

- [1 Web3 Number Guessing Game - Technical Documentation](#)
  - [1.1 Project Overview](#)
  - [1.2 Table of Contents](#)
  - [1.3 Architecture](#)
    - [1.3.1 High-Level Architecture Diagram](#)
    - [1.3.2 Transaction Flow](#)
  - [1.4 Frontend \(Flutter\)](#)
    - [1.4.1 Project Structure](#)
    - [1.4.2 Key Components](#)
    - [1.4.3 State Management](#)
    - [1.4.4 UI/UX Design](#)
  - [1.5 Backend \(Smart Contracts\)](#)
    - [1.5.1 Token Contract](#)
    - [1.5.2 Game Contract](#)
    - [1.5.3 Security Considerations](#)
  - [1.6 Integration Layer](#)
    - [1.6.1 Web3 Service](#)
    - [1.6.2 Storage Service](#)
  - [1.7 Deployment Information](#)
    - [1.7.1 !\[\]\(71ac35c616fd8bfda805d579390e24d8\_img.jpg\) Live Deployment Details](#)
  - [1.8 Getting Started](#)
    - [1.8.1 Prerequisites](#)
    - [1.8.2 Installation](#)
    - [1.8.3 Deployment](#)
  - [1.9 Game Mechanics](#)
    - [1.9.1 Gameplay](#)
    - [1.9.2 Transaction Architecture](#)
    - [1.9.3 Reward Structure](#)
  - [1.10 Development Scripts](#)
    - [1.10.1 Essential Scripts \(Kept\)](#)
    - [1.10.2 Removed Scripts](#)
  - [1.11 Testing](#)
    - [1.11.1 Frontend Testing](#)
    - [1.11.2 Smart Contract Testing](#)
    - [1.11.3 Manual Testing](#)
  - [1.12 Known Issues & Limitations](#)
    - [1.12.1 Architectural Limitations](#)
    - [1.12.2 Smart Contract Limitations](#)
    - [1.12.3 Frontend Limitations](#)
    - [1.12.4 Security Limitations](#)
  - [1.13 Future Improvements](#)
    - [1.13.1 Critical Architecture Improvements](#)
    - [1.13.2 Short-term Improvements](#)
    - [1.13.3 Medium-term Features](#)
    - [1.13.4 Long-term Vision](#)
    - [1.13.5 Recommended Migration Path](#)

# 1 Web3 Number Guessing Game - Technical Documentation

## 1.1 Project Overview

The Web3 Number Guessing Game is a blockchain-based application built with Flutter and Solidity where players guess numbers between 0-100 and earn GUESS tokens based on their accuracy. The game leverages Ethereum smart contracts to handle game logic and token distribution in a transparent and decentralized manner. **The game is completely free-to-play** - players only receive rewards when they win, and there are no entry fees.

**Architecture Note:** The current implementation uses a centralized transaction approach where all blockchain transactions are signed by a game manager's private key rather than individual user wallets. This provides a gasless gaming experience but has implications for true decentralization.

## 1.2 Table of Contents

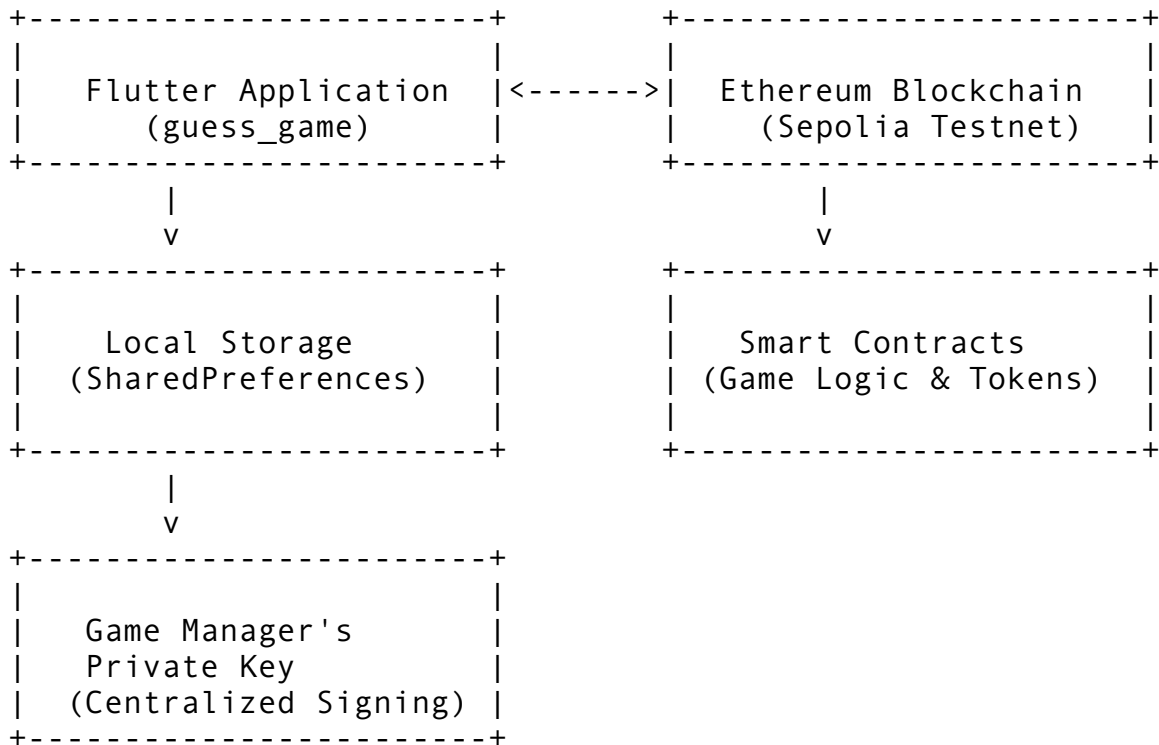
1. [Architecture](#)
2. [Frontend \(Flutter\)](#)
  - [Project Structure](#)
  - [Key Components](#)
  - [State Management](#)
  - [UI/UX Design](#)
3. [Backend \(Smart Contracts\)](#)
  - [Token Contract](#)
  - [Game Contract](#)
  - [Security Considerations](#)
4. [Integration Layer](#)
  - [Web3 Service](#)
  - [Storage Service](#)
5. [Deployment Information](#)
  - [Live Deployment Details](#)
  - [Contract Addresses](#)
  - [Network Configuration](#)
6. [Getting Started](#)
  - [Prerequisites](#)
  - [Installation](#)
  - [Deployment](#)
7. [Game Mechanics](#)
  - [Gameplay](#)
  - [Reward Structure](#)
8. [Development Scripts](#)
9. [Testing](#)
10. [Known Issues & Limitations](#)
11. [Future Improvements](#)

## 1.3 Architecture

The project follows a client-server architecture with:

- **Client:** Flutter mobile application (guess\_game)
- **Backend:** Ethereum blockchain with smart contracts
- **Integration:** Web3Dart library with centralized transaction signing
- **Network:** Deployed on Sepolia Testnet for testing

### 1.3.1 High-Level Architecture Diagram



### 1.3.2 Transaction Flow

1. **User Input:** Player enters guess in Flutter app
2. **Centralized Processing:** App uses game manager's private key to sign transaction
3. **Blockchain Execution:** Smart contract processes game with game manager as `msg.sender`
4. **Result Storage:** Game results stored under game manager's address
5. **Reward Distribution:** Tokens distributed to intended user address (specified in contract call)
6. **Local Tracking:** App maintains local record of user's games and results

## 1.4 Frontend (Flutter)

### 1.4.1 Project Structure

```
lib/
├── constants/          # App constants and contract addresses
│   ├── app_constants.dart # Main app constants
│   └── constants.dart     # Export file
```

```

├── contracts/           # ABI definitions and contract config
│   ├── contract_config.dart # Contract addresses and network
│   ├── erc20_abi.dart      # ERC-20 token ABI
│   ├── game_contract_abi.dart # Game contract ABI
│   └── contracts.dart      # Export file
├── config
│   ├── erc20_abi.dart      # ERC-20 token ABI
│   ├── game_contract_abi.dart # Game contract ABI
│   └── contracts.dart      # Export file
├── models/             # Data models
│   ├── game_result.dart    # Game result data structure
│   └── models.dart         # Export file
├── providers/          # State management
│   ├── app_provider.dart   # Main app state provider
│   └── providers.dart      # Export file
├── screens/            # UI screens
│   ├── home_screen.dart    # Main game interface
│   └── screens.dart        # Export file
├── services/           # Business logic services
│   ├── web3_service.dart   # Blockchain interaction service
│   ├── storage_service.dart # Local storage management
│   └── services.dart       # Export file
├── lib.dart            # Main library export
└── main.dart           # App entry point

```

## 1.4.2 Key Components

1. **Main Application** (`main.dart`):
  - Entry point for the Flutter application
  - Configures Material Design 3 theme (light/dark mode)
  - Sets up Provider state management
  - Initializes the application
2. **Home Screen** (`screens/home_screen.dart`):
  - Primary user interface for the game
  - Handles wallet connection state
  - Game play interface with number input
  - Real-time result display with performance indicators
  - Statistics display (games played, total rewards, accuracy)
3. **Game Result Model** (`models/game_result.dart`):
  - Represents the outcome of a single game
  - Stores target number, user guess, difference, reward amount, and timestamp

## 1.4.3 State Management

The app uses the Provider pattern for centralized state management:

- **AppProvider** (`providers/app_provider.dart`):
  - Manages wallet connection state and user address
  - Handles game state (idle, playing, showing results)
  - Coordinates with Web3Service for blockchain interactions
  - Manages loading states and error handling
  - Stores game statistics and history

## 1.4.4 UI/UX Design

- **Material Design 3:** Modern design system with Material You theming

- **Responsive Layout:** Adapts to different screen sizes and orientations
- **Color-coded Results:** Performance indicators with intuitive color schemes
- **Loading States:** Smooth loading animations during blockchain transactions
- **Error Handling:** User-friendly error messages and recovery options
- **Dark/Light Theme:** Automatic theme switching based on system preferences

## 1.5 Backend (Smart Contracts)

### 1.5.1 Token Contract

GuessToken.sol - ERC-20 token contract with enhanced features:

- **Token Details:**
  - Name: Guess Token
  - Symbol: GUESS
  - Decimals: 18
  - Max Supply: 1,000,000 tokens
  - Initial Owner Supply: 100,000 tokens
- **Key Features:**
  - **Minting System:** Role-based minting with owner control
  - **Access Control:** Minter role management (add/remove minters)
  - **Pausable:** Emergency pause functionality
  - **Burning:** Token holders can burn their tokens
  - **Supply Cap:** Hard cap at 1 million tokens
- **Security Features:**
  - OpenZeppelin standard implementation
  - Pause functionality for emergency situations
  - Role-based access control

### 1.5.2 Game Contract

NumberGuessingGame.sol - Main game logic contract:

- **Game Mechanics:**
  - **Free-to-Play:** No entry fees, players only receive rewards for winning
  - **Random Number Generation:** Pseudo-random (use Chainlink VRF for production)
  - **Winning Condition:** Guesses within 20 points of target are considered wins
  - **Automatic Rewards:** Winners receive tokens automatically
- **Key Functions:**
  - `playGame(uint256 guess)`: Main game function (free to play)
  - `getUserGameHistory(address user)`: Retrieves complete game history
  - `getLatestGameResult(address user)`: Gets the most recent game
  - `getUserTotalRewards(address user)`: Total rewards earned
  - `getUserTotalGames(address user)`: Total games played
  - `getUserAverageAccuracy(address user)`: Average guess accuracy
- **Reward Structure:**
  - Perfect Guess (0 difference): 50 GUESS tokens (10 base + 40 bonus)
  - Excellent ( $\leq 5$  difference): 17.5 GUESS tokens (10 + 75% bonus)
  - Very Good ( $\leq 10$  difference): 15 GUESS tokens (10 + 50% bonus)
  - Good ( $\leq 20$  difference): 12.5 GUESS tokens (10 + 25% bonus)
  - Poor ( $> 20$  difference): 0 GUESS tokens (loss, but free to play)

### 1.5.3 Security Considerations

- **OpenZeppelin Libraries:** Uses audited, battle-tested contract libraries
- **Reentrancy Protection:** ReentrancyGuard on all state-changing functions
- **Access Control:** Owner-only functions for contract administration
- **Pausable Contracts:** Emergency pause functionality
- **Input Validation:** Strict validation of all user inputs
- **Safe Math:** Built-in overflow protection in Solidity 0.8+

## 1.6 Integration Layer

### 1.6.1 Web3 Service

`web3_service.dart` handles all blockchain interactions with a centralized approach:

- **Connection Management:** Initializes Web3 client with Sepolia testnet
- **Contract Interaction:** Loads and interacts with deployed smart contracts
- **Centralized Signing:** Uses a hardcoded game manager private key for all transactions
- **Gasless Gaming:** Players don't need ETH or wallet setup - all gas fees paid by game manager
- **Transaction Processing:** Handles game transactions and confirmations on behalf of users
- **Error Handling:** Comprehensive error handling for blockchain operations
- **Gas Management:** Appropriate gas limits for contract interactions

**Key Implementation Details:** - All transactions signed with game manager's private key:

XXXXXXXXXX - Game manager address:

0xA720e09cfB31fcd03d74992373AEcF0818F111Af - Users provide only their address for reward distribution - No wallet connection required from users

### 1.6.2 Storage Service

`storage_service.dart` manages local data persistence:

- **User Address Storage:** Stores user's Ethereum address for reward distribution
- **Game History:** Local tracking of user's game results and statistics
- **User Preferences:** Saves app settings and preferences
- **Session Management:** Handles user session state
- **Data Clearing:** Clean data removal when user resets or changes address

## 1.7 Deployment Information

### 1.7.1 Live Deployment Details

The Web3 Number Guessing Game is currently deployed on **Sepolia Testnet** with the following configuration:

#### 1.7.1.1 Contract Addresses

Contract	Address	Purpose
** (ERC-20)**	0x2AC923843d160A63877b83EC7bC69027C97bc45e	

Contract	Address	Purpose
NumberGuessingGame	0x2a7081a264DDF15f9e43B237967F3599D743B0f5	GUESS token rewards Main game logic

### 1.7.1.2 🌐 Network Configuration

Parameter	Value
Network Name	Sepolia Testnet
Chain ID	11155111
RPC URL	<a href="https://ethereum-sepolia-rpc.publicnode.com">https://ethereum-sepolia-rpc.publicnode.com</a>
Currency Symbol	ETH
Block Explorer	<a href="https://sepolia.etherscan.io">https://sepolia.etherscan.io</a>

### 1.7.1.3 🔍 View Contracts on Block Explorer

- **GuessToken Contract:** <https://sepolia.etherscan.io/address/0x2AC923843d160A63877b83EC7bC69027C97bc45e>
- **Game Contract:** <https://sepolia.etherscan.io/address/0x2a7081a264DDF15f9e43B237967F3599D743B0f5>

### 1.7.1.4 💰 Get Testnet Tokens

To play the game, you need Sepolia ETH for gas fees:

Faucet	URL	Daily Limit
Sepolia Faucet	<a href="https://sepoliafaucet.com">https://sepoliafaucet.com</a>	0.5 ETH
Alchemy Faucet	<a href="https://sepoliafaucet.net">https://sepoliafaucet.net</a>	0.5 ETH
QuickNode Faucet	<a href="https://faucet.quicknode.com/ethereum/sepolia">https://faucet.quicknode.com/ethereum/sepolia</a>	0.1 ETH

### 1.7.1.5 📱 Add Sepolia Network to MetaMask

To connect to the game, add Sepolia testnet to your wallet:

```
{
  "networkName": "Sepolia Testnet",
  "rpcUrl": "https://ethereum-sepolia-rpc.publicnode.com",
  "chainId": "11155111",
  "symbol": "ETH",
```

```
"explorerUrl": "https://sepolia.etherscan.io"
}
```

**Quick Add Button:** [Add Sepolia to MetaMask](#)

#### 1.7.1.6 🎮 Ready to Play?

1. ✅ Add Sepolia network to your wallet
  2. ✅ Get some Sepolia ETH from faucets above
  3. ✅ Download the Flutter app
  4. ✅ Connect your wallet and start guessing!
- 

## 1.8 Getting Started

### 1.8.1 Prerequisites

- **Flutter SDK:** 3.7.0 or higher
- **Node.js:** 16.0 or higher
- **Git:** For version control
- **Ethereum Wallet:** MetaMask or compatible Web3 wallet
- **Sepolia ETH:** For testing transactions (free from faucets above)

### 1.8.2 Installation

#### 1. Clone the repository

```
git clone <repository-url>
cd quiz_app
```

#### 2. Install Flutter dependencies

```
flutter pub get
```

#### 3. Install smart contract dependencies

```
cd smart-contracts
npm install
cd ..
```

### 1.8.3 Deployment

#### 1. Configure Environment (edit smart-contracts/hardhat.config.js)

```
networks: {
  sepolia: {
    url: "YOUR_SEPOLIA_RPC_URL",
    accounts: ["YOUR_PRIVATE_KEY"]
  }
}
```



## 2. Deploy Contracts

```
cd smart-contracts
npx hardhat run scripts/deploy-testnet.js --network sepolia
```

## 3. Update Contract Addresses in lib/constants/app\_constants.dart:

```
static const String guessTokenContractAddress =
    'NEW_TOKEN_ADDRESS';
static const String gameContractAddress = 'NEW_GAME_ADDRESS';
```

## 4. Generate ABI Files

```
cd smart-contracts
npx hardhat run scripts/generate-abi.js
```

## 5. Run the App

```
flutter run
```

# 1.9 Game Mechanics

## 1.9.1 Gameplay

1. **Address Input:** User provides their Ethereum address (no wallet connection needed)
2. **Game Start:** User initiates a new game (completely free, no gas fees)
3. **Number Input:** User enters a guess between 0-100
4. **Centralized Processing:** App signs transaction with game manager's private key
5. **Blockchain Processing:** Smart contract generates random number and calculates results
6. **Reward Distribution:** Winners automatically receive GUESS tokens at their provided address
7. **Result Display:** Game shows target number, difference, and reward earned

## 1.9.2 Transaction Architecture

**Important:** All blockchain transactions are processed through the game manager's account: - **msg.sender:** Always the game manager (0xA720e09cfB31fcd03d74992373AEcF0818F111Af) - **Reward recipient:** User's provided address (specified in contract parameters) - **Game history:** Stored under game manager's address in smart contract - **User tracking:** Maintained locally in the Flutter app

## 1.9.3 Reward Structure

The game uses a tiered reward system based on guess accuracy:

Performance Level	Difference Range	Reward Amount	Description
Perfect	0	50 GUESS	Exact match - maximum reward
Excellent	1-5	17.5 GUESS	Very close guess
Very Good	6-10	15 GUESS	Close guess
Good	11-20	12.5 GUESS	Moderate accuracy

Performance Level	Difference Range	Reward Amount	Description
Loss	21+	0 GUESS	No reward, but free to play

### 1.9.3.1 Example Scenarios

**Target Number: 42**

Player	Guess	Difference	Performance	Reward
Alice	42	0	Perfect	50 GUESS
Bob	46	4	Excellent	17.5 GUESS
Carol	51	9	Very Good	15 GUESS
Dave	60	18	Good	12.5 GUESS
Eve	72	30	Loss	0 GUESS

## 1.10 Development Scripts

The project includes essential development scripts in `smart-contracts/scripts/`:

### 1.10.1 Essential Scripts (Kept)

1. `deploy-testnet.js`:
  - Main deployment script for testnet deployment
  - Deploys both GuessToken and NumberGuessingGame contracts
  - Sets up initial token approvals
  - Provides comprehensive deployment information and next steps
2. `generate-abi.js`:
  - Generates ABI files for Flutter integration
  - Creates filtered ABIs with only necessary functions
  - Outputs Dart files for contract interaction
3. `check-balance.js`:
  - Simple utility to check account balance
  - Useful for verifying wallet funding before deployment

### 1.10.2 Removed Scripts

The following development and testing scripts have been removed to keep the codebase clean: - All `test-*.js` files (18 test scripts) - Debug scripts (`debug-*.js`, `check-transactions.js`) - Demo scripts (`demo-*.js`) - Fix scripts (`fix-*.js`) - Old deployment scripts (`deploy-updated-contract.js`) - Utility scripts (`transfer-tokens.js`, `show-test-addresses.js`)

## 1.11 Testing

### 1.11.1 Frontend Testing

- **Unit Tests**: Test individual components and services
- **Widget Tests**: Test UI components and user interactions
- **Integration Tests**: Test complete user flows

### 1.11.2 Smart Contract Testing

- **Hardhat Tests:** Comprehensive contract testing
- **Network Testing:** Live testing on Sepolia testnet
- **Security Testing:** Audit contract security features

### 1.11.3 Manual Testing

1. Connect different wallet types
2. Test various guess scenarios
3. Verify reward calculations
4. Test error handling

## 1.12 Known Issues & Limitations

### 1.12.1 Architectural Limitations

#### 1.12.1.1 Centralized Transaction Model

- **All transactions signed by game manager:** The app uses a single private key for all blockchain interactions
- **User address mismatch:** Smart contract records all games under game manager's address, not individual users
- **Local-only user tracking:** Individual user statistics are maintained only in the app, not on-chain
- **Trust dependency:** Users must trust the game manager to distribute rewards correctly

#### 1.12.1.2 Implications of Current Architecture

- **Not truly decentralized:** Despite using blockchain, the transaction model is centralized
- **User game history:** `getLatestGameResult(userAddress)` will show "No games played" for individual users
- **Reward distribution works:** Tokens are correctly sent to user addresses despite centralized signing
- **Scalability concerns:** All gas costs borne by single game manager account

### 1.12.2 Smart Contract Limitations

- **Pseudo-Random Numbers:** Current implementation uses block-based randomness (not production-ready)
- **Centralized Rewards:** Owner must fund the contract with tokens for rewards
- **Game History Inconsistency:** Individual user addresses show zero games in contract queries
- **Single Point of Failure:** Game manager's private key compromise would affect entire system

### 1.12.3 Frontend Limitations

- **Mobile Focus:** UI optimized primarily for mobile devices
- **Address Input Required:** Users must manually provide Ethereum address
- **Network Dependency:** Requires stable internet connection
- **Local Data Dependency:** User statistics lost if app data is cleared

### 1.12.4 Security Limitations

- **Private Key Exposure:** Game manager's private key is embedded in the application code
- **No User Authentication:** No verification that provided address belongs to the user
- **Transaction Replay:** No protection against transaction replay attacks in current implementation

## 1.13 Future Improvements

### 1.13.1 Critical Architecture Improvements

- **True Wallet Integration:** Implement proper user wallet connection and individual transaction signing
- **Gasless Solutions:** Use meta-transactions or account abstraction for gasless gaming while maintaining decentralization
- **User Game History:** Modify smart contract to properly track individual user game histories
- **Enhanced Security:** Remove private key from application code, implement secure key management

### 1.13.2 Short-term Improvements

- **Chainlink VRF Integration:** Implement truly random number generation
- **Layer 2 Deployment:** Deploy on Polygon or Arbitrum for lower gas costs
- **Improved UI:** Enhanced mobile and web responsiveness
- **User Verification:** Add address ownership verification

### 1.13.3 Medium-term Features

- **Proper Decentralization:** Transition to individual user wallet interactions
- **Multiplayer Games:** Real-time multiplayer guessing competitions
- **Leaderboards:** Global and weekly leaderboards (with proper on-chain tracking)
- **Social Features:** Share results and challenge friends
- **Achievement System:** Badges and achievements for milestones

### 1.13.4 Long-term Vision

- **Tournament System:** Organized tournaments with bigger rewards
- **NFT Integration:** Special NFT rewards for top performers
- **Cross-chain Support:** Multi-chain deployment for broader accessibility
- **Advanced Analytics:** Detailed player statistics and performance tracking
- **Governance Token:** Community governance for game parameters

### 1.13.5 Recommended Migration Path

1. **Phase 1:** Implement proper wallet connection while maintaining gasless option
2. **Phase 2:** Add meta-transaction support for truly gasless decentralized gaming
3. **Phase 3:** Update smart contracts to properly track individual user histories
4. **Phase 4:** Remove centralized transaction signing and migrate to full decentralization