

FINAL PROJECT

Team members: Kairkulova Dilnaz, Myrzakhan Mereke

PredictionChain – Decentralized Crowdfunding for Prediction Events

Github: <https://github.com/mavisges1/PredictionChain.git>

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 - PredictionMarket and RewardToken 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:

- RewardToken handles ERC-20 reward token minting.
- PredictionMarket 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 PredictionMarket 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 PredictionMarket 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 RewardToken contract follows the ERC-20 standard and includes setMinter() to restrict minting rights only to the PredictionMarket 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 0.8.31.
5. Deploy RewardToken with Injected Provider – MetaMask.
6. Deploy PredictionMarket, passing the token's address.
7. Call setMinter() on RewardToken to set PredictionMarket 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).

- 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.**

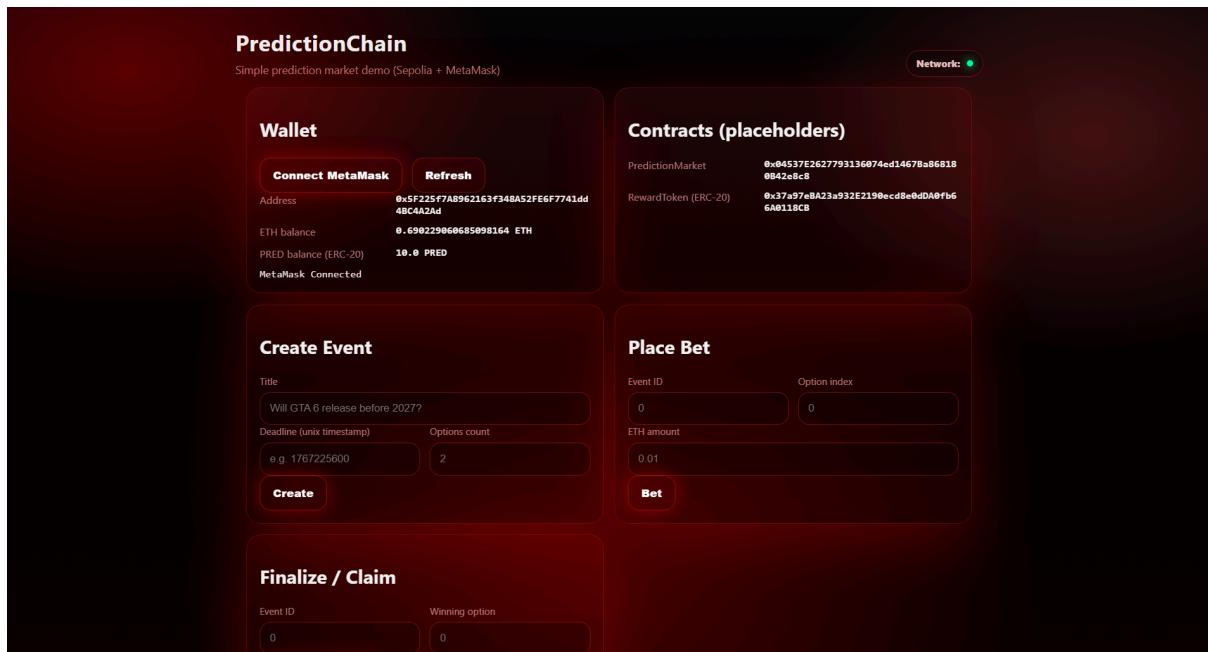
This ETH is only for testing and has no real monetary value.

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:** Solidity code for RewardToken.sol, which inherits from ERC20 and implements the mint function.
- MetaMask Extension:** Shows a transaction for "Contract deployment" from Account 3 to SepoliaETH. The transaction details are:

Status	Confirmed
From	Account 3
To	New contract
Nonce	27
Amount	-0 SepoliaETH
Gas Limit (Units)	1969239
Gas Used (Units)	1952575
Base fee (GWEI)	1.030147474
Priority fee (GWEI)	1.5
Total gas fee	0.00494 SepoliaETH
Max fee per gas	0.00000003 SepoliaETH
Total	0.0049403 SepoliaETH
- Activity Log:** Shows a single entry for the deployment transaction.

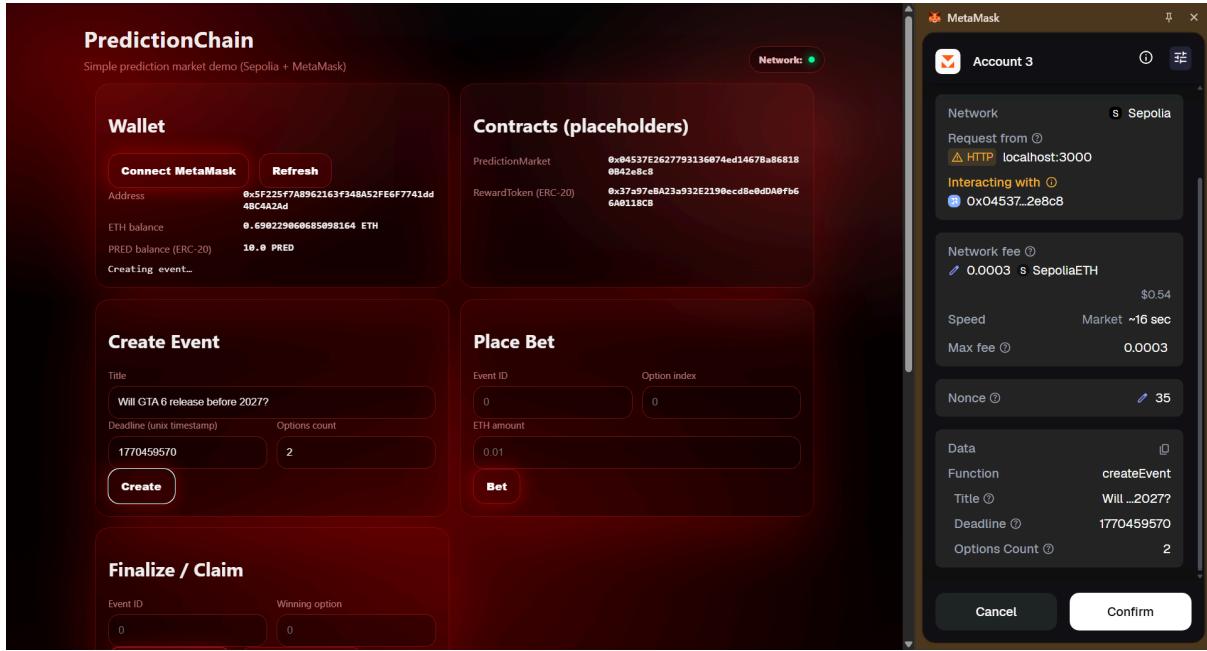
The screenshot shows the REMIX IDE interface with the following details:

- 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:** Solidity code for RewardToken.sol, identical to the first screenshot.
- MetaMask Extension:** Shows a transaction for "Set Market" from Account 3 to SepoliaETH. The transaction details are:

Status	Confirmed
From	Account 3
To	0x37a97...11BC
Nonce	28
Amount	-0 SepoliaETH
Gas Limit (Units)	44594
Gas Used (Units)	44216
Base fee (GWEI)	1.105769069
Priority fee (GWEI)	1.5
Total gas fee	0.000115 SepoliaETH
Max fee per gas	0.00000003 SepoliaETH
Total	0.00011522 SepoliaETH
- Activity Log:** Shows a single entry for the market creation transaction.

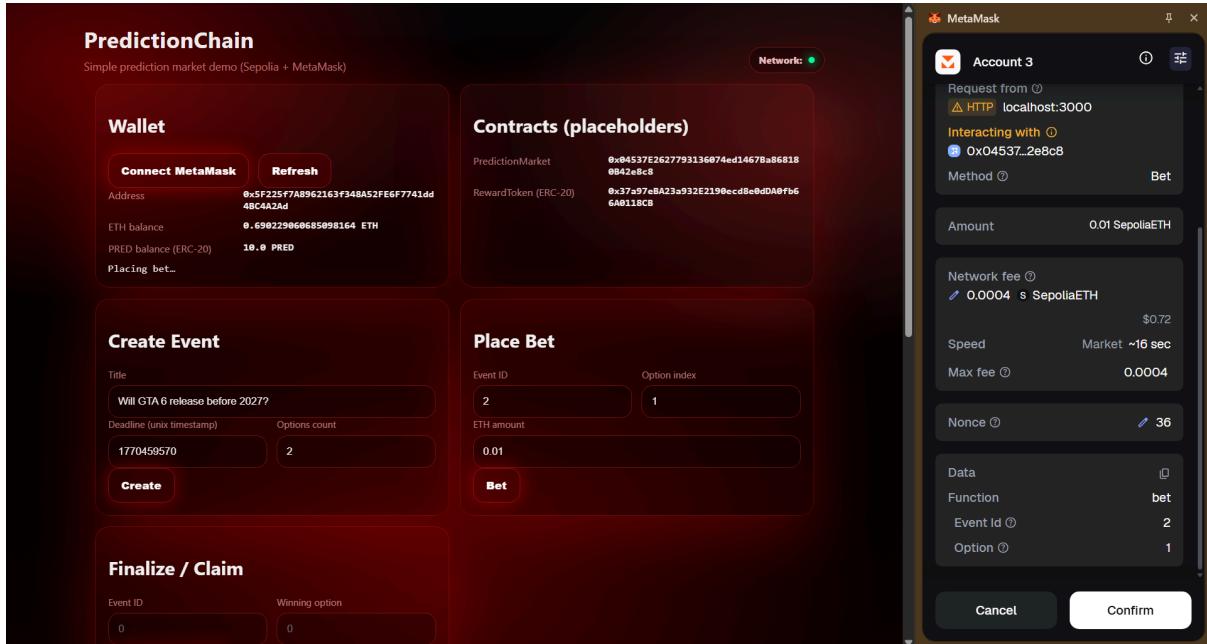
Screenshot 3 – Create Event Transaction

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



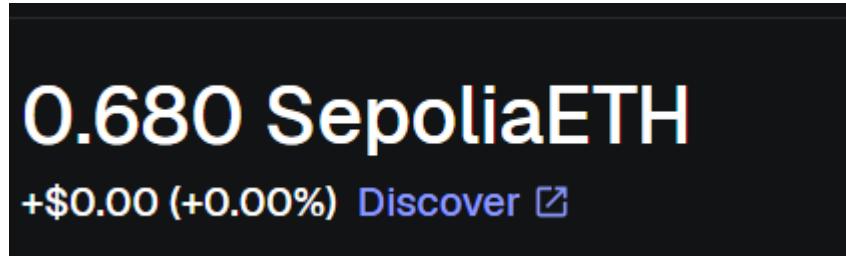
Screenshot 4 – Place Bet with ETH

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



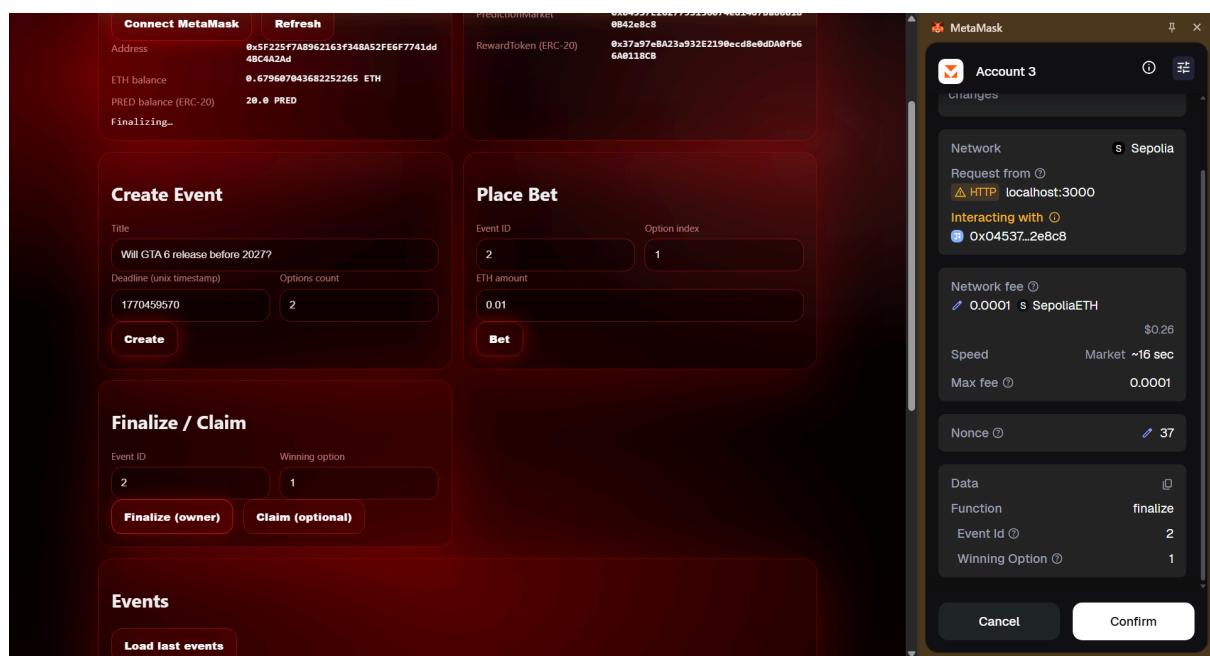
Screenshot 5 – ERC-20 Token Balance

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.



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: 0x04537219277751507450206752000510, 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

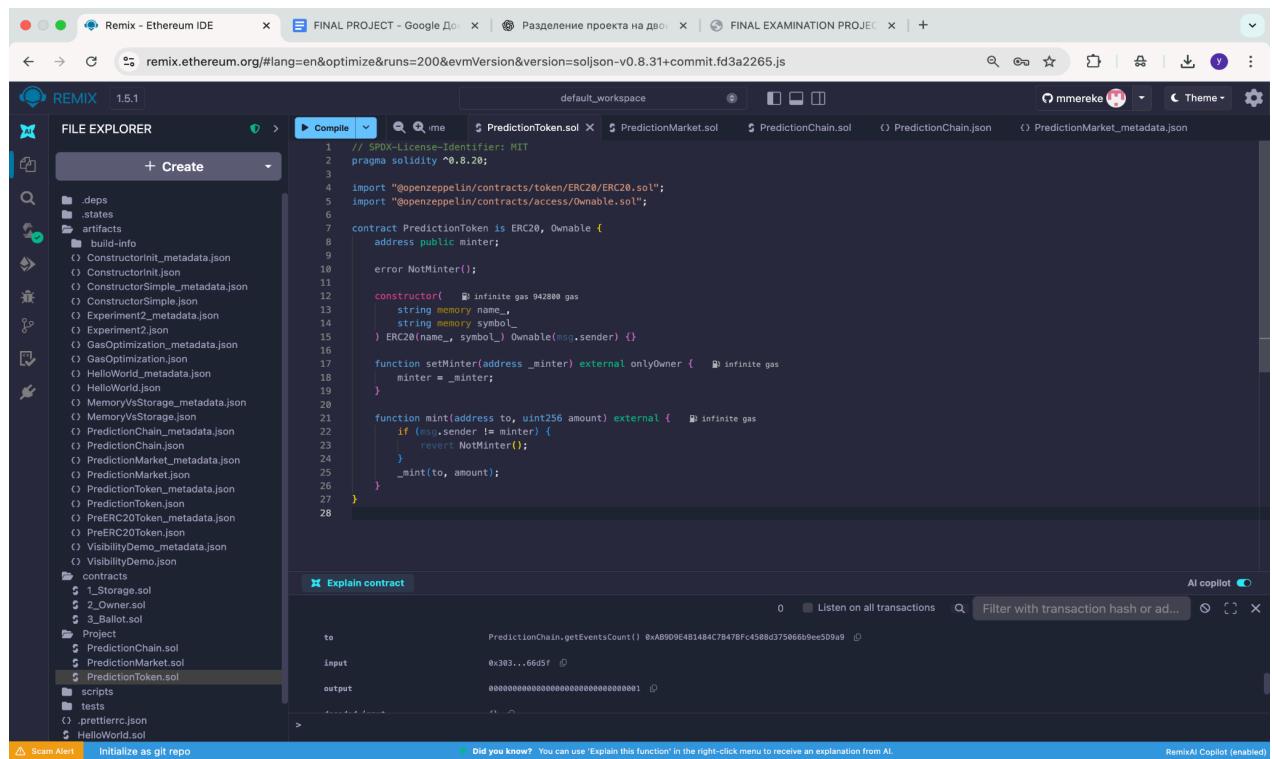
Claim Confirmed

-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 mint operation.

```
// 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 lists various files and contracts, including 'build-info', 'contracts', and specific files like '1_Storage.sol', '2_Owner.sol', and '3_Ballot.sol'. The central area displays the Solidity code for 'PredictionChain.sol':

```

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 }
9

```

Below the code, an 'Explain contract' panel is open, showing the function signature 'PredictionChain.getEventsCount()'. It displays the transaction hash '0xAB09E4B1484C70478Fc4508d375066b9ee509a9' and the output value '0x393...66d5f'. There are also buttons for 'Listen on all transactions', 'Filter with transaction hash or address...', and 'AI copilot'.

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.

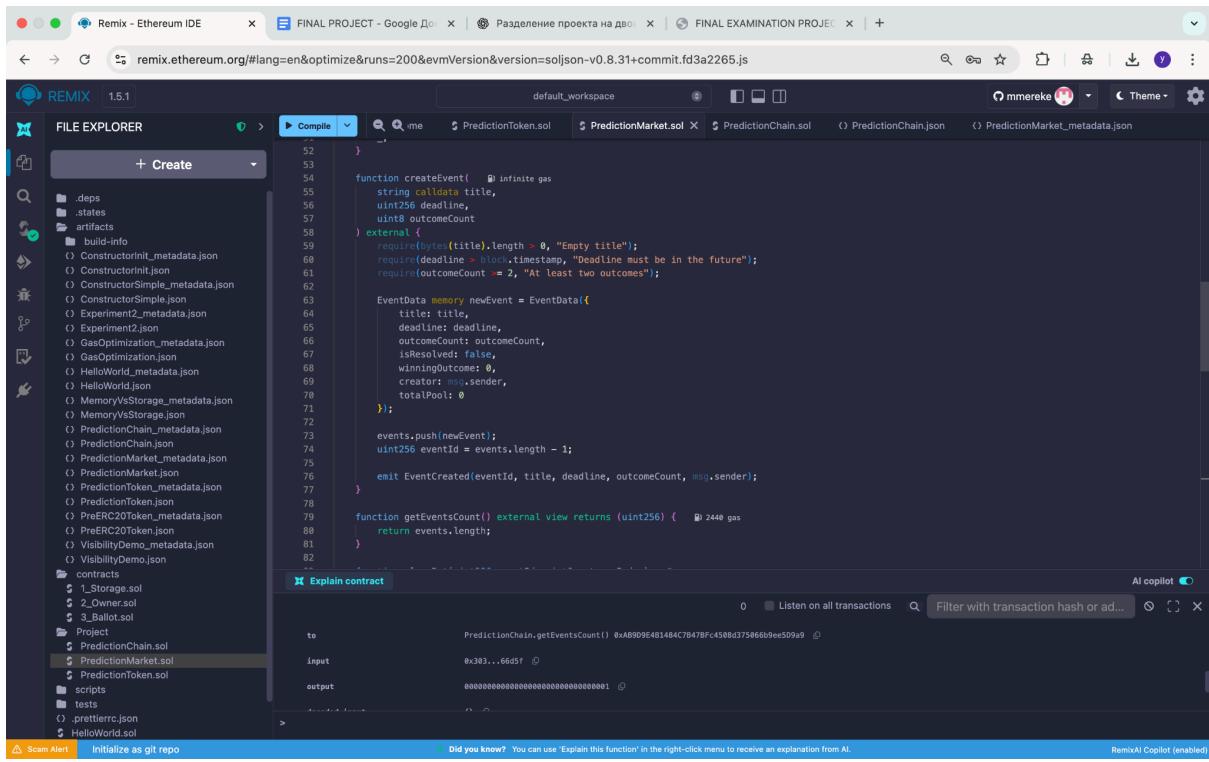
For every 1 wei contributed to an event, the user receives 1 PRED token.

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' that takes parameters like title, deadline, and outcomeCount, and returns an event object. It also includes logic to validate the title and deadline. Below the code editor, there's an 'Explain contract' section with details about the 'getEventsCount' function, which returns the number of events. The bottom status bar indicates a 'Scam Alert' and an 'Initialize as git repo' button.

```
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");

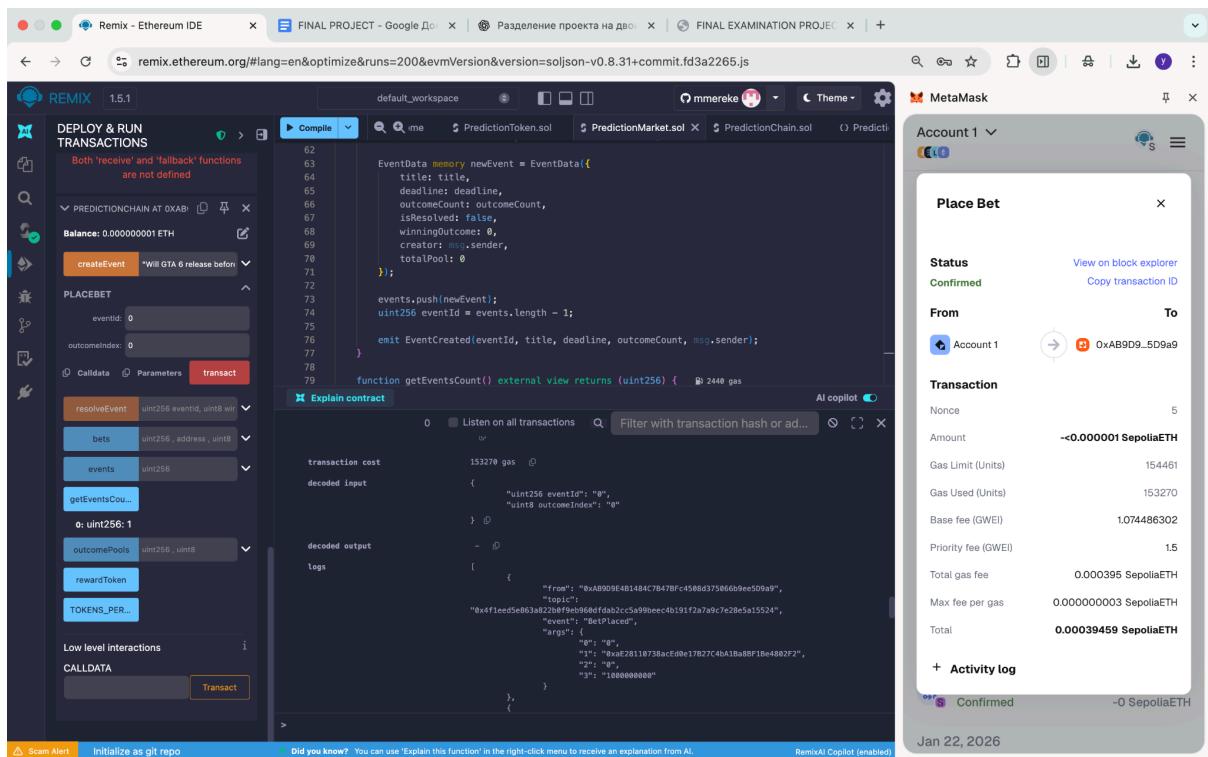
    EventData memory newEvent = EventData({
        title: title,
        deadline: deadline,
        outcomeCount: outcomeCount,
        isResolved: false,
        winningOutcome: 0,
        creator: msg.sender,
        totalPool: 0
    });

    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. It includes buttons for "Buy", "Swap", "Send", and "Receive". Below these are tabs for "Tokens", "DeFi", "NFTs", and "Activity". The "Activity" tab is selected, displaying a list 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.

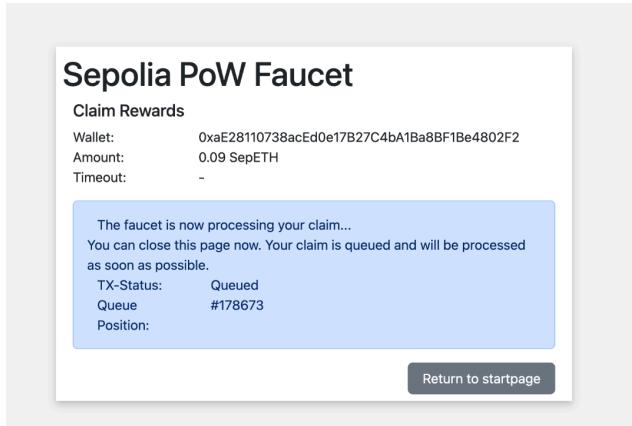
2.8. Sepolia Testnet Deploy

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 Explain contract feature enabled. The central panel displays the Solidity code for PredictionChain.sol, which interacts with PredictionMarket.sol. The 'Explain contract' sidebar provides detailed explanations for various contract functions and their interactions with the blockchain. The sidebar includes sections for status, transaction hash, block hash, block number, from, to, transaction cost, decoded input, decoded output, and logs. The logs section shows a log entry for a 'createEvent' function call, providing a breakdown of the arguments and their corresponding hex values. To the right of the IDE, the MetaMask extension is open, showing a history of transactions on the Sepolia test network, including confirmations for contract interactions and deployments.

This screenshot is similar to the one above but includes a call trace at the bottom of the central code editor. The call trace shows a sequence of function calls between PredictionChain and PredictionMarket contracts, with specific arguments and return values highlighted. This visualizes the flow of data between the two contracts during a transaction. The rest of the interface, including the Explain contract sidebar and the MetaMask extension, remains consistent with the first screenshot.

remix.ethereum.org/#lang=en&optimize&runs=200&evmVersion&version=soljson-v0.8.31+commit.fd3a2265.js

DEPLOY & RUN TRANSACTIONS

Both 'receive' and "fallback" functions are not defined.

PREDICTIONCHAIN AT 0xAB1...

PLACEBET

eventid: 0
outcomeIndex: 0

Calldata Parameters

resolvEvent uint256 eventid, uint8 val
bets uint256 address, uint8
events uint256

getEventsC...

o: uint256: 1
outcomePools uint256, uint8
rewardToken
TOKENS_PER_...

Low level interactions

CALDATA

Transact

Explain contract

0 Listen on all transactions Filter with transaction hash or ad...

```

0xa0d9e4b1484c7b47
import "./PredictionMarket.sol";
...
/// @title PredictionChain
/// @dev Thin wrapper around PredictionMarket for convenience / naming
call to PredictionChain.getEventsCount

    [call] from: 0xae28110738acd0e17b274b1ba8bf1be480f2
    to: PredictionChain.getEventsCount() data: 0x301...66df
    from: 0xa0d9e4b1484c7b47
    to: PredictionChain.getEventsCount() 0xA0D9E4B1484C7B47BF<508d375060b9ee509a9
    input: 0x301...66df
    output: 0000000000000000000000000000000000000000000000000000000000000001
    decoded input: 0
    decoded output: {
        "0": "uint256: 1"
    }
    logs: []
    raw logs: []
transact to PredictionChain.placeBet pending ...

```

Did you know? You can use 'Explain this function' in the right-click menu to receive an explanation from AI.

RemixAI Copilot (enabled)

MetaMask

Account 1

Transaction request

Estimated changes

Network Sepolia Request from remix.ethereum.org Interacting with 0xAB1D9D9...5D9a9

Amount <0.000001 SepoliaETH

Network fee

Speed Market ~12 sec

Cancel Confirm

remix.ethereum.org/#lang=en&optimize&runs=200&evmVersion&version=soljson-v0.8.31+commit.fd3a2265.js

DEPLOY & RUN TRANSACTIONS

mint address to, uint256 amount
renounceOwn...
setMinter 0xAB1D9D9E4B1484C7B47
transfer address to, uint256 value
transferFrom address from, address to
transferOwn... address newOwner
allowance address owner, address to
balanceOf 0x28110738acd0e17b274b1ba8bf1be480f2
o: uint256: 10000000000

Low level interactions

CALDATA

Transact

Both 'receive' and "fallback" functions are not defined.

PREDICTIONCHAIN AT 0xAB1...

Explain contract

0 Listen on all transactions Filter with transaction hash or ad...

```

90     require(block.timestamp > ev.deadline, "Event already finished");
91     require(outcomeIndex > ev.outcomeCount, "Invalid outcome");
92     require(msg.value > 0, "Bet must be > 0");
93     ev.totalPool += msg.value;
call to PredictionChain.placeBet pending ...
view on Etherscan view on Blockscout
[block:10198028 txIndex:12] from: 0xaE2...80F2
to: PredictionChain.placeBet(uint256,uint8) 0xAB1...5D9a9 value: 1000000000 wei
data: 0x03...00000 logs: 2 hash: x057...aa607
The method "debug_traceTransaction" does not exist / is not available.
The method "debug_traceTransaction" does not exist / is not available.
call to PredictionToken.balanceOf

    [call] from: 0xae28110738acd0e17b274b1ba8bf1be480f2
    to: PredictionToken.balanceOf(address) data: 0x70a...80F2
    from: 0xaE28110738acd0e17b274b1ba8bf1be480f2
    to: PredictionToken.balanceOf(address) 0x2ac98aab672a13f69fA5c939f36dd0daCf09f56
    input: 0x70a...80F2
    output: 0000000000000000000000000000000000000000000000000000000000000001
    decoded input: {
        "0": "address account": "0xaE28110738acd0e17b274b1ba8bf1be480f2"
    }
    decoded output: {
        "0": "uint256: 1000000000"
    }
    logs: []
    raw logs: []

```

Did you know? You can use 'Explain this function' in the right-click menu to receive an explanation from AI.

RemixAI Copilot (enabled)

MetaMask

Accounts

Search your accounts

Account 1 \$0.00

Add account

Hidden (1)

Add wallet