**Product Requirement Document: AI-Powered Remittance Agent ("AgentPay")**

**1. Introduction & Vision**

1.1. Problem Statement

Cross-border remittances are plagued by inefficiencies. Users face high transaction fees, unfavorable and often opaque exchange rates, slow settlement times, and complex user experiences. Traditional systems are slow, while existing crypto-based solutions can be intimidating for non-technical users due to wallet management, price volatility, and security concerns. There is a clear market need for a solution that combines the speed and low cost of digital assets with the simplicity and trust of modern FinTech applications.

1.2. Vision

Our vision is to create the most intelligent, efficient, and user-friendly cross-border payment platform. By leveraging an AI agent, we will abstract away the complexities of blockchain technology, offering users a simple conversational interface to send money internationally. "AgentPay" will be the "Web3-native Wise," providing near-instant, low-cost, and fully transparent remittances powered by a secure and scalable distributed ledger.

**2. Goals & Objectives**

**2.1. Business Goals**

* **Disrupt the Market:** Capture market share from incumbents by offering significantly lower fees and faster settlement times.
* **Establish a Compliance-Forward Model:** Build a platform that is transparent and auditable by default, simplifying regulatory adherence.
* **Achieve Scalability:** Build on infrastructure capable of handling enterprise-level transaction volume (>10,000 TPS).

**2.2. User Goals**

* **Simplicity:** Send money internationally as easily as sending a text message.
* **Speed:** Have funds delivered to the recipient in seconds, not days.
* **Cost-Effectiveness:** Pay minimal, transparent fees with no hidden exchange rate markups.
* **Trust & Security:** Feel confident that funds are secure and the transaction is compliant with regulations.

**3. User Personas**

**3.1. Primary Persona: The Sender (Priya, the Freelancer)**

* **Bio:** Priya is a 28-year-old graphic designer based in the U.S. who works with international clients and sends money home to her family in India every month.
* **Needs & Goals:** She needs a fast, reliable, and inexpensive way to send money. She is tech-savvy but finds traditional crypto wallets and exchanges complex and risky. She values transparency and wants to know exactly how much the recipient will get.
* **Pain Points:** Frustrated with the high percentage-based fees of services like Western Union and the multi-day settlement times of bank wires.

**3.2. Secondary Persona: The Recipient (Anil, the Parent)**

* **Bio:** Anil is Priya's 60-year-old father living in India. He has a smartphone and a local bank account but is not technologically sophisticated.
* **Needs & Goals:** He needs to receive money from Priya reliably and directly into his bank account without any complicated steps.
* **Pain Points:** Has experienced delays in receiving funds and finds it difficult to track incoming transfers from traditional services.

**4. Features & Functional Requirements (MVP Scope)**

The MVP will focus on demonstrating the core end-to-end flow on a test network.

| **Epic** | **User Story** | **Functional Requirements** |
| --- | --- | --- |
| **Agent Interaction** | As Priya, I want to instruct an AI agent to send money using a simple chat command so that the process is fast and intuitive. | - A web-based chat interface for user input. - The AI agent must parse natural language to identify intent (send), amount, currency (stablecoin), and recipient address. - The agent must provide a confirmation prompt before executing (e.g., "You want to send 100 PYUSD to 0.0.XXXX. Confirm?"). |
| **Transaction Execution** | As Priya, I want the agent to securely and quickly execute the transfer on the blockchain once I confirm. | - The agent will use the Hedera JavaScript SDK to construct a Hedera Token Service (HTS) transfer transaction.  - The transaction will transfer a pre-configured HTS token (representing PYUSD) on the Hedera Testnet.  - The agent must sign the transaction using the sender's pre-configured private key.  - The agent must submit the transaction to the Hedera Testnet and wait for finality. |
| **Notifications** | As Priya, I want to receive a confirmation that the transfer was successful, including a transaction ID for my records. | - Upon successful transaction confirmation, the agent must return a success message to the chat interface. - The message must include the Hedera transaction ID, which can be verified on a public explorer like HashScan. |
| **Compliance (Mocked)** | As a platform operator, I need to ensure all transactions are compliant. | - For the MVP, KYC/AML is assumed. The agent will proceed as if both sender and receiver accounts are pre-verified.  - The agent will log key actions (e.g., "Initiating transfer," "Transaction confirmed") to the Hedera Consensus Service (HCS) to simulate an auditable trail.5 |

**5. Non-Functional Requirements**

* **Performance:** Transaction finality on the Hedera Testnet should be achieved in ~3-5 seconds.
* **Security:** Private keys for testnet accounts must be stored securely in environment variables and never exposed on the client side.
* **Usability:** The chat interface must be clean, simple, and provide clear feedback to the user at every step.
* **Reliability:** The system should handle basic errors gracefully (e.g., invalid account ID format, insufficient testnet funds).

**6. System Architecture & Tech Stack**

* **Frontend:** Simple React or Next.js web application providing the chat UI.
* **Backend:** Node.js server with an API endpoint to receive chat messages.
* **AI Agent Layer:** An AI agent built using a framework like LangChain and the **Hedera Agent Kit**. The agent will be equipped with "tools" that wrap Hedera SDK functions.
* **Distributed Ledger:** Hedera Testnet. Transactions will utilize the Hedera Token Service (HTS) and Hedera Consensus Service (HCS).7
* **Settlement Asset:** A fungible token created on the Hedera Testnet to represent PayPal USD (PYUSD).

**7. Assumptions & Dependencies**

* **Testnet Focus:** The MVP will operate exclusively on the Hedera Testnet. No real funds will be used.
* **Pre-Configured Accounts:** Users (sender/receiver) will use pre-existing Hedera testnet accounts. The MVP will not include an onboarding or wallet creation flow.
* **Mocked Compliance & Fiat Rails:** KYC/AML checks and fiat on/off-ramps are out of scope and will be simulated in the product narrative only.
* **Technology Availability:** The project depends on the continued availability and functionality of the Hedera Testnet, Hedera SDKs, and the Hedera Agent Kit.

**5-Day MVP Development Plan**

This is an aggressive plan for a single developer or a small, focused team to build a proof-of-concept. The goal is to demonstrate the core technical feasibility.

**Pre-Work: Environment Setup**

* Install Node.js, npm/yarn.
* Set up a code editor (e.g., VS Code).
* Create a GitHub repository for the project.

**Day 1: Foundation & Hedera Setup**

**Objective:** Establish the project structure and secure testnet accounts and assets.

**Tasks:**

1. **Project Scaffolding:**
   * Initialize a new Next.js project for the frontend.
   * Create a server directory for the Node.js backend.
   * Initialize a package.json for the backend and install initial dependencies (express, dotenv, @hashgraph/sdk).
2. **Hedera Testnet Accounts:**
   * Go to the Hedera Portal to create two new testnet accounts: one for the Sender (Priya) and one for the Recipient (Anil).
   * Securely save the Account IDs and Private Keys in a .env file in the server directory.
   * Use the Hedera Faucet to fund both accounts with test HBAR.
3. **Create a Test Stablecoin:**
   * Write and run a simple Node.js script using the Hedera SDK (TokenCreateTransaction) to create a new fungible token on the testnet.8 Name it "Test PYUSD" (TPYUSD).
   * Mint an initial supply to the Sender's account.
   * Associate the token with the Recipient's account so they can receive it.
   * Save the Token ID in the .env file.

**Deliverable:** A functional project structure with two funded Hedera testnet accounts, one of which holds a custom HTS token.

**Day 2: Core Transaction Logic**

**Objective:** Build and validate the backend service for executing a token transfer.

**Tasks:**

1. **Create a Hedera Service Module:**
   * In the backend, create a hederaService.js file.
   * Write a function transferTokens(senderId, senderKey, receiverId, tokenId, amount) that:
     + Initializes the Hedera client.
     + Uses TransferTransaction() to construct the HTS token transfer.4
     + Signs the transaction with the sender's private key.9
     + Executes the transaction and waits for the receipt.
     + Returns the transaction ID on success or throws an error on failure.
2. **Build a Test Script:**
   * Create a testTransfer.js script that imports transferTokens and executes a hardcoded transfer between the two testnet accounts.
3. **Verify on Explorer:**
   * Run the test script and confirm the transaction is successful.
   * Use the returned transaction ID to look up the transaction on HashScan to verify the token transfer.

**Deliverable:** A reliable backend function that can successfully transfer HTS tokens on the Hedera Testnet.

**Day 3: AI Agent Integration**

**Objective:** Create an AI agent that can understand a command and use the Hedera service tool.

**Tasks:**

1. **Set Up Agent Environment:**
   * Install LangChain and the Hedera Agent Kit (hedera-agent-kit, @langchain/openai, etc.) in the backend.10
   * Add your OpenAI (or other LLM provider) API key to the .env file.
2. **Define Agent Tools:**
   * Using the Hedera Agent Kit documentation, create a "tool" for the agent that wraps the transferTokens function created on Day 2.5
   * The tool's description should be clear for the LLM, e.g., "Use this tool to transfer a specified amount of a token from a sender to a receiver on the Hedera network."
3. **Implement the Agent Logic:**
   * Create an agentService.js file.
   * Write a function processMessage(message) that takes the user's text, passes it to the LangChain agent executor, and invokes the appropriate tool.
   * The prompt for the agent should instruct it to parse the user's request and use the transfer tool with the correct parameters.

**Deliverable:** An agent that can be called programmatically to parse a string like "send 10 TPYUSD to 0.0.5678" and execute the corresponding transferTokens function.

**Day 4: Frontend Chat UI & API**

**Objective:** Build the user-facing interface and connect it to the backend.

**Tasks:**

1. **Build the Chat Component:**
   * In the Next.js frontend, create a simple chat component with a message history display and a text input field.
   * Manage the chat state (list of messages) using React's useState.
2. **Create the Backend API Endpoint:**
   * In the Node.js server, create a /api/chat endpoint that accepts a POST request with the user's message.
   * This endpoint will call the agentService.processMessage() function from Day 3.
   * It should wait for the agent's execution to complete and return a JSON response with the success message and transaction ID (or an error).
3. **Connect Frontend to Backend:**
   * Implement the onSubmit handler in the chat component to make a fetch call to the /api/chat endpoint.
   * When the response is received, add both the user's message and the agent's response to the chat history.

**Deliverable:** A web page where a user can type a message, send it to the backend, and see the agent's response appear in the chat.

**Day 5: End-to-End Testing, Polish & Demo Prep**

**Objective:** Ensure the full workflow is functional and prepare for demonstration.

**Tasks:**

1. **Full Flow Testing:**
   * Conduct at least 10 end-to-end tests. Type a transfer command into the UI, verify the agent's response in the UI, and check the transaction on HashScan.
   * Test edge cases: What happens if the user's message is ambiguous? What if the account ID is malformed? (The agent should respond with a clarifying question or an error).
2. **UI/UX Polish:**
   * Add loading indicators to the UI while the backend is processing the transaction.
   * Format the success message to be user-friendly and include a clickable link to the transaction on HashScan.
   * Add a simple header and introductory text explaining what the PoC does.
3. **Code Cleanup & Documentation:**
   * Add comments to the code, especially in the service and agent files.
   * Create a README.md file explaining the project, how to set it up (install dependencies, create .env file), and how to run it.
4. **Prepare for Demo:**
   * Record a short video or prepare a live demonstration of the end-to-end flow.

**Deliverable:** A completed, functional proof-of-concept demonstrating the core value proposition, ready for a stakeholder demo.