ZJU-blockchain-course-2025

⬆️ 可以修改成你自己的项目名。

> 第二次作业要求（以下内容提交时可以删除）：
>
> 进阶的去中心化彩票系统，参与方包括：竞猜玩家、公证人
>
> **背景**：传统的体育彩票系统（例如我国的体育彩票）一般没有彩票交易功能：例如，对于“NBA本赛季MVP为某球员/F1的赛季总冠军为某车队”这类持续时间长的事件的下注一般在赛季开始前就会买定离手，这使得一旦出现突发或不确定事件（如球员A赛季报销/球队B买入强力球星/C车队车手受伤等），很多玩家的选择便会立即失去意义，导致彩票游戏的可玩性下降。因此，一个有趣的探索方向是让彩票系统拥有合规、方便的交易功能。
>
> 建立一个进阶的去中心化彩票系统（可以是体育彩票，或其它任何比赛节目的竞猜，例如《中国好声音》《我是歌手》年度总冠军等，可以参考 Polymarket ），在网站中：
>
> - 公证人（你自己）可以创立许多竞猜项目：例如某场比赛的输赢、年度总冠军的得主等，每个项目应当有2个或多个可能的选项，一定的彩票总金额（由公证人提供），以及规定好的结果公布时间。
> - 玩家首先领取到测试所需以太币。在网站中，对于一个竞猜项目和多个可能的选项：
>   1. 每个竞彩玩家都可以选择其中的某个选项并购买一定金额（自己定义）的彩票，购买后该玩家会获得一张对应的彩票凭证（一个 ERC721 合约中的 Token）
>   2. 在竞彩结果公布之前，任何玩家之间可以买卖他们的彩票，以应对项目进行期间的任何突发状况。具体的买卖机制如下：一个玩家可以以指定的金额挂单出售（ERC721 Delegate）自己的彩票，其它玩家如果觉得该彩票有利可图就可以买入他的彩票。双方完成一次 ERC721 Token 交易。
>   3. 公证人可以在时间截止时（简单起见，你可以随时终止项目）输入竞猜的结果并进行结算。所有胜利的玩家可以平分奖池中的金额。
> - Bonus（最多5分，若想要完成，可以直接将功能整合进上述要求中）：
>   1. （2分）发行一个 ERC20 合约，允许用户领取 ERC20 积分，并使用ERC20积分完成上述流程。
>   2. （3分）对交易彩票的过程实现一个简单的链上订单簿：卖方用户可以以不同价格出售一种彩票，网页上显示当前订单簿的信息（多少价格有多少该彩票正在出售）。其他用户可以根据最优价格购买彩票。
> - 可以对上述需求进行合理更改和说明。请大家专注于功能实现，网站UI美观程度不纳入评分标准，能让用户能够舒适操作即可。

以下内容为作业仓库的README.md中需要描述的内容。请根据自己的需要进行修改并提交。

作业提交方式为：**提交视频文件**和**仓库的链接**到指定邮箱。

EasyBet：一个基于区块链分叉的竞猜 DApp

课程: 区块链与数字货币
姓名: 王建皓
学号: 3230105346
日期: 2025年11月4日

项目介绍

本项目是一个名为 "EasyBet" 的去中心化应用（dApp）。它构建在以太坊区块链（或兼容的 EVM 链）上，旨在提供一个完整的竞猜平台。

项目包含三个核心智能合约：

- `BetToken.sol`: ERC20 代币合约，用于平台内交易
- `TicketNFT.sol`: ERC721 NFT 合约，代表竞猜门票
- `EasyBet.sol`: 主合约，处理竞猜逻辑和订单簿系统

前端使用 React 和 TypeScript 构建，用户可以通过它与智能合约进行交互。本项目利用 Hardhat 作为开发环境，在本地分叉网络上进行合约的部署和测试。

功能介绍

公证人 (Owner)

- 创建竞猜项目（设置标题、描述、选项、票价、最大票数、持续时间）
- 结算竞猜项目（设置获胜选项，分发奖金）
- 查看所有竞猜项目及其状态

玩家 (User)

- 连接 MetaMask 钱包
- 领取测试代币
- 查看当前活跃的竞猜项目
- 购买竞猜门票（选择选项）
- 在二级市场挂单出售门票
- 查看订单簿（按项目分类的挂单信息）
- 购买他人挂单的门票
- 自动领取奖金（获胜者按票数比例分配）

功能实现分析（含订单簿 Bonus +3）

智能合约架构

1. BetToken.sol (ERC20 代币合约)

solidity

```
contract BetToken is ERC20, Ownable {
    mapping(address => bool) public hasClaimed;

    function claimTokens() external {
        require(!hasClaimed[msg.sender], "User has already claimed tokens");
        hasClaimed[msg.sender] = true;

        uint256 amount = 100 * 10 ** decimals();
        _mint(msg.sender, amount);
    }
}
```

- 实现 ERC20 标准代币
- 提供代币领取功能，每个地址可领取 100 个代币
- 只有合约所有者可以额外铸造代币

2. TicketNFT.sol (ERC721 NFT 合约)

solidity

```
contract TicketNFT is ERC721, Ownable {
    struct TicketInfo {
        uint256 projectId;
        uint256 optionIndex;
        uint256 purchasePrice;
        uint256 purchaseTime;
    }

    mapping(uint256 => TicketInfo) public ticketInfo;
    mapping(uint256 => uint256[]) public projectTickets;
    mapping(address => uint256[]) public ownerTickets;
}
```

- 实现 ERC721 标准，每个门票都是唯一的 NFT
- 记录门票的详细信息：所属项目、选择选项、购买价格、购买时间
- 维护项目门票列表和用户门票列表，便于查询
- 重写 transfer 函数来跟踪所有权变化

3. EasyBet.sol (主合约)

solidity

```
contract EasyBet is Ownable, ReentrancyGuard {
    using Counters for Counters.Counter;

    Counters.Counter private _projectCounter;
    Counters.Counter private _orderCounter;
}
```

- 继承 `Ownable` 和 `ReentrancyGuard` 确保安全性和权限控制
- 使用计数器管理项目和订单ID

核心数据结构

项目结构 (Project)

solidity

```
struct Project {
    uint256 id;
    string title;
    string description;
    string[] options;
    uint256 ticketPrice;
    uint256 totalPrize;
    uint256 maxTickets;
    uint256 soldTickets;
    uint256 endTime;
    address creator;
    ProjectStatus status;
    uint256 winningOption;
}

enum ProjectStatus { Active, Settled, Cancelled }
```

订单结构 (Order)

solidity

```
struct Order {
    uint256 id;
    uint256 ticketId;
    uint256 projectId;
    address seller;
    uint256 price;
    uint256 createTime;
    bool active;
}
```

核心功能实现

1. 项目创建

solidity

```
function createProject(
    string memory _title,
    string memory _description,
    string[] memory _options,
    uint256 _ticketPrice,
    uint256 _maxTickets,
    uint256 _duration
) external returns (uint256) {
    require(_options.length >= 2, "At least two options are required");
    require(_ticketPrice > 0, "Ticket price must be greater than zero");

    uint256 totalPrize = _ticketPrice * _maxTickets;
    require(
        IERC20(betToken).transferFrom(msg.sender, address(this), totalPrize),
        "Transfer of prize tokens failed"
    );

    // 创建项目逻辑...
    emit ProjectCreated(projectId, msg.sender, _title, totalPrize);
}
```

- 创建者需要预存总奖池金额
- 设置项目基本信息和时间限制
- 发射事件通知前端

2. 门票购买

solidity

```
function purchaseTicket(uint256 _projectId, uint256 _optionIndex)
    external
    nonReentrant
    returns (uint256)
{
    // 检查项目状态和时间
    require(project.status == ProjectStatus.Active, "Project is not active");
    require(block.timestamp < project.endTime, "Project has ended");

    // 支付门票费用
    require(
        IERC20(betToken).transferFrom(msg.sender, address(this), project.ticketPrice),
        "Transfer of ticket price failed"
```

```
    );

    // 铸造NFT门票
    uint256 ticketId = ITicketNFT(ticketNFT).mintTicket(
        msg.sender,
        _projectId,
        _optionIndex,
        project.ticketPrice
    );

    project.soldTickets++;
    project.totalPrize += project.ticketPrice;
}
```

3. 订单簿系统（Bonus +3）

挂单功能:

solidity

```
function listTicket(uint256 _ticketId, uint256 _price) external nonReentrant {
    require(ITicketNFT(ticketNFT).ownerOf(_ticketId) == msg.sender, "Not ticket owner");
    require(_price > 0, "Price must be greater than zero");
    require(ticketToOrder[_ticketId] == 0, "Ticket already listed");

    // 创建订单记录
    orders[orderId] = Order({
        id: orderId,
        ticketId: _ticketId,
        projectId: projectId,
        seller: msg.sender,
        price: _price,
        createTime: block.timestamp,
        active: true
    });

    // 维护索引
    projectOrders[projectId].push(orderId);
    userOrders[msg.sender].push(orderId);
    ticketToOrder[_ticketId] = orderId;
}
```

购买挂单:

solidity

```
function buyListedTicket(uint256 _orderId) external nonReentrant {
    Order storage order = orders[_orderId];
    require(order.active, "Order is not active");
```

```
    require(order.seller != msg.sender, "Cannot buy your own ticket");

    // 支付代币给卖家
    require(
        IERC20(betToken).transferFrom(msg.sender, order.seller, order.price),
        "Payment transfer failed"
    );

    // 转移NFT所有权
    ITicketNFT(ticketNFT).transferFrom(order.seller, msg.sender, order.ticketId);

    // 更新订单状态
    order.active = false;
    delete ticketToOrder[order.ticketId];
}
```

4. 项目结算和奖金分发

solidity

```
function settleProject(uint256 _projectId, uint256 _winningOption) external onlyOwner {
    project.status = ProjectStatus.Settled;
    project.winningOption = _winningOption;

    // 统计获胜票数
    uint256 winningTicketsCount = 0;
    for (uint256 i = 0; i < ticketIds.length; i++) {
        (, uint256 optionIndex,,) = ITicketNFT(ticketNFT).getTicketInfo(ticketIds[i]);
        if (optionIndex == _winningOption) {
            winningTicketsCount++;
        }
    }

    // 按比例分发奖金
    if (winningTicketsCount > 0) {
        uint256 prizePerTicket = project.totalPrize / winningTicketsCount;

        for (uint256 i = 0; i < ticketIds.length; i++) {
            (, uint256 optionIndex,,) = ITicketNFT(ticketNFT).getTicketInfo(ticketIds[i]);
            if (optionIndex == _winningOption) {
                address winner = ITicketNFT(ticketNFT).ownerOf(ticketIds[i]);
                require(
                    IERC20(betToken).transfer(winner, prizePerTicket),
                    "Prize transfer failed"
                );
                emit PrizeDistributed(_projectId, winner, prizePerTicket);
            }
        }
    }
}
```

前端实现要点

1. 钱包连接和合约初始化

typescript

```
const connectWallet = async () => {
    const web3Provider = new ethers.providers.Web3Provider((window as any).ethereum);
    await web3Provider.send("eth_requestAccounts", []);
    const web3Signer = web3Provider.getSigner();
    const userAccount = await web3Signer.getAddress();

    // 初始化合约实例
    const betTokenContract = new ethers.Contract(betTokenAddress, betTokenABI, web3Signer);
    const easyBetContract = new ethers.Contract(easyBetAddress, easyBetABI, web3Signer);
    const ticketNFTContract = new ethers.Contract(ticketNFTAddress, ticketNFTABI,
web3Signer);
};
```

2. 数据获取和状态管理

typescript

```
const fetchData = async () => {
    // 获取活跃项目
    const activeProjects = await easyBetContract.getActiveProjects();

    // 获取项目详情
    for (let projectId of activeProjects) {
        const project = await easyBetContract.getProject(projectId);

        // 获取项目订单簿
        const orderIds = await easyBetContract.getProjectOrderBook(projectId);
        const orders = await Promise.all(
            orderIds.map(orderId => easyBetContract.getOrder(orderId))
        );

        // 获取票数统计
        const ticketStats = await easyBetContract.getProjectTicketStats(projectId);
    }
};
```

3. 交易处理

typescript

```
const handleTransaction = async (transactionPromise: Promise<any>, successMessage: string)
=> {
    try {
        setLoading(true);
        const tx = await transactionPromise;
        await tx.wait();
        setSuccessMessage(successMessage);
        await fetchData(); // 刷新数据
    } catch (error) {
        setErrorMessage(`Transaction failed: ${error.message}`);
    } finally {
        setLoading(false);
    }
};
```

如何运行

环境要求

- Node.js (v16+)
- Ganache CLI 或 Ganache GUI
- MetaMask 浏览器扩展
- Hardhat 开发框架

部署步骤

1. 启动本地区块链

bash

```
# 使用 Ganache
npx ganache-cli -d -i 1337 --chainId 1337
# 或使用 Ganache GUI，设置端口为 8545
```

2. 配置 Hardhat

javascript

```
// hardhat.config.js
module.exports = {
  networks: {
    ganache: {
      url: "http://127.0.0.1:8545",
      chainId: 1337,
      accounts: [privateKey1, privateKey2] // 从 Ganache 获取
    }
  }
};
```

3. 编译合约

bash

```
cd contracts
npm install
npx hardhat compile
```

npm install 这里遇到了问题，有一些包会找不到，所以需要手动intall一个版本。还有后面import的ether也需要自己安装@5版本。

4. 部署合约

创建部署脚本 `scripts/deploy.js`:

javascript

```
async function main() {
    // 部署 BetToken
    const BetToken = await ethers.getContractFactory("BetToken");
    const betToken = await BetToken.deploy();
    await betToken.deployed();
    console.log("BetToken deployed to:", betToken.address);

    // 部署 TicketNFT
    const TicketNFT = await ethers.getContractFactory("TicketNFT");
    const ticketNFT = await TicketNFT.deploy();
    await ticketNFT.deployed();
    console.log("TicketNFT deployed to:", ticketNFT.address);

    // 部署 EasyBet
    const EasyBet = await ethers.getContractFactory("EasyBet");
    const easyBet = await EasyBet.deploy(betToken.address, ticketNFT.address);
    await easyBet.deployed();
    console.log("EasyBet deployed to:", easyBet.address);

    // 设置 TicketNFT 的所有者为 EasyBet 合约
    await ticketNFT.transferOwnership(easyBet.address);
    console.log("TicketNFT ownership transferred to EasyBet");
```

```
}

main().catch((error) => {
    console.error(error);
    process.exitCode = 1;
});
```

运行部署:

bash

```
npx hardhat run scripts/deploy.ts --network ganache
```

5. 配置前端

bash

```
cd frontend
npm install
```

配置合约地址和 ABI:

javascript

```
// src/config.js
export const CONTRACT_ADDRESSES = {
    betToken: "0x...",
    easyBet: "0x...",
    ticketNFT: "0x..."
};

// 从 artifacts 复制 ABI
export const CONTRACT_ABIS = {
    betToken: [...],
    easyBet: [...],
    ticketNFT: [...]
};
```

6. 启动前端

bash

```
npm start
```

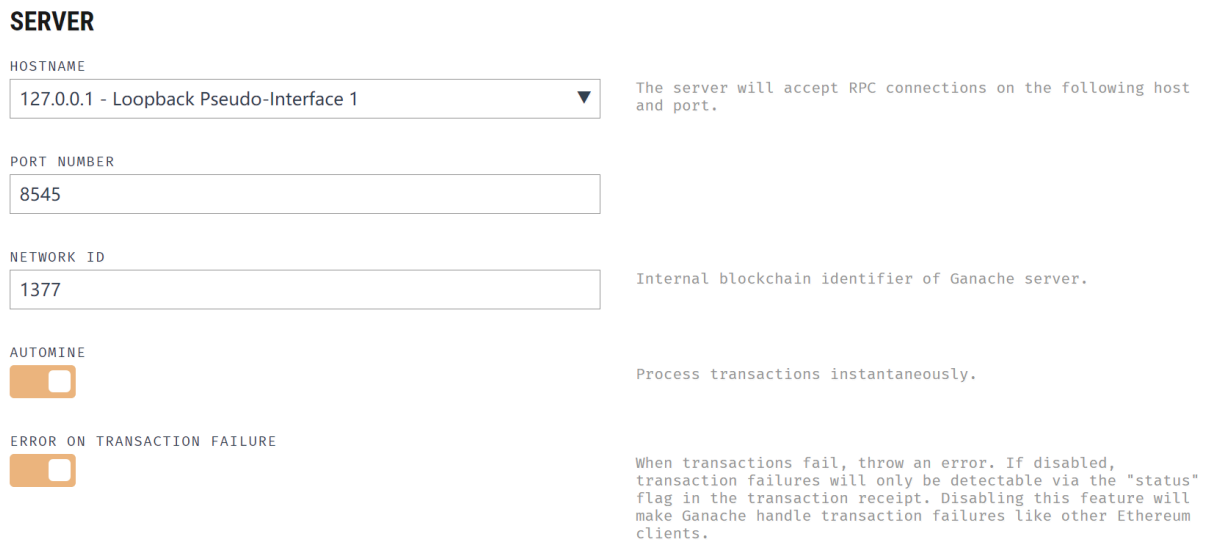
7. 配置 MetaMask

- 添加自定义网络：http://localhost:8545，链ID: 1337
- 导入 Ganache 账户私钥
- 添加自定义代币（BetToken）

测试流程

1.关于合约的部署：

- 打开Ganache,并配置URL以及chain ID：



- 运行

```
npx hardhat run scripts/deploy.ts --network ganache
```

部署合约，然后记下各个合约的地址:

```
PS D:\BET\contracts> npx hardhat run scripts/deploy.ts --network ganache
WARNING: You are currently using Node.js v18.20.8, which is not supported by Hardhat. This can lead to unexpected behavior. See https://v2.hardhat.org/nodejs-versions


🚀 开始部署 EasyBet 系统...
✅ 成功连接到网络: ganache
📦 当前区块: 0
📝 部署者地址: 0xbD2aC257307773a3851Ab1FFaFA6e2de3d91DbEB

1. 📦 部署 BetToken (ERC20)...
   ✅ BetToken 部署成功: 0xBBC3c3f22A956174374Ce7B9138be5cBF775D865

2. 🎫 部署 TicketNFT...
   ✅ TicketNFT 部署成功: 0xB563A702803f2774Dd966340B37DDdB30F877215

3. 🎲 部署 EasyBet 主合约...
   ✅ EasyBet 部署成功: 0x8728A7c47D5E1AfAa2339dfbd63571469aF8bb57

4. 🔄 设置 TicketNFT 所有者...
   ✅ 所有权转移成功

🎉 部署完成！合约地址:
   BetToken:    0xBBC3c3f22A956174374Ce7B9138be5cBF775D865
   TicketNFT:   0xB563A702803f2774Dd966340B37DDdB30F877215
   EasyBet:     0x8728A7c47D5E1AfAa2339dfbd63571469aF8bb57
   部署者:      0xbD2aC257307773a3851Ab1FFaFA6e2de3d91DbEB

💾 合约地址已保存到 deployed-addresses.json
```

2. 配置前端：

- 进入 ./frontend 目录。
- 打开 src/App.tsx 文件。
- 找到const CONTRACT_ADDRESSES = {...}; 这一行。
- 将 之前的合约地址分别替换为上一步复制的合约地址。
- **(重要)** 找到 const contractsABI = [...]。将 ./contracts/artifacts/contracts目录下各个合约中的 abi 数组内容复制并对应替换掉 App.tsx 中的 [...] 部分。

```
// 合约地址（部署后更新）
const CONTRACT_ADDRESSES = {
    betToken: '0xBBC3c3f22A956174374Ce7B9138be5cBF775D865', // 部署后替换
    ticketNFT: '0xB563A702803f2774Dd966340B37DDdB30F877215', // 部署后替换
    easyBet: '0x8728A7c47D5E1AfAa2339dfbd63571469aF8bb57'    // 部署后替换
};
```

- 之后在 ./frontend 目录下启动（需要提前在该目录下运行npm install）：

```
npm start
```

```
PS D:\BET\frontend> npm start

> frontend@0.1.0 start
> react-scripts start

(node:28276) [DEP_WEBPACK_DEV_SERVER_ON_AFTER_SETUP_MIDDLEWARE] DeprecationWarning: 'onAfterSetupMiddleware' option is d
eprecated. Please use the 'setupMiddlewares' option.
(Use `node --trace-deprecation ...` to show where the warning was created)
(node:28276) [DEP_WEBPACK_DEV_SERVER_ON_BEFORE_SETUP_MIDDLEWARE] DeprecationWarning: 'onBeforeSetupMiddleware' option is
 deprecated. Please use the 'setupMiddlewares' option.
Starting the development server...
Compiled successfully!

You can now view frontend in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.106.1:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
No issues found.
```

接着进入到:

3. 代币领取

- 连接钱包(需要提前登录metaMask,并在metamask中新建Ganache网络：并导入Ganache 里的秘钥)：

Ganache

网络名称

Ganache

根据我们的记录，该网络名称可能与此链 ID 不匹配。

建议名称： Localhost 8545

默认 RPC（远程过程调用）URL

127.0.0.1:8545

链 ID

1337

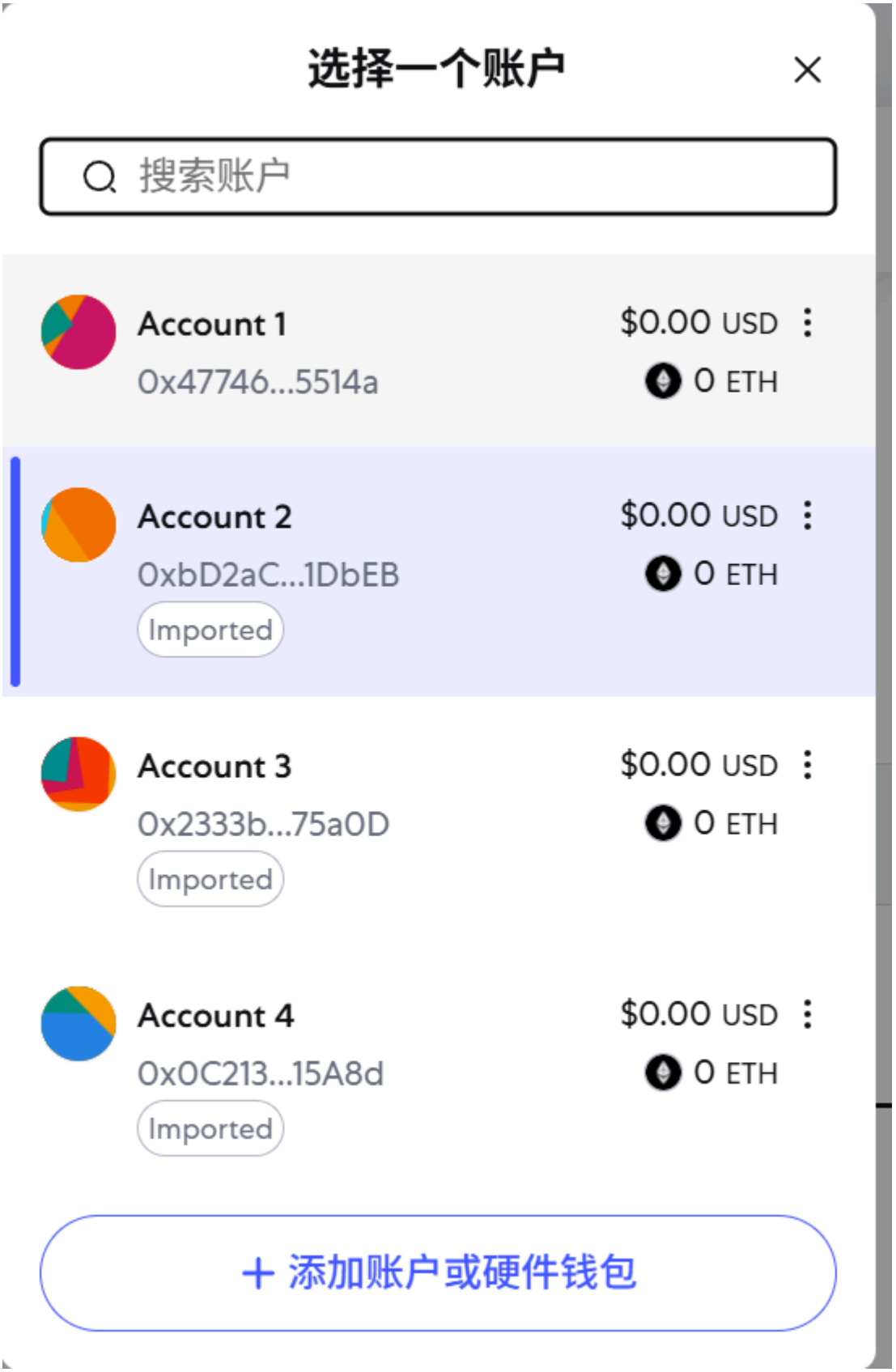
货币符号

ETH

区块浏览器 URL

添加 URL

保存



- 点击 "Claim Tokens" 领取测试代币

- 在 MetaMask 中确认交易

4. 创建竞猜项目（公证人）

- 切换到公证人账户
- 填写项目信息：标题、描述、选项、票价、最大票数、持续时间（注意最大票数*票价需要小于公证人金额）：

- 支付初始奖池金额
- 确认交易

由于操作简便一点，先后续内容会通过视频展示：

5. 购买门票（玩家）

- 切换到玩家账户
- 选择活跃项目
- 选择竞猜选项和购买数量
- 支付门票费用

- 获得 NFT 门票

6. 二级市场交易

- 查看订单簿中的挂单信息
- 挂单出售：设置价格，确认挂单
- 购买挂单：选择挂单，支付价格，获得门票所有权

7. 项目结算

- 公证人设置获胜选项
- 系统自动计算并分发奖金
- 玩家自动收到奖金