

## **FINAL PROJECT**

**Team members:** Kairkulova Dilnaz, Myrzakhan Mereke

**PredictionChain – Decentralized Crowdfunding for Prediction Events**

### **1. Project description:**

#### **1. Overview of the Application Architecture**

Our project is called PredictionChain, and the main idea is to build a decentralized prediction platform where users can bet on different outcomes using test ETH. In this system, an event is created with a title, a deadline, and several possible outcomes such as A/B/C. Users connect their MetaMask wallets, choose an outcome they believe will happen, and send test ETH to the smart contract. All contributions are stored on-chain, and the contract tracks exactly how much each user placed on each outcome.

A campaign becomes finalizable once the deadline passes. After that, the event creator calls the function to finalize the campaign and choose the correct outcome. If a user predicted the right outcome, they are allowed to claim a proportional share of the entire ETH pool. This creates a transparent and fair distribution mechanism where everyone's reward depends on how much they contributed.

The system also includes an ERC-20 token called PredictionChain Participation Token (PRED), which was deployed on the Sepolia test network. This token is not used as real money but works as a participation or reputation token. For every 1 wei contributed to any event, the contract automatically mints 1 PRED token to the user.

The main smart contracts — PredictionToken and PredictionChain — were deployed through Remix using MetaMask. I manually tested all core functions: creating events, placing bets with ETH value, minting PRED tokens, and checking updated user balances. The architecture focuses on simplicity, transparency, and safe prediction categories such as game releases, movie ratings, IT updates, or true/false events. Overall, PredictionChain demonstrates a complete decentralized prediction workflow with real blockchain interaction on the Sepolia testnet.

Since the project is developed for educational purposes, the event finalization process is performed manually by the contract owner. This approach allows the system to function without external data sources.

In a production-ready prediction market, event outcomes would typically be resolved using decentralized oracles or governance mechanisms. However, such solutions were intentionally excluded from this project to keep the focus on smart contract fundamentals, frontend interaction, and secure on-chain logic.

## **2. Explanation of Design and Implementation Decisions**

The system was designed with clarity and educational value in mind. A prediction-based model was chosen instead of regular crowdfunding because it clearly demonstrates smart contract features: deadlines, reward logic, outcome selection, and proportional distribution.

We separated the logic into two contracts:

- **PredictionToken** handles ERC-20 reward token minting.
- **PredictionChain** handles events, bets, deadlines, and payouts.

The decision to mint 1 PRED per 1 wei makes the reward formula simple, transparent, and easy for students to verify. Deployment through Remix and MetaMask was also chosen because this method provides the clearest visualization of testnet interactions.

## **3. Description of Smart Contract Logic**

The PredictionChain contract enables users to create prediction events by specifying a title, deadline, and number of outcomes. Each event stores its own ETH pool and outcome pools.

The placeBet() function accepts test ETH and records the contribution amount inside mappings. The function also triggers ERC-20 minting using the PredictionToken contract.

Once the deadline has passed, the creator finalizes the event by selecting the correct outcome. Winning participants can then claim their proportional share of the pool, based on how much they contributed to the correct outcome.

The PredictionToken contract follows the ERC-20 standard and includes setMinter() to restrict minting rights only to the PredictionChain contract.

## **4. Explanation of Frontend-to-Blockchain Interaction**

Although the contracts were tested directly through Remix, the same logic applies to any JavaScript frontend. MetaMask is used to request account access, check the selected network, and sign all blockchain transactions.

The interaction flow is:

1. MetaMask connects the user's address.

- 2. The user selects an action (create event, bet, finalize).**
- 3. The frontend sends a transaction to the smart contract.**
- 4. MetaMask asks the user to confirm.**
- 5. The transaction is mined and reflected on-chain.**

**Remix fully supports MetaMask integration, so all interactions were performed and verified through Sepolia.**

## **5. Deployment and Execution Instructions**

- 1. Install MetaMask and switch to the Sepolia test network.**
- 2. Obtain test ETH using a faucet.**
- 3. Open Remix IDE and import both Solidity files.**
- 4. Compile contracts using Solidity 0.8.28.**
- 5. Deploy PredictionToken with Injected Provider – MetaMask.**
- 6. Deploy PredictionChain, passing the token's address.**
- 7. Call setMinter() on PredictionToken to set PredictionChain as the minter.**
- 8. Test the system by creating events, placing bets, and checking token balances.**

**All deployments and tests were completed successfully in Sepolia.**

## **6. Description of the Process for Obtaining Test ETH**

**To obtain test ETH for deployment and testing, I used a public Sepolia faucet. The steps were:**

- 1. Copy my MetaMask wallet address.**
- 2. Visit the faucet ([sepolia-faucet.pk910.de](http://sepolia-faucet.pk910.de)).**
- 3. Paste the address and request ETH.**

4. The faucet sent 0.25 SepoliaETH within seconds.
5. The balance appeared in MetaMask and was used for smart contract deployment and transactions.

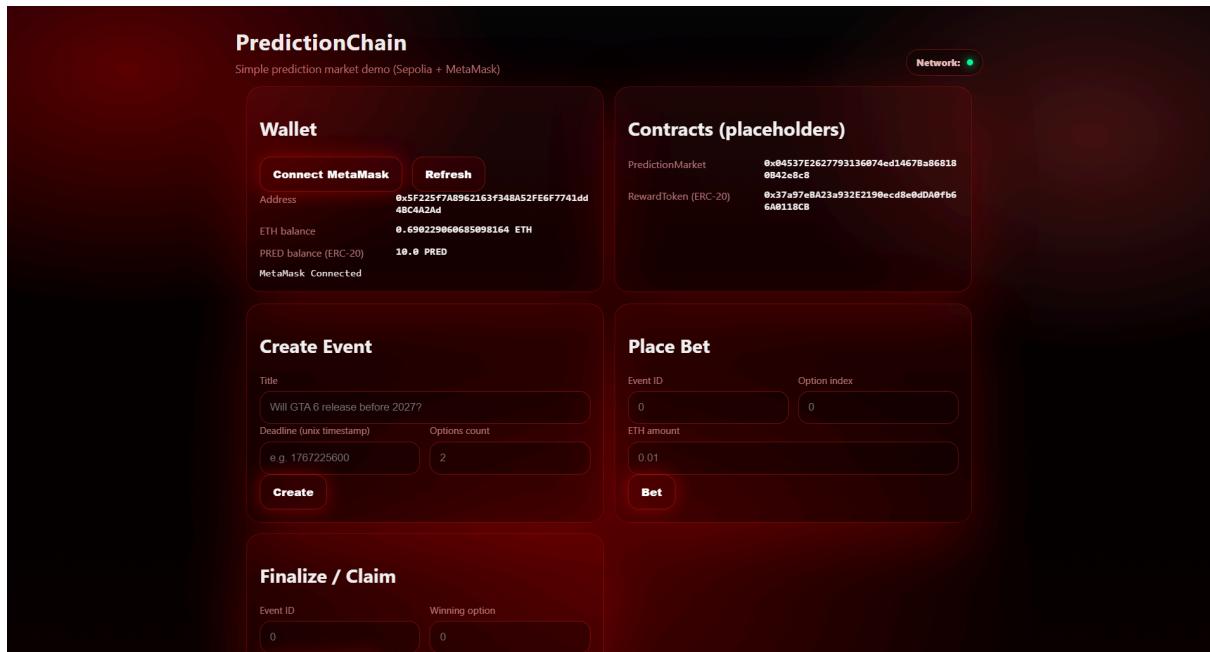
This ETH is only for testing and has no real monetary value.

## 7. Screenshots / Experimental Results

This section presents screenshots that demonstrate successful deployment and execution of the PredictionChain decentralized application on the Sepolia test network.

### Screenshot 1 – MetaMask Connected

**Description:** MetaMask wallet connected to the Sepolia test network with the active account displayed.



### Screenshot 2 – Contract Deployment

**Description:** Deployment of the PredictionToken and PredictionChain smart contracts using Remix IDE and MetaMask.

The screenshot shows the REMIX IDE interface with the following details:

- REMIX Version:** 1.5.1
- Deploy & Run Transactions:** Account 3 (0x5f2...4a2ad) selected.
- Gas Limit:** Custom set to 3000000 Wei.
- Contract:** IERC1155Errors - @openzeppelin/contracts.
- Code:** RewardToken.sol (ERC20 contract code).
- MetaMask:** Shows a transaction for "Contract deployment" from Account 3 to SepoliaETH with a total value of 0.0049403 SepoliaETH, status confirmed.
- Activity Log:** One transaction listed as "Confirmed" from Account 3 to SepoliaETH.

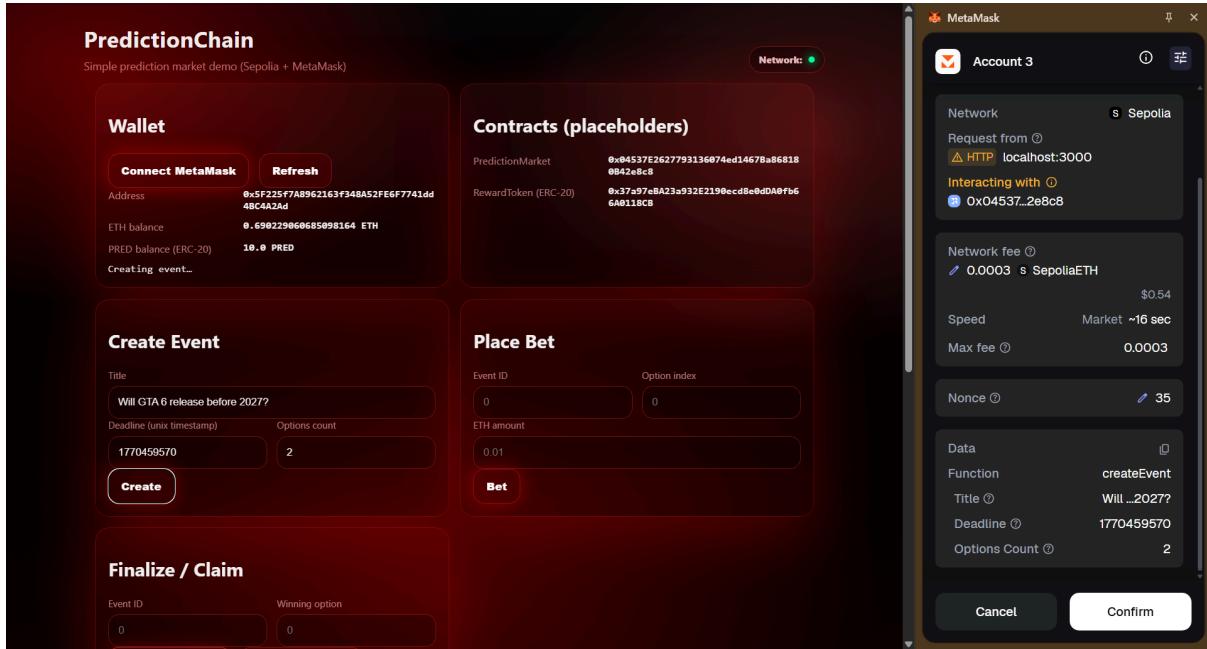
  

The screenshot shows the REMIX IDE interface with the following details:

- REMIX Version:** 1.5.1
- Deploy & Run Transactions:** Account 3 (0x5f2...4a2ad) selected.
- Gas Limit:** Custom set to 3000000 Wei.
- Contract:** IERC1155Errors - @openzeppelin/contracts.
- Code:** RewardToken.sol (ERC20 contract code).
- MetaMask:** Shows a transaction for "Set Market" from Account 3 to 0x37a97...11BCB with a total value of 0.000011522 SepoliaETH, status confirmed.
- Activity Log:** One transaction listed as "Confirmed" from Account 3 to SepoliaETH.

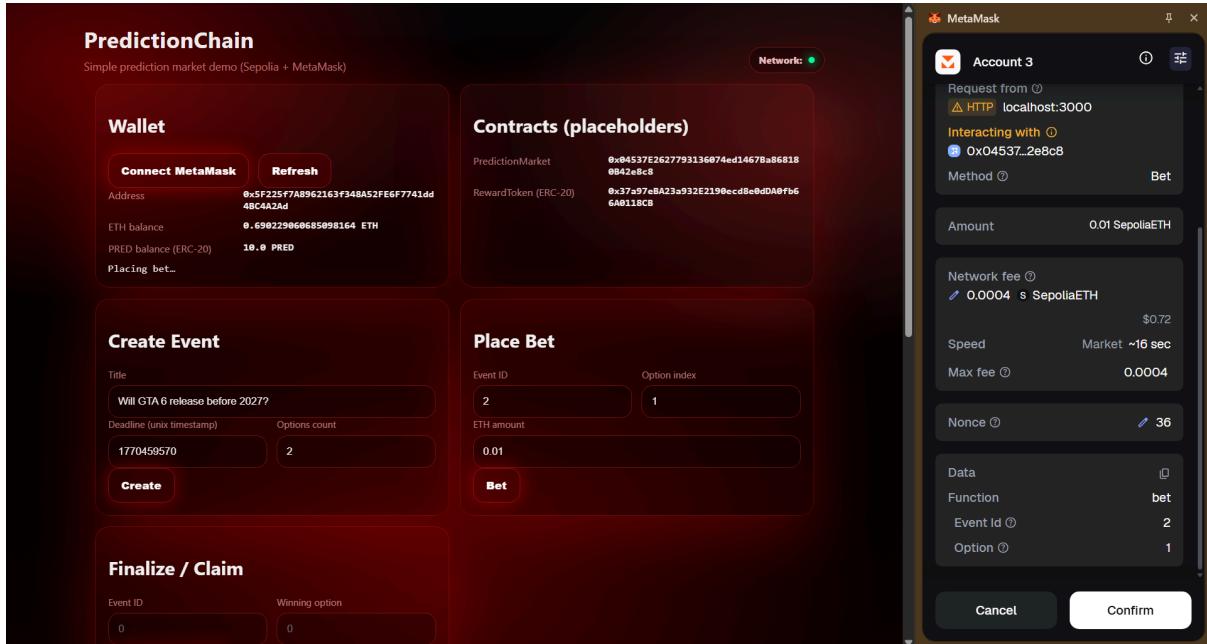
## Screenshot 3 – Create Event Transaction

**Description: Successful execution of the createEvent() function confirmed in MetaMask.**



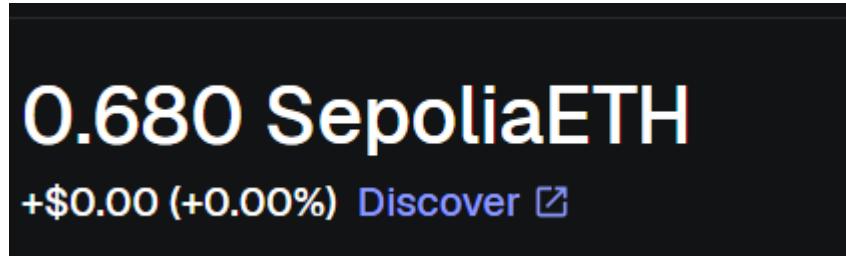
## Screenshot 4 – Place Bet with ETH

**Description:** User placing a bet by sending test ETH to the PredictionChain contract.



## Screenshot 5 – ERC-20 Token Balance

**Description:** Updated PRED token balance after participating in a prediction event.



## Screenshot 6 – Finalize Event

*Description:* Event finalization after the deadline with a selected winning outcome.

A screenshot of a web-based application interface for managing a prediction market. It includes sections for "Create Event", "Place Bet", and "Finalize / Claim". The "Create Event" section shows a title "Will GTA 6 release before 2027?", a deadline of "1770459570", and an option count of "2". The "Place Bet" section shows an event ID of "2", an option index of "1", and a bet amount of "0.01". The "Finalize / Claim" section shows an event ID of "2", a winning option of "1", and buttons for "Finalize (owner)" and "Claim (optional)". To the right, a MetaMask wallet window is open, showing account details like "Account 3", "Network: Sepolia", and a transaction history involving "0x04537..2e8c8".

## Screenshot 7 – Claim Reward

*Description:* Winning participant successfully claiming ETH reward from the contract.

Address: 0x5F225f7A8962163f348A52F6F7741dd4BC4A2d

ETH balance: 0.679607043682252265 ETH

PRED balance (ERC-20): 20.0 PRED

PredictionMarket: 0x04537E19277931507450206752000210, 0x842e8c8, 0x37a97e8A23a932E2190ecd8e0d0A0fb6

RewardToken (ERC-20): 6A0118CB

Estimated changes: You receive + 0.01 \$ SepoliaETH

Network: Sepolia

Request from: HTTP localhost:3000

Interacting with: 0x04537..2e8c8

Method: Claim

Network fee: 0.0002 \$ SepoliaETH

Speed: Market ~16 sec

Max fee: 0.0002

Nonce: 38

Account 3 ▾

0.689 SepoliaETH

+\$0.00 (+0.00%) Discover

\$ Buy

Swap

Send

Receive

Tokens DeFi NFTs Activity

Sepolia ▾

Feb 7, 2026

Claim Confirmed

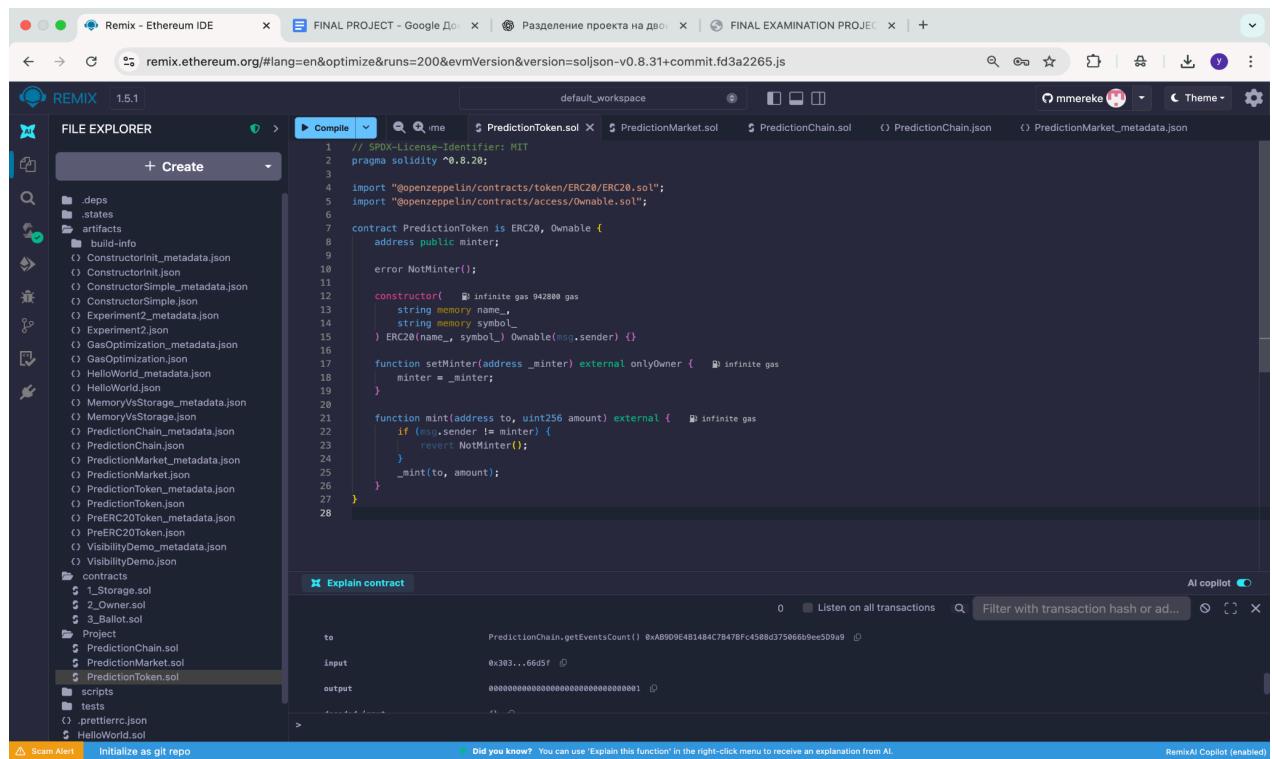
-0 SepoliaETH

-0 SepoliaETH

## 2.1. Smart Contract Implementation

Smart contract implementation in this project includes two main components:

1. RewardToken (ERC-20 participation token)
2. PredictionMarket (main prediction market and crowdfunding logic)



The screenshot shows the Remix Ethereum IDE interface. The left sidebar displays the file explorer with various contracts and artifacts. The central area shows the Solidity code for the `PredictionToken` contract. The code imports the `ERC20` and `Owable` contracts from OpenZeppelin. It defines a `PredictionToken` contract that inherits from `ERC20` and `Owable`. The constructor takes `_name`, `_symbol`, and `_minter` as parameters. It includes a `setMinter` function to change the minter address and a `mint` function to mint tokens to a specified address. The `mint` function checks if the sender is the minter and reverts if not. The bottom panel shows the transaction details for a recent interaction.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "openzeppelin/contracts/token/ERC20/ERC20.sol";
import "openzeppelin/contracts/access/Owable.sol";

contract PredictionToken is ERC20, Owable {
    address public minter;

    error NotMinter();

    constructor(string memory name_, string memory symbol_) ERC20(name_, symbol_) Owable(msg.sender) {}

    function setMinter(address _minter) external onlyOwner {
        minter = _minter;
    }

    function mint(address to, uint256 amount) external {
        if (msg.sender != minter) {
            revert NotMinter();
        }
        _mint(to, amount);
    }
}
```

The screenshot shows the Ethereum IDE (Remix) interface. The top navigation bar includes tabs for 'Remix - Ethereum IDE', 'FINAL PROJECT - Google Документы', 'Разделение проекта на два', and 'FINAL EXAMINATION PROJECT'. The main workspace is titled 'default\_workspace' and contains several tabs: 'PredictionToken.sol', 'PredictionChain.sol' (which is currently active), 'PredictionMarket.sol', and 'PredictionMarket\_metadata.json'. The 'FILE EXPLORER' sidebar on the left lists various Ethereum files and contracts. The central area displays the Solidity code for the 'PredictionChain' contract:

```

1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.20;
3
4 import "./PredictionMarket.sol";
5
6 contract PredictionChain is PredictionMarket {
7     constructor(address rewardToken) PredictionMarket(rewardToken) { }
8 }

```

Below the code, an 'Explain contract' panel provides details about the selected contract. It shows the contract's address (0xAB09E4B1484C70478Fc4508d375066b9ee509a9), the function 'getEventsCount()', and its parameters: 'to' (0x393...66d5f), 'input' (empty), and 'output' (00000000000000000000000000000001).

## 2.2. ERC-20 Reward Token

**The project includes an ERC-20 token called PredictionChain Participation Token (PRED), implemented in the RewardToken smart contract.**

**This token is automatically minted whenever a user participates in a prediction event. The token is not used as a currency and has no real monetary value. Instead, it serves as a participation and reputation token, demonstrating ERC-20 integration within a decentralized application.**

**The token follows the ERC-20 standard.**

**Minting rights are restricted to the PredictionMarket smart contract using a dedicated setter function.**

**No ownership-based control is used in order to keep the token logic simple, transparent, and suitable for educational purposes.**

## 2.3. Token Minting Logic

**Token minting is triggered automatically inside the placeBet() function of the PredictionMarket contract.**

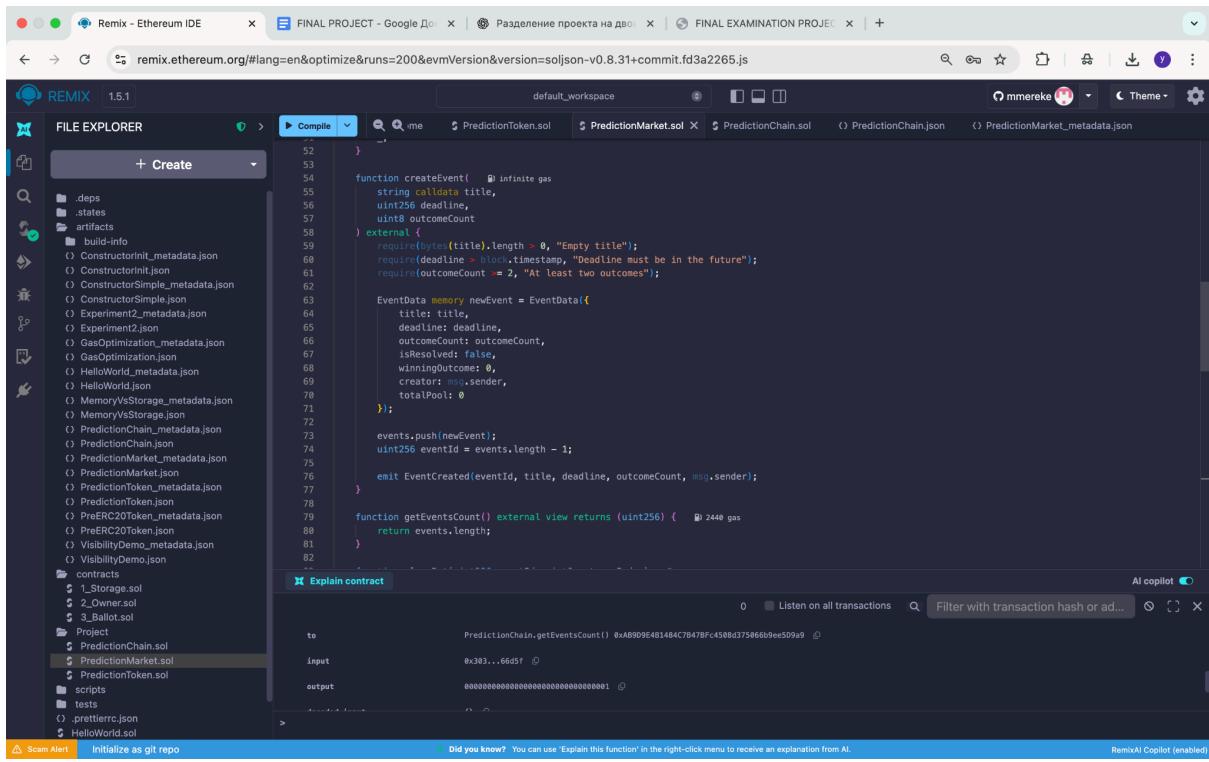
**For every 1 wei contributed to an event, the user receives 1 PRED token.**

**This simple reward formula was chosen to make the system easy to understand and to clearly demonstrate the relationship between ETH contributions and ERC-20 token issuance.**

## 2.4. Event/Campaign Creation

Users can create events (campaigns) defining:

- title
- deadline
- number of outcomes



The screenshot shows the Remix Ethereum IDE interface. The top navigation bar includes tabs for 'FINAL PROJECT - Google Документы', 'Разделение проекта на два файла', and 'FINAL EXAMINATION PROJECT'. The main workspace shows the 'default\_workspace' tab selected. On the left, the 'FILE EXPLORER' sidebar lists various Ethereum projects and files, such as 'HelloWorld.sol', 'MemoryVsStorage.sol', and 'GasOptimization.sol'. The central area displays the Solidity code for the 'PredictionToken.sol' contract. The code defines a function 'createEvent' with parameters: title (string), deadline (uint256), and outcomeCount (uint8). It includes validation logic to ensure the title is non-empty, the deadline is in the future, and there are at least two outcomes. It also creates a new event object with fields like title, deadline, outcomeCount, isResolved (false), winningOutcome (0), creator (msg.sender), and totalPool (0). The event is then added to an array of events, and a transaction is emitted with the event ID, title, deadline, outcomeCount, and sender. The bottom right corner of the interface indicates 'RemixAI Copilot (enabled)'.

```
function createEvent(
    string calldata title,
    uint256 deadline,
    uint8 outcomeCount
) external {
    require(title.length > 0, "Empty title");
    require(deadline > block.timestamp, "Deadline must be in the future");
    require(outcomeCount >= 2, "At least two outcomes");

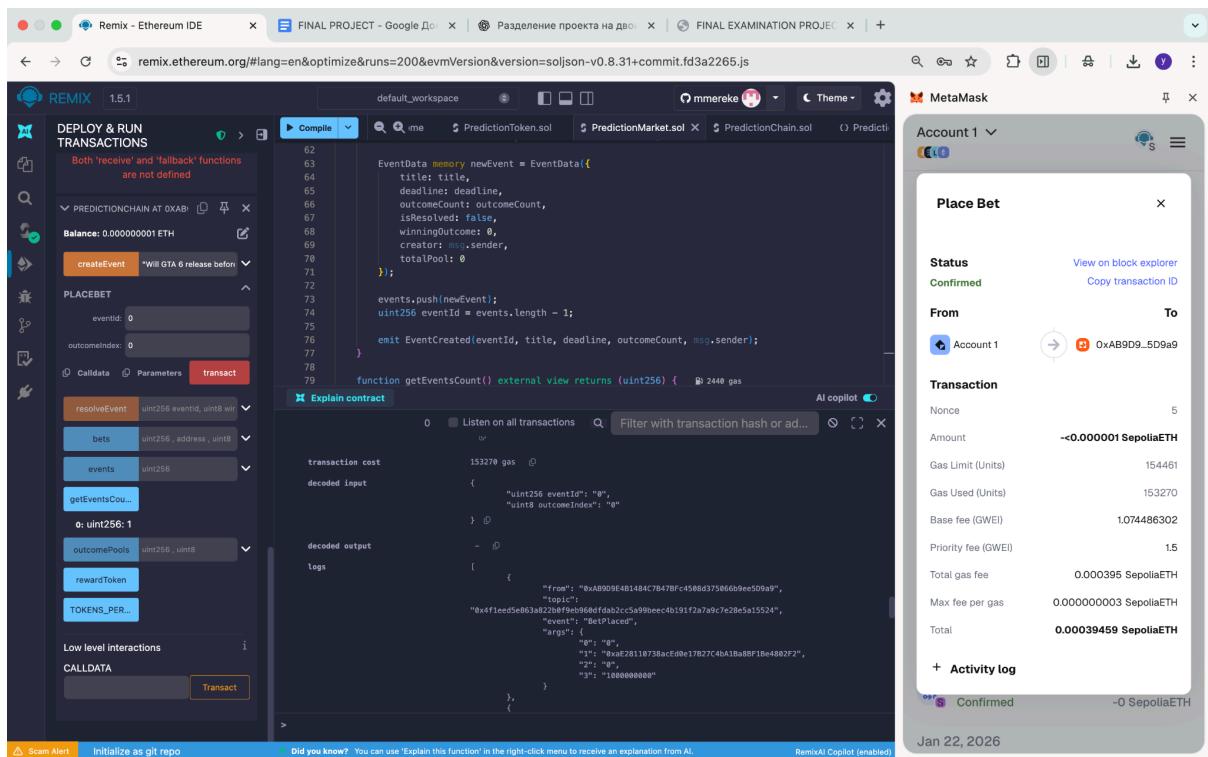
    EventData memory newEvent = EventData({
        title: title,
        deadline: deadline,
        outcomeCount: outcomeCount,
        isResolved: false,
        winningOutcome: 0,
        creator: msg.sender,
        totalPool: 0
    });

    events.push(newEvent);
    uint256 eventId = events.length - 1;
    emit EventCreated(eventId, title, deadline, outcomeCount, msg.sender);
}

function getEventsCount() external view returns (uint256) {
    return events.length;
}
```

## 2.5. Contributions (placeBet)

Users participate by sending ETH to the contract using placeBet().  
The contract records the contribution amount and outcome choice.



## 2.6. Event Finalization and Reward Claiming

**After an event reaches its deadline, it can be finalized by the event creator using the finalize() function.**

**During finalization, the correct outcome is selected.**

**A participant is considered a winner if the outcome selected during placeBet() matches the winning outcome defined during finalization.**

**Winning participants can call the claim() function to withdraw their share of the ETH pool.**

**The payout is calculated proportionally, based on the amount of ETH contributed by the user relative to the total ETH contributed to the winning outcome.**

**To ensure security:**

- each participant can claim rewards only once,
- repeated claims are rejected by the smart contract,
- participants who selected an incorrect outcome cannot withdraw ETH.

**This logic ensures a fair and transparent reward distribution mechanism enforced entirely on-chain.**

## 2.7. MetaMask Integration

The project fully integrates MetaMask:

- account connection
- Sepolia network validation
- signing all transactions (deploy, createEvent, placeBet)

The screenshot shows the Remix Ethereum IDE interface. On the left, the sidebar displays various tools and the current environment set to "Injected Provider - MetaMask" and "Sepolia (11155111) network". The main workspace shows a deployed contract named "Ownable - @openzeppelin/contract" with a balance of 0 Gwei. The right panel, titled "Account 1", shows a balance of 0.315 SepoliaETH. Below the balance are buttons for "Buy", "Swap", "Send", and "Receive". The "Activity" tab is selected, displaying a history of transactions from February 5, 2026, to January 22, 2026. The transactions listed are: "Place Bet" (Confirmed), "Contract interaction" (Confirmed), "Set Minter" (Confirmed), "Contract deployment" (Confirmed), and "Contract deployment" (Confirmed). All transactions show a change of -0 SepoliaETH.

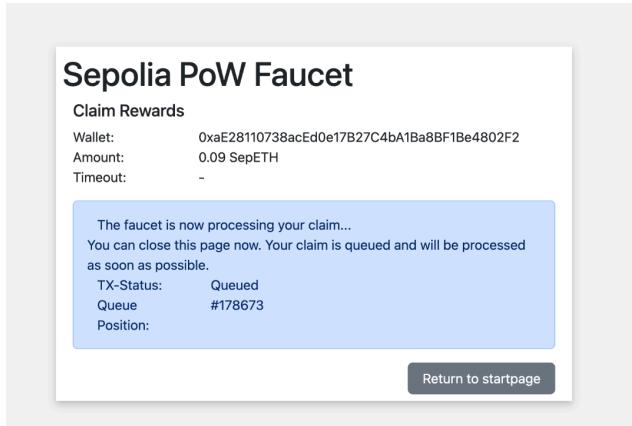
## 2.8. Sepolia Testnet Deploy

Both contracts were deployed to the Sepolia test network using Remix and MetaMask.

**PredictionToken deployed:** 0x2ac98aab672a13F69FA5e939F366dd6daCF09f56

**PredictionChain deployed:** 0xAB9D9E4B1484C7B47BFc4508d375066b9ee5D9a9

## 2.9. Test ETH Receipt



### 3. Manual Testing

**Manual contract testing was performed via Remix:**

1. **createEvent eventCount = 1**
2. **placeBet with value transaction confirmed**
3. **Check pools updated correctly**
4. **Check PRED balance tokens minted**
5. **MetaMask showed all TX confirmations**

The screenshot shows the Remix IDE interface with the following details:

- Remix 1.5.1** is the version.
- TRANSACTIONS** tab is active, showing deployed contracts and their interactions.
- PREDICTONTOKEN** at address **0x2a...** has a balance of **0 ETH**. Interactions listed include:
  - approve: address spender, uint256
  - mint: address to, uint256 amount
  - renounceOwnership
  - setMinter: address minter
  - transfer: address to, uint256 value
  - transferFrom: address from, address to, uint256 value
  - transferOwnership: address newOwner
  - allowance: address owner, address spender
  - balanceOf: address account
  - decimals
  - minter
  - name
  - owner
  - symbol
  - totalSupply
- Explain contract** AI copilot is available for the deployed code.
- Metamask** extension is connected, showing **0.316 SepoliaETH** balance and recent activity:
  - Feb 5, 2026: Set Minter, Contract deployment (both confirmed)
  - Jan 22, 2026: Contract deployment (confirmed)
- Activity** tab is selected in Metamask.

The screenshot shows a Remix IDE interface with the following details:

- Remix Version:** 1.5.1
- Default Workspace:** default\_workspace
- Contracts:** PredictionToken.sol, PredictionMarket.sol, PredictionChain.sol
- Deploy & Run Transactions:** DEPLOY & RUN TRANSACTIONS
- Balance:** 0 ETH
- PredictionChain Functions:** createEvent, placeBet, resolveEvent, bets, events, getEventsByUser, o:, outcomePools, rewardToken, TOKENS\_PER...
- Low level interactions:** CALLDATA
- MetaMask:** Account 1 - 0.316 SepoliaETH
- Contract Interaction History (Feb 5, 2026):**
  - Contract interaction -0 SepoliaETH (Confirmed)
  - Set Minter -0 SepoliaETH (Confirmed)
  - Contract deployment -0 SepoliaETH (Confirmed)
  - Contract deployment -0 SepoliaETH (Confirmed)
- Contract Deployment History (Jan 22, 2026):**
  - Contract deployment -0 SepoliaETH (Confirmed)

The screenshot shows the Remix IDE interface with the following details:

- Remix Version:** 1.5.1
- Default Workspace:** default\_workspace
- Contracts:**
  - PredictionToken.sol**
  - PredictionMarket.sol**
  - PredictionChain.sol** (Currently selected)
- Deploy & Run Transactions:** Shows a button for "totalSupply".
- Low level interactions:** Shows a note: "Both 'receive' and 'fallback' functions are not defined".
- PREDITIONCHAIN AT 0xAb1...**:
  - Balance: 0 ETH
  - Functions:
    - createEvent: "Will GTA 6 release before 2027?"
    - placeBet: uint256 eventID, uint8 out
    - resolveEvent: uint256 eventID, uint8 vir
    - bets: uint256 , address , uint8
    - events: uint256
    - getEventsCount: uint256
  - Variables:
    - 0: uint256: 1
    - outcomePools: uint256 , uint8
    - rewardToken
    - TOKENS\_PER...
  - Low level interactions
  - CALLDATA
- Explain contract**: A feature to explain Solidity code.
- raw logs**: Shows a log entry for a "createEvent" call.
- call to PredictionChain.getEventsCount**: Shows a call to the contract with parameters: from: 0xac28110738acede1b727c4ba1ba8bf1be4802f2, to: PredictionChain.getEventsCount() data: 0x30...66d5f
- Debug**: A button to start a debugger.
- MetaMask**: Shows Account 1 with 0.316 SepoliaETH.
- Transactions:**
  - Feb 5, 2026:
    - Contract interaction: Confirmed -0 SepoliaETH
    - Set Minter: Confirmed -0 SepoliaETH
    - Contract deployment: Confirmed -0 SepoliaETH
    - Contract deployment: Confirmed -0 SepoliaETH
  - Jan 22, 2026:
    - Contract deployment: Confirmed -0 SepoliaETH

The screenshot shows a dual-pane interface for Ethereum development. The left pane, titled "REMX 1.5.1", is the Remix IDE workspace. It displays a Solidity code editor with the following code:

```
pragma solidity ^0.8.0;
import "./PredictionMarket.sol";
/// @title PredictionChain
/// @dev Thin wrapper around PredictionMarket for convenience / naming
```

Below the code editor is the "Explain contract" feature, which provides an explanation for the current transaction. The transaction details are as follows:

- Call to `PredictionChain.getEventsCount`
- From: `0xaea28110738acd0e17b274caba8bf1be4802f2`
- To: `PredictionChain.getEventsCount()`
- Data: `0x30...66d5f`
- Input: `0x303...06d5f`
- Output: `0000000000000000000000000000000000000001`
- Decoded Input: `"@": "uint256: 1"`
- Decoded Output: `{}`
- Logs: `[]`
- Raw Logs: `[]`

The right pane is the MetaMask extension's "Transaction request" screen. It shows the transaction details and interaction with the Sepolia test network. The transaction hash is `0x4034657e262e642d0fAbd0dea84c384a32151e44c1375622f101a56a903a`, and the transaction index is 26.

The screenshot shows the Remix IDE and the MetaMask extension integrated into a browser window. The Remix interface on the left displays a sidebar with deployment options like 'mint', 'renounceOwn...', 'setMinter', 'transfer', 'transferFrom', 'transferOwner...', 'allowance', 'balanceOf', and 'or: uint256: 1000000000'. Below this are sections for 'Low level interactions' and 'CALLDATA'. A message at the bottom states: 'Both 'receive'' and 'fallback' functions are not defined'. The main workspace shows a Solidity code editor with the following code:

```
require(block.timestamp < ev.deadline, "Event already finished");
require(outcomeIndex < ev.outcomeCount, "Invalid outcome");
require(msv.value == 0, "Bet must be > 0");
ev.totalPool += msv.value;
```

An 'Explain contract' button is present. The right side of the screen shows the MetaMask extension's 'Accounts' interface, which lists 'Account 1' with a balance of '\$0.00'. There is also a 'Hidden (1)' account entry. At the bottom right, there is a 'Add wallet' button.