DECENTRALISED LOTTERY SYSTEM

1. Introduction

This project is a Decentralized Lottery System built on the Ethereum blockchain. It provides a transparent, tamper-proof, and trustless alternative to traditional lottery systems. Users can participate by purchasing tickets using cryptocurrency, while a designated admin can start and end lotteries, pick winners, and manage funds—all enforced through smart contracts.

---

2. Objectives

- **Develop a secure and decentralized lottery system.**
- **Eliminate third-party control and fraud in lotteries.**
- **Ensure fairness and transparency using blockchain.**
- **Provide a user-friendly interface for participation.**

---

3. Technologies Used

Smart Contracts	**Solidity**
Blockchain Emulator	**Ganache**
Development Framework	**Truffle**
Frontend	**React.js**
Blockchain Interface	**Web3.js**
Wallet Integration	**Metamask**

4. System Architecture

A. Backend (Blockchain)

- **Smart contracts deployed using Truffle.**
- **Local Ethereum blockchain simulated with Ganache.**

B. Frontend

- **Built in React.js.**
- **Connects to Ethereum network via Web3.js.**
- **Integrated with MetaMask for user transactions.**

C. Directory Structure

bash

CopyEdit

```
                    lottery-system/
├── contracts/               # Solidity Smart Contracts
   ├── migrations/              # Deployment Scripts
     ├── client/                  # React Frontend
         ├── src/contracts/       # Contract ABIs
 |    └── .env                    # Environment Variables
    ├── truffle-config.js        # Truffle Settings
```

5. Features

Admin

- **Creates the lottery.**
- **Initiates the button which picks a winner.**
- **View list of participants.**

User

- Join the lottery by paying ETH.
- View current lottery status.
- Get notified of the winner.

---

6. Smart Contract Design

Key State Variables

Solidity

```
address public admin;
address[] public players;
uint public ticketPrice;
bool public lotteryActive;
```

Important Functions

- `createLottery()` – Initializes a new lottery with specified ticket price and player limit.
- `buyTicket()` – Allows a user to join the lottery by paying the ticket price.
- `pickWinner()` – Randomly selects a winner from participants and marks the lottery as ended (admin only).
- `claimPrize()` – Lets the winner withdraw their prize after the lottery ends.
- `getParticipants()` – Returns the list of current participants in the lottery.
- `getAllLotteries()` – Returns lottery status, ticket price, prize pool, and participant count.

Modifiers

- `onlyAdmin`: Ensures only the admin can execute specific functions.
- `require(msg.value >= ticketPrice)`: Enforces minimum entry payment.

---

7. Optimizations

Smart Contract

- Used `memory` variables to reduce storage costs.
- Avoided loops with high gas usage.
- Grouped operations for efficiency.

Frontend

- Used React hooks for dynamic state handling.
- Used event listeners for contract changes.
- Cached contract instances for performance.

Dev Environment

- Used `.env` for flexibility.
- Copied contract ABIs only once to reduce redundancy.

---

8. Security Measures

Threat	Mitigation Strategy
Reentrancy Attacks	Follows Checks-Effects-Interactions pattern.
Randomness Bias	Uses pseudo-random generation (not secure for production).
Fund Misuse	Auto transfers to winner, no manual handling.
Private Key Safety	All key operations handled via MetaMask.
