Below is a detailed project specification for the implementation of the "Zero Loss Lottery" system based on the requirements provided. This document outlines the project's scope, features, technical stack, implementation details, and an expanded description of the website UI and required components. The system is tailored for a hackathon demonstration, leveraging the Sepolia Ethereum testnet.

Project Title: NoRiskPot

Project Description

The Zero Loss Lottery (NoRiskPot) is a blockchain-based lottery system where participants purchase tickets using PYUSD (a stablecoin) and can win prizes without losing their initial investment. Unlike traditional lotteries, all participants are refunded their ticket purchase amount, and prizes are funded exclusively from interest manually added by the admin. This project is designed for a hackathon demo, using the Sepolia testnet, a frontend-generated random seed for winner selection, and a manual interest model without DeFi integration.

Core Features

- Ticket Purchasing:
    - Users can purchase a maximum of 10 tickets per person, with each ticket priced at 1 PYUSD.
    - Ticket purchases are executed through a smart contract, transferring PYUSD from the user's wallet to the contract.
- Interest Model:
    - The admin manually adds interest to the lottery pool using predefined buttons:
        - "Add 10% Interest" (simulating a monthly rate).
        - "Add 2.5% Interest" (simulating a weekly rate).
    - Interest is calculated as a percentage of the total pool size (total tickets sold × 1 PYUSD) and added by the admin.
- Winner Selection:
    - The admin manually triggers the draw by submitting a random seed generated by the frontend (not cryptographically secure, suitable only for demo purposes).
    - Two winners are selected pseudo-randomly using the seed:

        - 1st Place: Receives 50% of the interest pool.
        - 2nd Place: Receives 30% of the interest pool.
    - The remaining 20% of the interest goes to the platform owner.
- Refunds and Prizes:
    - After the draw, all participants can claim their original ticket purchase amount (1 PYUSD per ticket).
    - Winners additionally receive their respective shares of the interest as prizes.
- Admin Controls:
    - The admin can add interest and trigger the draw manually at any time, bypassing any timer for demo flexibility.

Technical Stack

- Blockchain: Sepolia (Ethereum testnet).
- Smart Contract:
    - Language: Solidity.
    - Development Framework: Hardhat.
- Stablecoin: PYUSD (sourced from an existing faucet for testing).
- Frontend:
    - Framework: React.js.
    - Blockchain Interaction: ethers.js.
    - Wallet Integration: MetaMask.
- RPC Provider: Google Cloud Platform's Blockchain RPC for Sepolia.
- Testing:
    - Unit tests with Hardhat.
    - Integration tests on Sepolia testnet.
- Deployment:
    - Smart contract deployed to Sepolia testnet.
    - Frontend hosted on IPFS or Vercel.

Website UI and Components

The frontend is a single-page application (SPA) built with React.js, offering an intuitive and visually appealing interface for both participants and the admin. Below is a detailed breakdown of the UI sections and their required components, expanded for clarity:

1. Home Page

- Purpose: Serves as the main dashboard, providing a snapshot of the lottery's status and user-specific information.
- Components:
    - Lottery Pool Card:
        - Displays real-time data:
            - Total Tickets Sold: Number of tickets purchased across all users.
            - Total Pool Size: Total PYUSD collected (e.g., "50 PYUSD").
            - Current Interest Pool: Total interest added by the admin (e.g., "5 PYUSD").
            - Draw Status: Indicates "Draw Pending" or "Draw Completed" with a visual badge (e.g., green for pending, red for completed).
    - User Dashboard:
        - Shows personalized data for the connected wallet:
            - Tickets Purchased: Number of tickets bought by the user (e.g., "3/10").
            - Buy More Tickets Button: Enabled if the user has purchased fewer than 10 tickets; links to the "Buy Tickets" page.
    - Navigation Bar:
        - Links to:
            - "Buy Tickets" page.
            - "Admin Panel" (visible only to the admin's wallet address).
            - "Claim Funds" page (enabled post-draw).
        - Styled with a clean, modern design (e.g., fixed top bar with hover effects).

2. Buy Tickets Page

- Purpose: Facilitates ticket purchases with a simple, user-friendly workflow.
- Components:
    - Ticket Quantity Selector:
        - Input field or dropdown to select the number of tickets (1–10).
        - Validates against the user's current ticket count (e.g., disables options exceeding the 10-ticket limit).
    - PYUSD Approval Button:
        - Prompts the user to approve the smart contract to spend PYUSD if not already done.
        - Displays "Approval Pending" during the transaction and "Approved" upon success.

    - Buy Tickets Button:
        - Triggers the `buyTickets` function on the smart contract.
        - Disabled until approval is complete and sufficient PYUSD balance is confirmed.
    - Feedback Banner:
        - Displays transaction status:
            - Success: "Successfully purchased 3 tickets!"
            - Error: "Insufficient PYUSD balance" or "Transaction failed."
        - Includes a loading spinner during processing.

3. Admin Panel (Admin-Only Access)

- Purpose: Provides the admin with tools to manage the lottery, restricted to the contract owner's wallet address.
- Components:
    - Interest Addition Controls:
        - 10% Interest Button: Calls `addInterest(10)` and transfers the calculated amount from the admin's wallet.
        - 2.5% Interest Button: Calls `addInterest(2)` and transfers the calculated amount.
        - Each button displays a confirmation prompt (e.g., "Add 5 PYUSD to the pool?") before execution.
    - Draw Trigger Section:
        - Trigger Draw Button: Generates a random seed (e.g., via `Math.random()`) and calls `drawWinners` with the seed.
        - Disabled after the draw is completed; shows "Draw Already Completed" if clicked again.
    - Admin Status Panel:
        - Displays:
            - Current total pool size.
            - Interest pool size.
            - Draw completion status.
        - Updates dynamically after each admin action.

4. Claim Funds Page

- Purpose: Allows users to retrieve their refunds and prizes after the draw.
- Components:
    - User Claim Summary:
        - Shows:
            - Tickets Purchased: Number of tickets (e.g., "3 tickets").

        - Refund Amount: Total PYUSD to be refunded (e.g., "3 PYUSD").
        - Prize Amount: If a winner, displays the prize (e.g., "2.5 PYUSD for 1st Place").
        - Status: "Claim Available" or "Already Claimed."
    - Claim Funds Button:
        - Calls the `claimFunds` function to transfer refunds and prizes to the user's wallet.
        - Disabled if funds have already been claimed or the draw is pending.
    - Transaction Feedback:
        - Shows a loading state during the claim process, followed by:
            - Success: "Claimed 3 PYUSD refund + 2.5 PYUSD prize!"
            - Error: "Claim failed. Please try again."

5. Wallet Connection Module

- Purpose: Manages wallet integration and network selection across all pages.
- Components:
    - Connect Wallet Button:
        - Located in the navigation bar; prompts MetaMask connection.
        - Switches the network to Sepolia testnet if not already selected.
    - Wallet Info Display:
        - Shows the connected wallet's truncated address (e.g., "0x123…abcd").
        - Displays the user's PYUSD balance, fetched via ethers.js.
    - Network Warning:
        - Alerts the user if connected to the wrong network (e.g., "Please switch to Sepolia testnet").

Additional UI Details

- Responsive Design: Optimized for desktop and mobile devices using own CSS. No library or framework is used. Add animation on the button and other required components of the website.
- Visual Styling: Clean, modern aesthetic with a lottery-themed color scheme (e.g., gold for prizes, blue for trust).
- Real-Time Updates: Uses ethers.js event listeners to refresh pool data and user status automatically.

Implementation Details

Smart Contract: `ZeroLossLottery`

- State Variables:
  - `address public owner`: Contract deployer (admin).
  - `IERC20 public pyusd`: PYUSD token contract address (provided by the faucet).
  - `uint256 public ticketPrice`: 1 PYUSD (1e18 wei, assuming 18 decimals).
  - `uint256 public maxTicketsPerUser`: 10.
  - `mapping(address => uint256) public tickets`: Tracks tickets per user.
  - `address[] public participants`: Array of unique ticket buyers.
  - `uint256 public totalTickets`: Total tickets sold.
  - `uint256 public interestPool`: Total interest added.
  - `address[2] public winners`: Stores the two winners' addresses.
  - `bool public drawCompleted`: Tracks draw status.
- Key Functions:
  - `buyTickets(uint256 numTickets)`:
    - Requires `numTickets ≤ maxTicketsPerUser - tickets[msg.sender]`.
    - Transfers `numTickets * ticketPrice` PYUSD to the contract.
    - Updates `tickets` and `participants`.
  - `addInterest(uint256 percentage)` (admin only):
    - Calculates interest as `(totalTickets * ticketPrice * percentage) / 100`.
    - Transfers the amount from the admin's wallet to `interestPool`.
  - `drawWinners(uint256 seed)` (admin only):
    - Uses the seed to pseudo-randomly select two unique winners from `participants`.
    - Stores winners and sets `drawCompleted = true`.
  - `claimFunds()`:
    - Refunds `tickets[msg.sender] * ticketPrice` to the user.
    - Transfers prizes to winners (50% or 30% of `interestPool`).
    - Prevents double-claiming.

Frontend Integration

- ethers.js: Facilitates contract interaction (e.g., calling `buyTickets`, fetching `totalTickets`).
- MetaMask: Handles transaction signing and network selection.
- React Features:
  - State Management: `useState` for ticket counts, pool size, and user status.
  - Effects: `useEffect` to listen for contract events and update the UI.

    - Conditional Rendering: Shows admin controls or claim options based on wallet address and draw status.

Testing and Deployment

- Unit Testing:
    - Hardhat scripts to test ticket purchases, interest addition, winner selection, and fund claims.
- Integration Testing:
    - Deploy to Sepolia and simulate a full lottery cycle: ticket sales, interest addition, draw, and claims.
- Deployment:
    - Smart contract on Sepolia testnet.
    - Frontend hosted on IPFS for decentralization or Vercel for simplicity.

Demo Flow

1. User Participation:
    - Connect MetaMask, acquire PYUSD from the faucet, and buy up to 10 tickets.
2. Admin Management:
    - Add interest (e.g., 10%) and trigger the draw with a frontend-generated seed.
3. Post-Draw:
    - Users check their status and claim refunds and prizes via the "Claim Funds" page.

Conclusion

The Zero Loss Lottery project demonstrates a no-loss lottery concept on the Sepolia testnet, using PYUSD for transactions and a manual interest model for simplicity. The expanded UI design ensures an engaging user experience, while the technical implementation leverages Solidity, React.js, and ethers.js to deliver a functional hackathon demo. This setup highlights blockchain transparency and the innovative use of stablecoins in a lottery system.