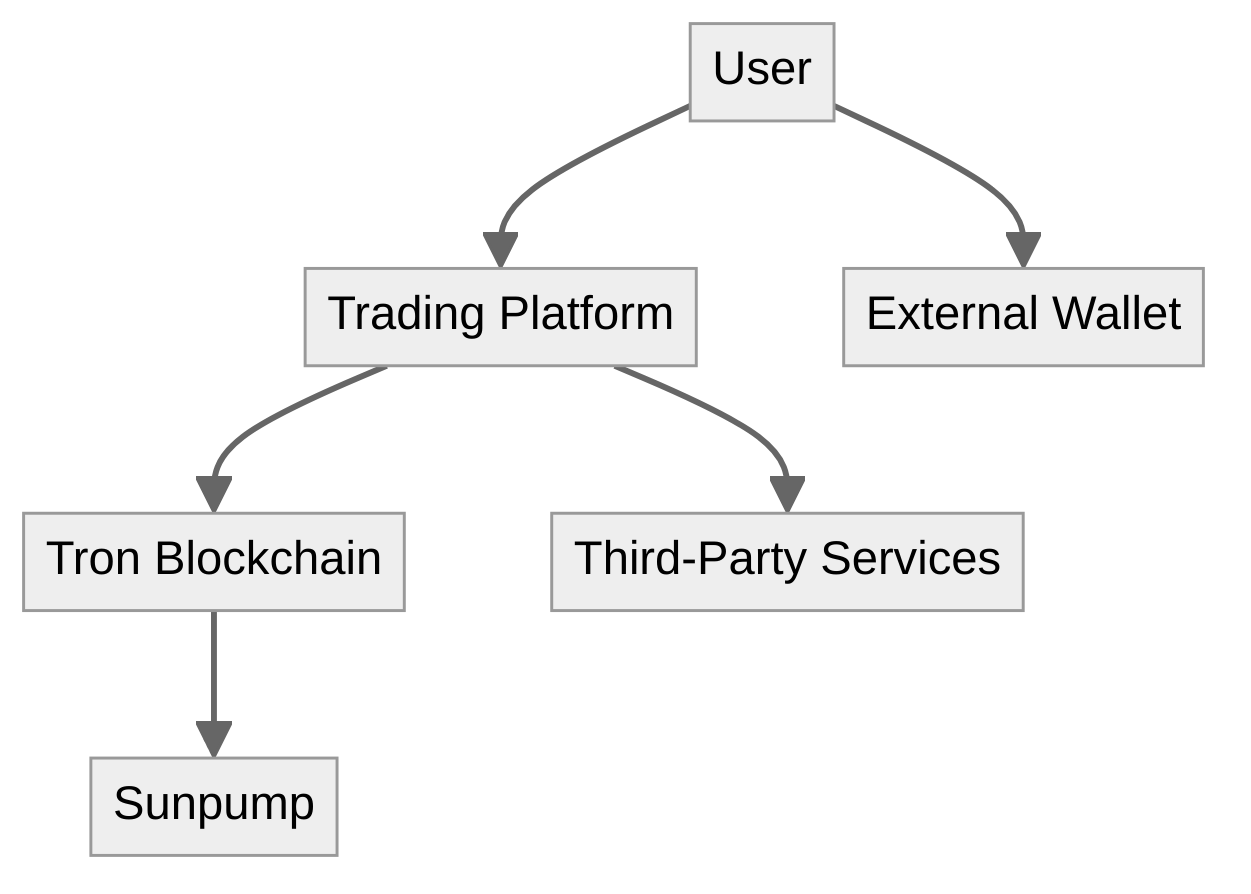


**2.2 Data Flow Diagrams**

* **Context Diagram (Level 0):** This diagram shows the system as a single process interacting with external entities such as users, the Tron blockchain, and third-party services.
* **Detailed DFD (Level 1):** This diagram breaks down the key processes, such as user registration, placing an order, executing a trade, and viewing transaction history, showing how data flows between the frontend, backend, and blockchain.

**Diagrams Needed:**

1. **Context Diagram:**
   * Depicts the system as a black box interacting with external entities (Users, Tron Blockchain, etc.).



1. **Detailed DFD (Level 1):**
   * Shows processes like user authentication, trading, and data storage in detail.

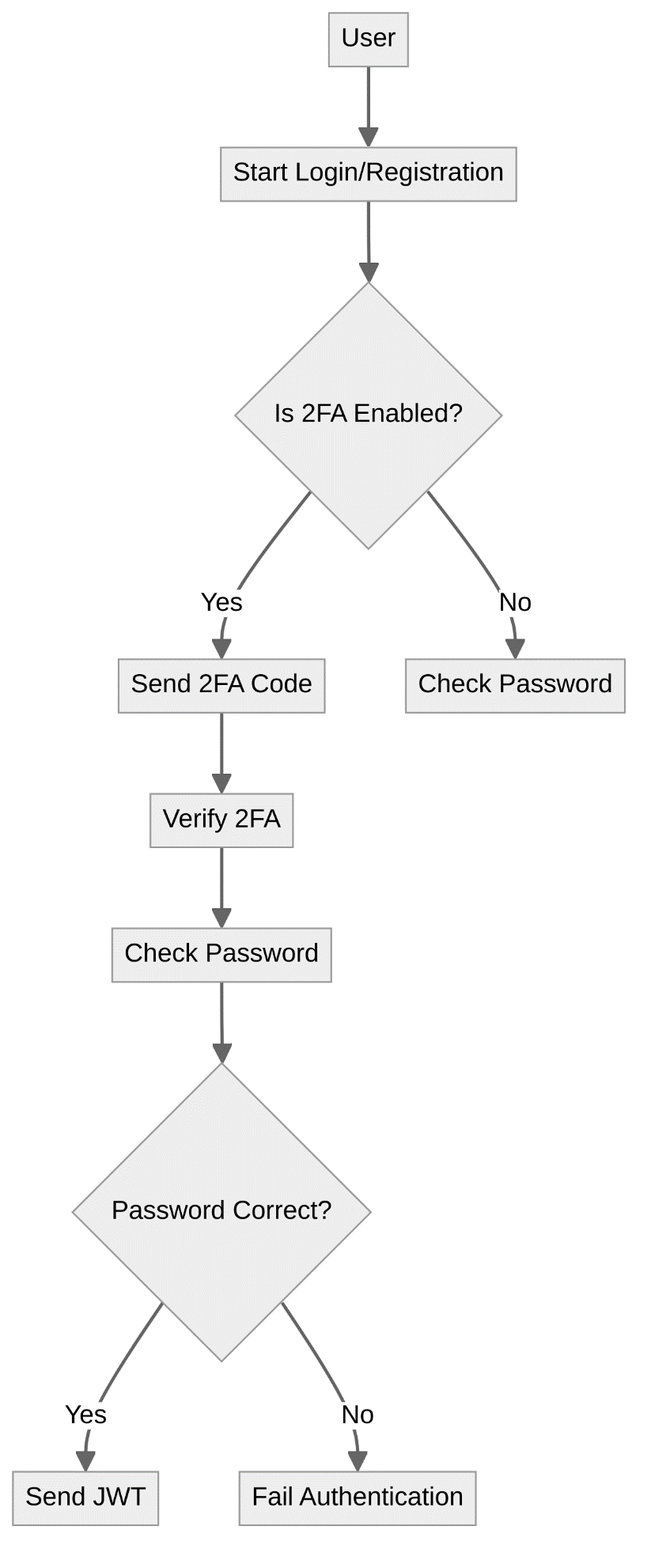
**3. Detailed Features**

**3.1 User Authentication**

* **Sign-Up/Login:** Users can create accounts using email/password or through Tron wallets (TronLink, TronWallet, Math Wallet).
* **Two-Factor Authentication (2FA):** 2FA is optional for users, enhancing security when enabled.
* **Password Recovery:** Password recovery is done securely via email verification.
* **Technical Details:**
  + **Authentication Method:** JSON Web Token (JWT) will be used for managing sessions.
  + **Encryption:** Passwords will be hashed using bcrypt.
  + **Security Protocols:** SSL/TLS will be used for securing data in transit.

**Diagram Needed:**

1. **User Authentication Flowchart:**
   * Illustrates the process of user registration, login, password recovery, and 2FA.

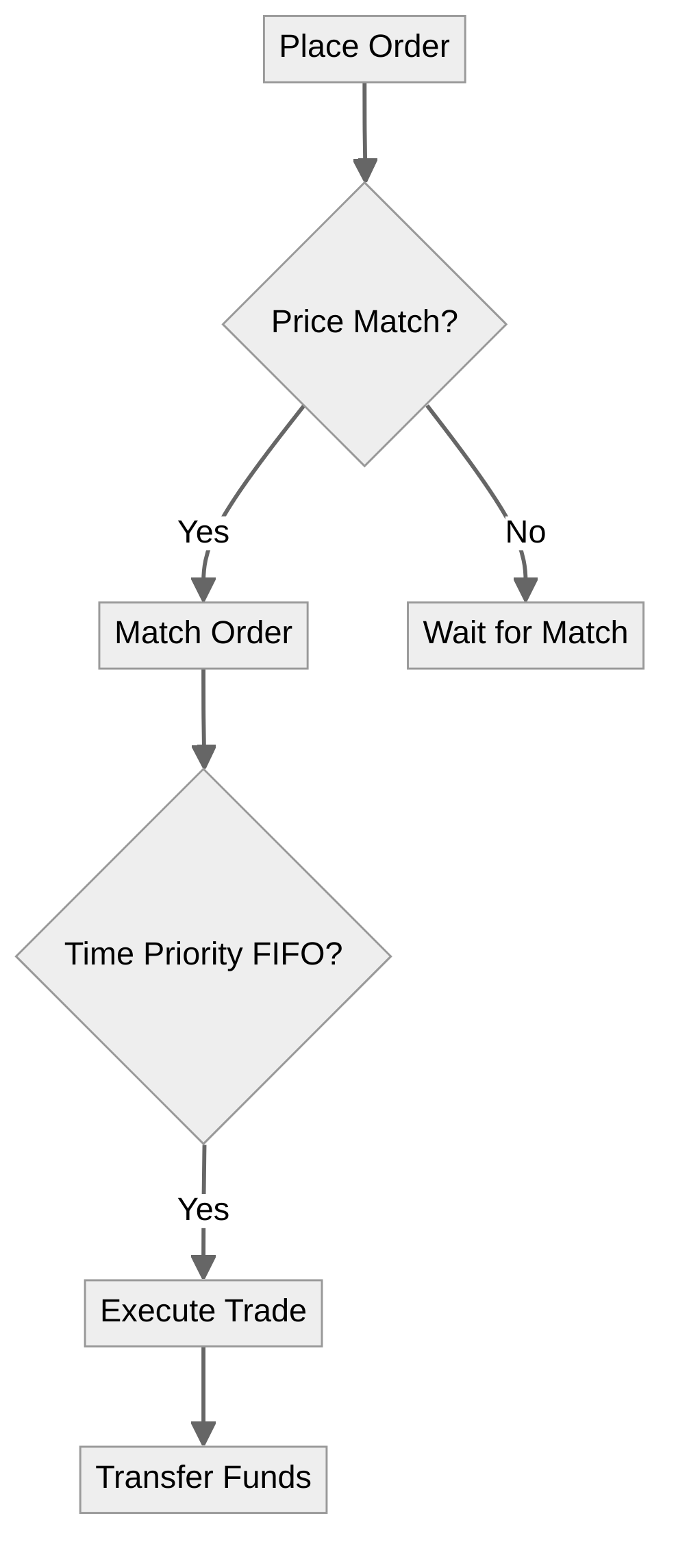


**3.2 Trading Engine**

* **Order Placement:** Users can place buy/sell orders specifying the price and amount.
* **Order Matching:** Orders will be matched based on price and time priority using a FIFO (First In, First Out) algorithm.
* **Trade Execution:** When orders are matched, trades are executed, and funds are transferred automatically.
* **Advanced Trading Options:**
  + **Stop-Loss**
  + **Trailing Stop-Loss**
  + **Limit Orders**
  + **Take Profit**
  + **Trailing Take Profit**
* **Technical Details:**
  + **Matching Algorithm:** FIFO will be used for order matching.
  + **Smart Contracts:** These will handle the logic for placing, matching, and executing trades.

**Diagram Needed:**

1. **Order Matching and Execution Diagram:**
   * Visualizes how orders are placed, matched, and executed, including advanced trading options.

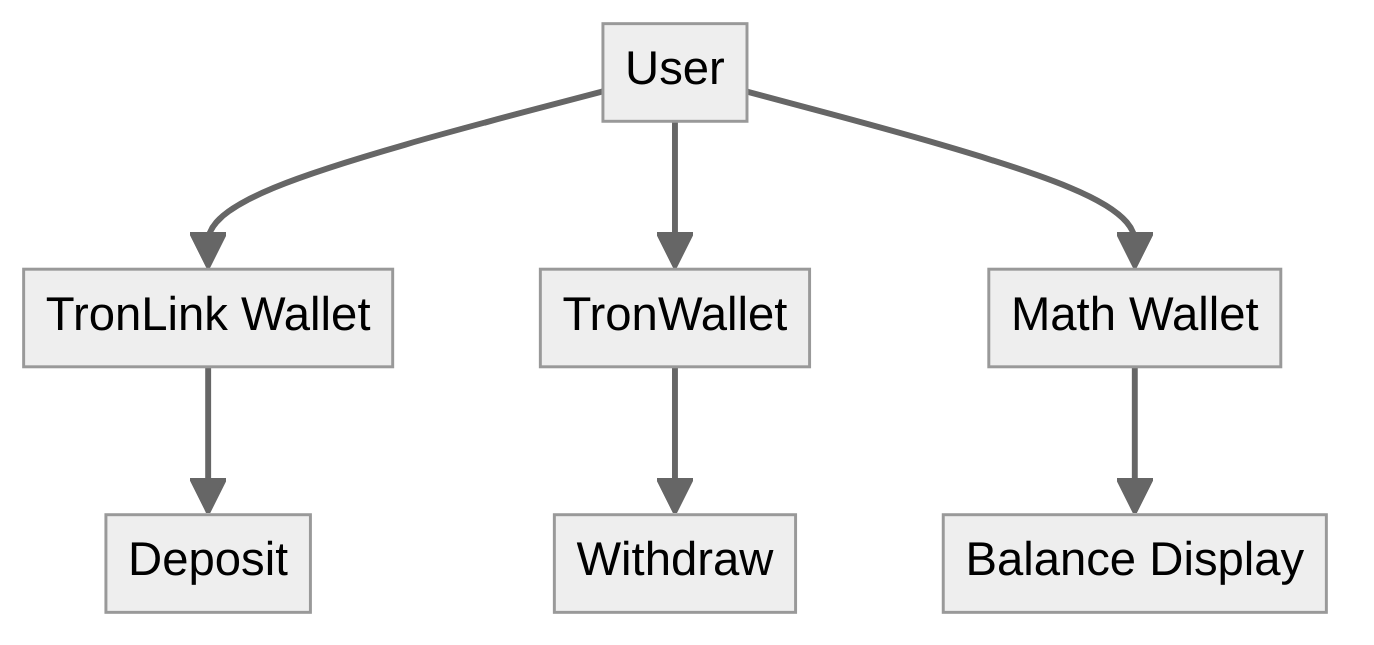


**3.3 Wallet Integration**

* **Supported Wallets:** TronLink, TronWallet, and Math Wallet.
* **TRX and TRC20 Token Support:** Users can deposit, withdraw, and manage TRX and TRC20 tokens.
* **Balance Display:** Real-time wallet balances will be displayed on the dashboard.
* **Transaction History:** A full history of user transactions will be accessible.
* **Technical Details:**
  + **Wallet SDKs:** Integration with TronLink, TronWallet, and Math Wallet.
  + **Security Measures:** Multi-signature transactions for high-value withdrawals will be implemented to enhance security.

**Diagram Needed:**

1. **Wallet Integration Diagram:**
   * Shows how the platform interacts with supported wallets for deposits, withdrawals, and balance display.

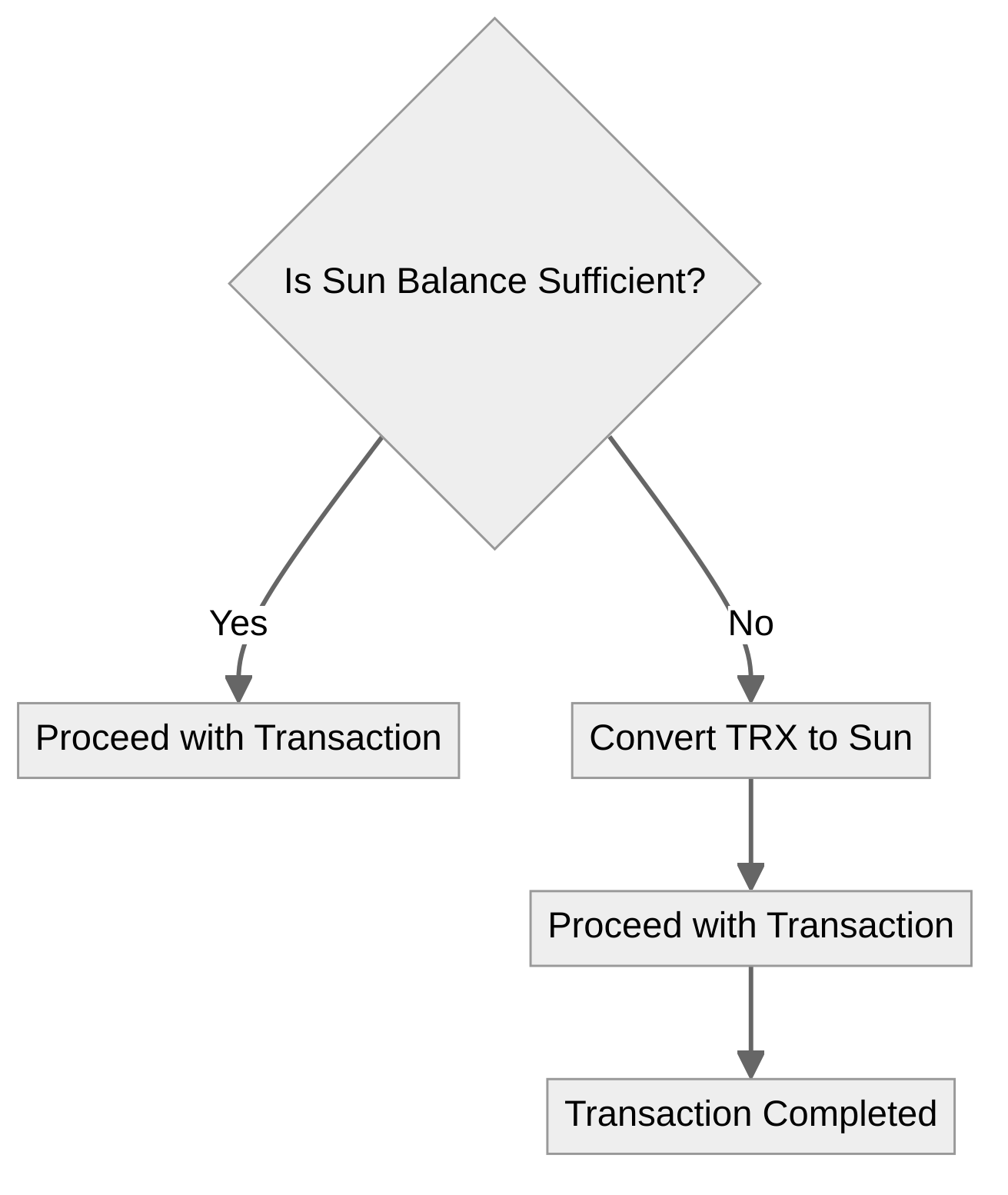


**3.4 Sunpump Integration**

* **Sun-Based Transactions:** The platform will use Sun to reduce transaction fees.
* **Liquidity Management:** Integration with Sunpump will ensure sufficient liquidity for transactions.
* **Automatic TRX-to-Sun Conversion:** If the available Sun balance does not cover the transaction fee, the platform will automatically convert enough TRX to Sun to complete the transaction.
* **Technical Details:**
  + **SDK Integration:** The Sunpump SDK will be used for managing Sun-based transactions.
  + **Fee Structure:** Automatically converting TRX to Sun as needed for fee coverage.

**Diagram Needed:**

1. **Sunpump Integration Flowchart:**
   * Illustrates the process of Sun-based transactions and automatic TRX-to-Sun conversion.

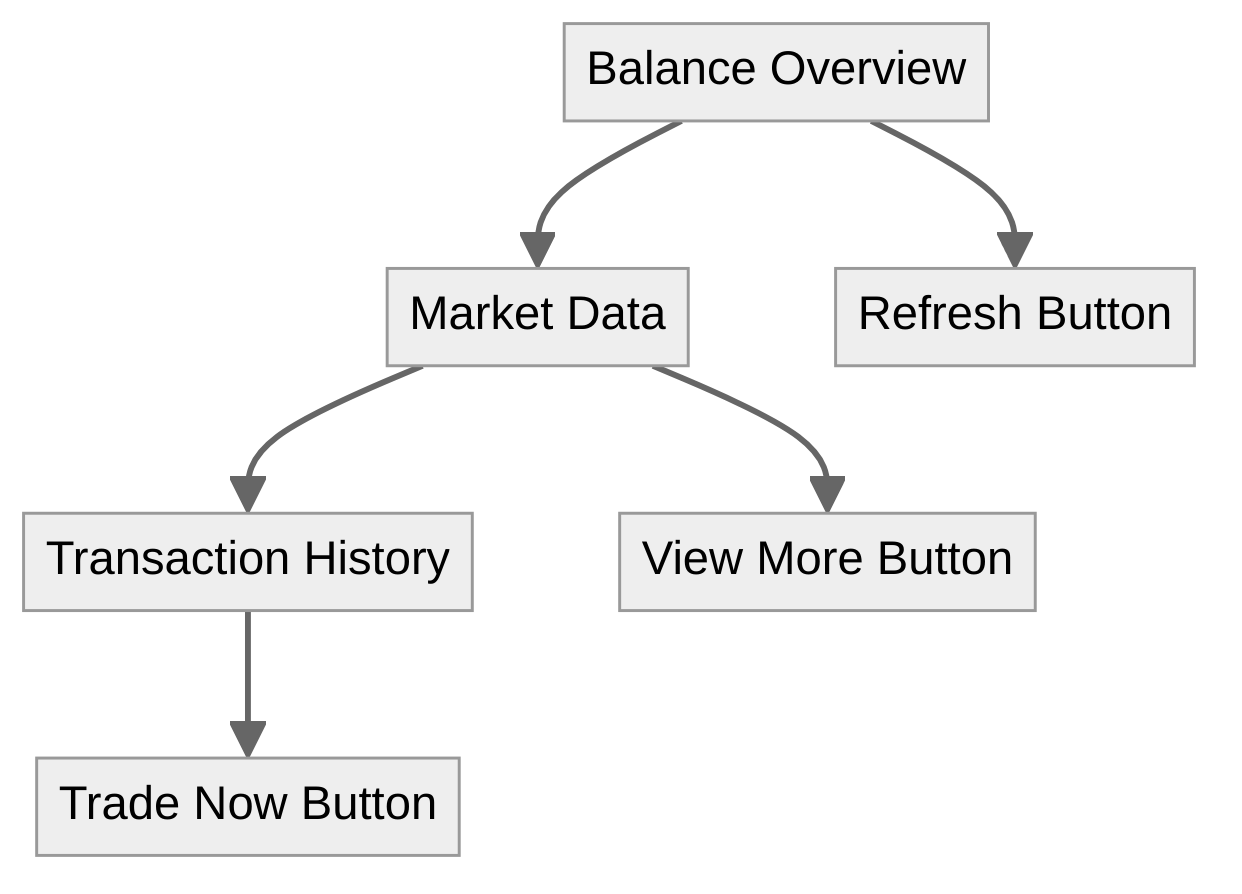


**3.5 Dashboard**

* **Balance Overview:** Displays an overview of TRX, TRC20 tokens, and Sun balances.
* **Market Data:** Shows real-time prices, volumes, and trading pairs.
* **Transaction History:** Provides a detailed history of all user transactions.
* **Technical Details:**
  + **Real-time Updates:** WebSockets will be used for real-time data updates.
  + **Data Visualization:** Libraries like Chart.js will be used for rendering charts and graphs.
  + **Layout:** A standard layout will be used initially, with potential customization options in future updates.

**Diagram Needed:**

1. **Dashboard Wireframe:**
   * A simple wireframe showing the standard layout of the dashboard with balance overview, market data, and transaction history.

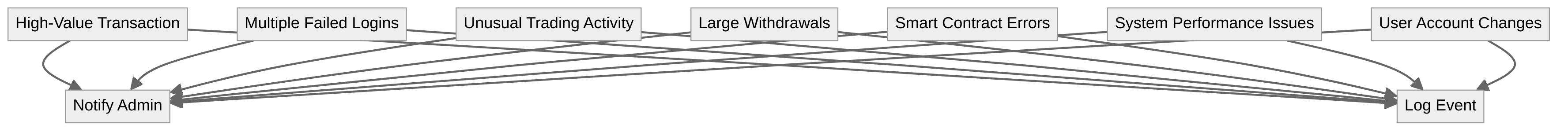


**3.6 Admin Panel**

* **User Management:** Admins can view, block, or delete user accounts.
* **Transaction Monitoring:** Admins will monitor all platform transactions for suspicious activity.
* **System Metrics:** Key metrics such as trade volume and system performance will be displayed.
* **Automated Alerts:** Admins will receive alerts for:
  + High-Value Transactions
  + Multiple Failed Login Attempts
  + Unusual Trading Activity
  + Large Withdrawals
  + Smart Contract Errors
  + System Performance Issues
  + User Account Changes
* **Technical Details:**
  + **Role-Based Access Control (RBAC):** Secure admin functionalities with role-based permissions.
  + **Audit Logs:** Detailed logs of all admin actions will be maintained for security and compliance.

**Diagram Needed:**

1. **Admin Panel Alerts Diagram:**
   * Shows the different triggers for automated alerts and the corresponding admin actions.



**4. Technology Stack**

**4.1 Frontend**

* **Framework:** React will be used for developing the frontend.
* **Tools and Libraries:** Redux for state management, Bootstrap for UI components.

**4.2 Backend**

* **Framework:** Node.js with Express.js for handling API requests and blockchain interactions.
* **Database:** MongoDB will be used for storing user data, transaction history, and platform settings.

**4.3 Blockchain Interaction**

* **Tools:** TronWeb will be used for interacting with the Tron blockchain and smart contracts.
* **Smart Contract Language:** Solidity will be used for writing the smart contracts.

**5. Milestones and Timeline**

**5.1 Milestone 1: Setup and Configuration**

* **Tasks:** Setting up the development environment, configuring tools, initializing repositories.
* **Timeline:** 1-2 weeks.

**5.2 Milestone 2: Smart Contract Development**

* **Tasks:** Writing, testing, and deploying smart contracts for core features like trading, wallet management, and Sunpump integration.
* **Timeline:** 3-4 weeks.

**5.3 Milestone 3: Backend Development**

* **Tasks:** Developing the backend APIs, integrating TronWeb, and connecting to the MongoDB database.
* **Timeline:** 3-4 weeks.

**5.4 Milestone 4: Frontend Development**

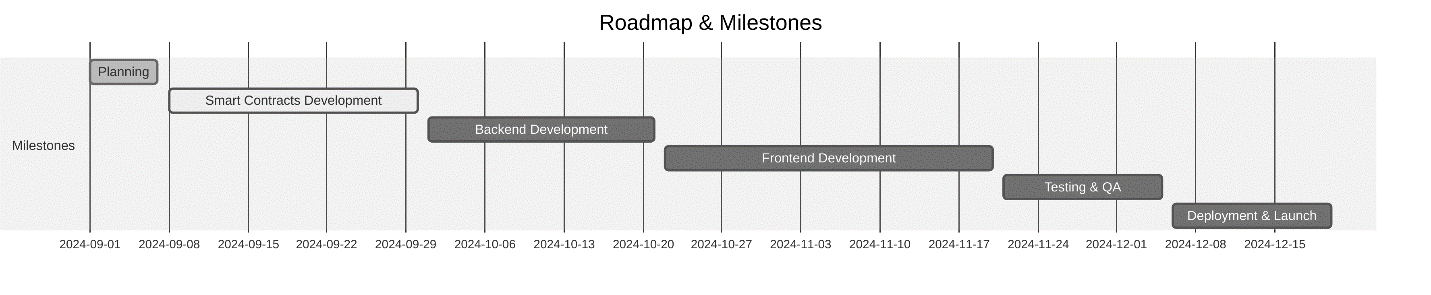
* **Tasks:** Developing the user interface in React, integrating APIs, and implementing real-time data updates.
* **Timeline:** 4-5 weeks.

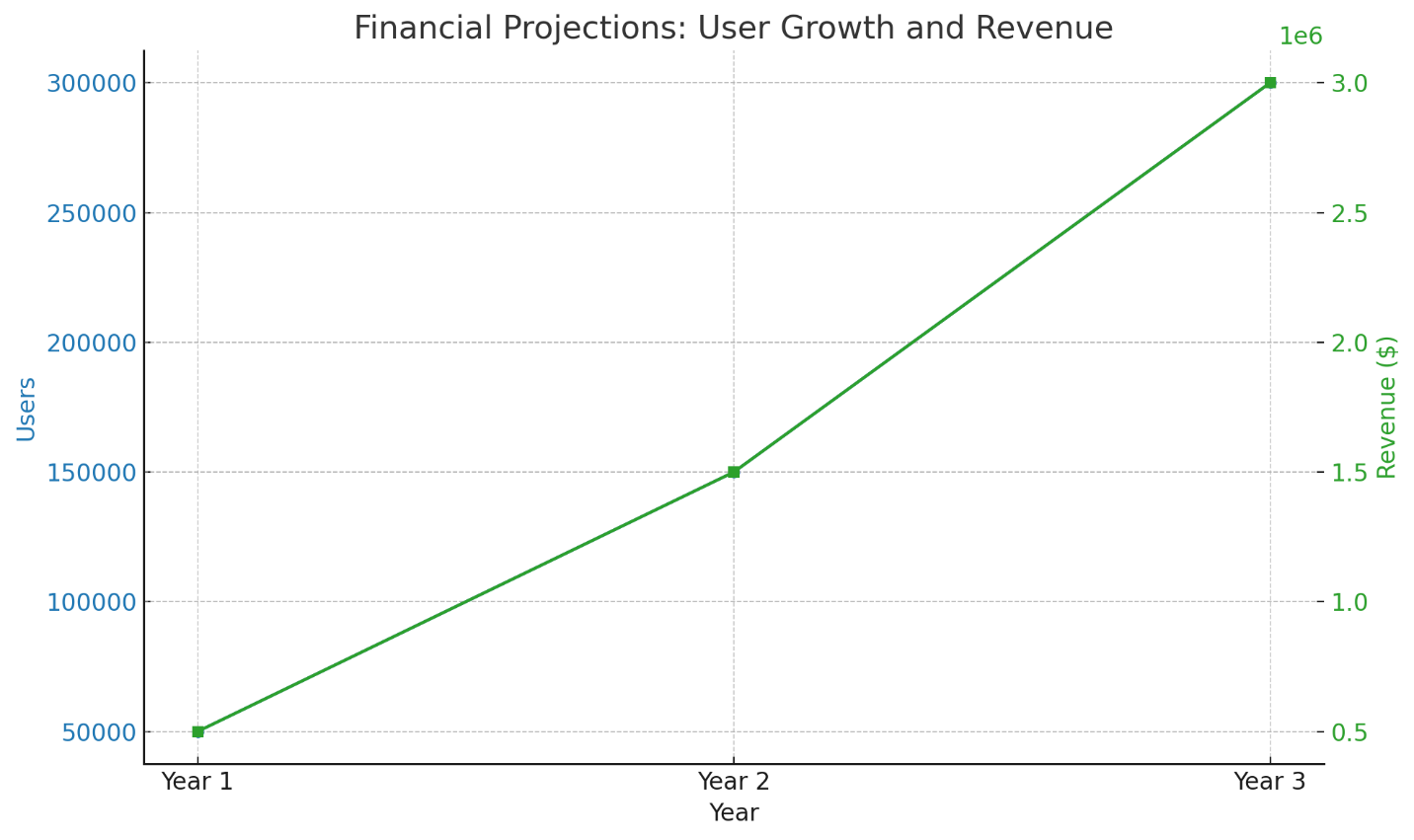
**5.5 Milestone 5: Testing and QA**

* **Tasks:** Conducting thorough testing, including unit tests, integration tests, and user acceptance testing.
* **Timeline:** 2-3 weeks.

**5.6 Milestone 6: Deployment and Launch**

* **Tasks:** Deploying the platform on the Tron mainnet, setting up production servers, and monitoring initial user activity.
* **Timeline:** 1-2 weeks.





I can guide you on how to create the diagrams and charts step-by-step, but currently, I can't directly generate images or diagrams for you. However, here's a detailed guide on how to create each diagram using popular tools like Lucidchart, Draw.io, or similar software:

**1. System Architecture Diagram**

**Tool:** Lucidchart, Draw.io

**Steps:**

1. **Start with the Components:**
   * Draw rectangles or circles to represent the main components: Frontend (React), Backend (Node.js with Express.js), MongoDB, and Tron Blockchain.
2. **Connect the Components:**
   * Use arrows to show the flow of data between the frontend and backend.
   * Connect the backend to the MongoDB database and the Tron Blockchain.
3. **Label the Connections:**
   * Label the arrows to indicate what kind of data is being transferred, e.g., "API Requests," "Database Queries," "Smart Contract Calls."

**2. Data Flow Diagram (DFD)**

**Tool:** Lucidchart, Draw.io

**Steps:**

1. **Identify Processes:**
   * Create processes for key activities like "User Registration," "Order Placement," "Order Matching," and "Trade Execution."
2. **Connect External Entities:**
   * Include external entities like "User" and "Tron Blockchain" and connect them to the relevant processes.
3. **Define Data Stores:**
   * Add data stores like MongoDB for "User Data" and "Transaction History."
4. **Show Data Flow:**
   * Use arrows to depict how data moves between processes, data stores, and external entities.

**3. User Authentication Flowchart**

**Tool:** Lucidchart, Draw.io

**Steps:**

1. **Create Start/End Points:**
   * Add a start point for when the user initiates the login or registration process.
2. **Include Decision Points:**
   * Add decision points for actions like "Is 2FA enabled?" or "Is the password correct?"
3. **Flow Actions:**
   * Show actions like "Send JWT," "Hash Password with bcrypt," or "Send Email Verification."
4. **End Points:**
   * Add end points for successful login/registration or failure scenarios.

**4. Order Matching and Execution Diagram**

**Tool:** Lucidchart, Draw.io

**Steps:**

1. **List the Key Actions:**
   * Create nodes for actions like "Place Order," "Match Order," and "Execute Trade."
2. **Connect with Arrows:**
   * Use arrows to indicate the sequence of actions from order placement to execution.
3. **Include Conditions:**
   * Show where conditions like "Price Match" or "Time Priority (FIFO)" affect the flow.

**5. Wallet Integration Diagram**

**Tool:** Lucidchart, Draw.io

**Steps:**

1. **Identify Wallets:**
   * Include icons or shapes for TronLink, TronWallet, and Math Wallet.
2. **Connect to Platform:**
   * Use arrows to show how each wallet interacts with the platform for "Deposit," "Withdrawal," and "Balance Display."
3. **Label Actions:**
   * Clearly label actions such as "Fetch Balance," "Transfer TRX," and "Authenticate User."

**6. Sunpump Integration Flowchart**

**Tool:** Lucidchart, Draw.io

**Steps:**

1. **Start with a Decision Point:**
   * Create a decision node asking, "Is Sun balance sufficient?"
2. **Add Actions:**
   * Show actions like "Convert TRX to Sun" if balance is insufficient, or "Proceed with Transaction" if it is sufficient.
3. **End Point:**
   * Conclude with "Transaction Completed."

**7. Dashboard Wireframe**

**Tool:** Figma, Sketch, or Draw.io

**Steps:**

1. **Create Sections:**
   * Divide the screen into sections for "Balance Overview," "Market Data," and "Transaction History."
2. **Add Widgets:**
   * Mock up widgets for balance display, real-time price charts, and transaction lists.
3. **Include Interaction Points:**
   * Add placeholders for buttons like "Trade Now" or "View More."

**8. Admin Panel Alerts Diagram**

**Tool:** Lucidchart, Draw.io

**Steps:**

1. **List Alert Triggers:**
   * Create nodes for each trigger like "High-Value Transaction" and "Multiple Failed Logins."
2. **Connect to Admin Actions:**
   * Use arrows to show the flow from trigger detection to "Notify Admin" or "Log Event."
3. **Label Connections:**
   * Clearly label each connection with the type of alert or admin response.

**Tools for Diagram Creation:**

* **Lucidchart:** <https://www.lucidchart.com/>
* **Draw.io (Diagrams.net):** https://app.diagrams.net/
* **Figma:** <https://www.figma.com/>
* **Sketch:** <https://www.sketch.com/>

These steps will help you create the diagrams that visually represent your platform’s architecture and workflows. If you need help with specific parts or have any other questions, feel free to ask!