README

声明

该项目仅用于教学用途，不存在商业用途。

项目背景

基于社区发展和学员学习进阶需要，C2N社区推出启动台项目。整体项目定位是社区基础项目发行平台。
项目除了满足学习用途，更加鼓励同学在平台贡献自己的智慧和代码。

项目发展一共分为三个阶段：

第一阶段：学习和任务阶段，C2N技术团队和社区同学一起迭代该项目（4-5月份）

第二阶段：社区内部项目孵化阶段，满足社区同学发挥团队的创造力（6月份开始）

第三阶段：外部合作和开源发展阶段（待定）

演示地址

https://c2-n-launchpad.vercel.app/

产品需求

内部版本没有kyc，注册流程
C2N launchpad是一个区块链上的一个去中心化发行平台，专注于启动和支持新项目。它提供了一个平台，允许新的和现有的项目通过代币销售为自己筹集资金，同时也为投资者提供了一个参与初期项目投资的机会。下面是C2N launchpad产品流程的大致分析：

1. 项目申请和审核

- 申请：项目方需要在C2N launchpad上提交自己项目的详细信息，包括项目介绍、团队背景、项目目标、路线图、以及如何使用筹集的资金等。
- 审核：C2N launchpad团队会对提交的项目进行审核，评估项目的可行性、团队背景、项目的创新性、以及社区的兴趣等。这一过程可能还包括与项目方的面对面或虚拟会议。

2. 准备代币销售

- 设置条款：一旦项目被接受，C2N launchpad和项目方将协商代币销售的具体条款，包括销售类型（如公开销售或种子轮）、价格、总供应量、销售时间等。

- 准备市场：同时，项目方需要准备营销活动来吸引潜在的投资者。C2N launchpad也可能通过其平台和社区渠道为项目提供曝光。

3. KYC和白名单

- KYC验证：为了符合监管要求，参与代币销售的投资者需要完成Know Your Customer（KYC）验证过程。
- 白名单：完成KYC的投资者可能需要被添加到白名单中，才能在代币销售中购买代币。

4. 代币销售

- 销售开启：在预定时间，代币销售开始。根据销售条款，投资者可以购买项目方的代币。
- 销售结束：销售在达到硬顶或销售时间结束时关闭。

5. 代币分发

- 代币分发：销售结束后，购买的代币将根据约定的条款分发给投资者的钱包。

用户质押平台币，获得参与项目IDO的购买权重，后端配置项目信息并操作智能合约生成新的sale，用户在sale开始之后进行购买，项目结束后，用户进行claim

平台流程参考

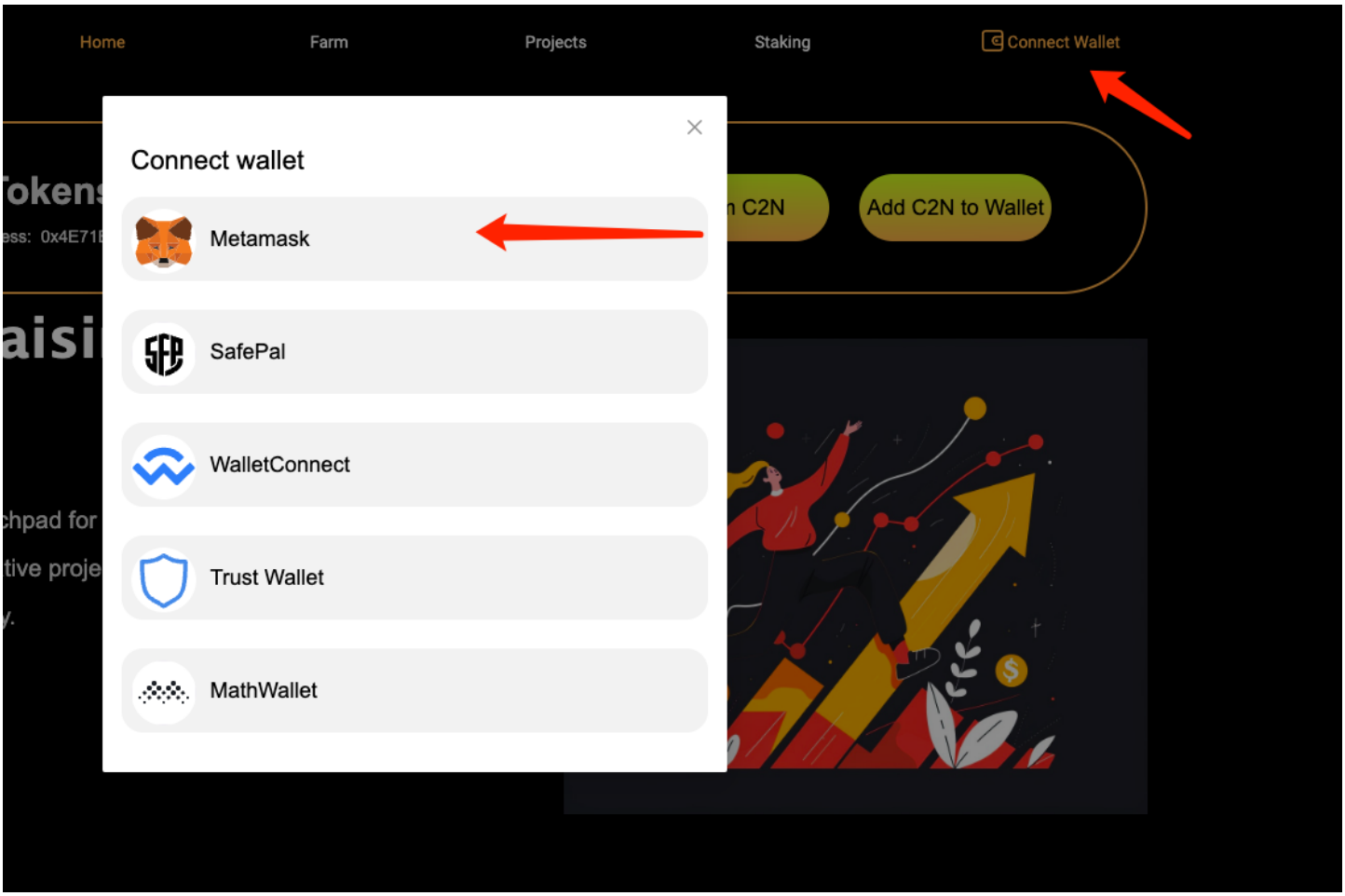
https://medium.com/avalaunch/avalunch-tutorials-platform-overview-1675547b5aff

功能操作

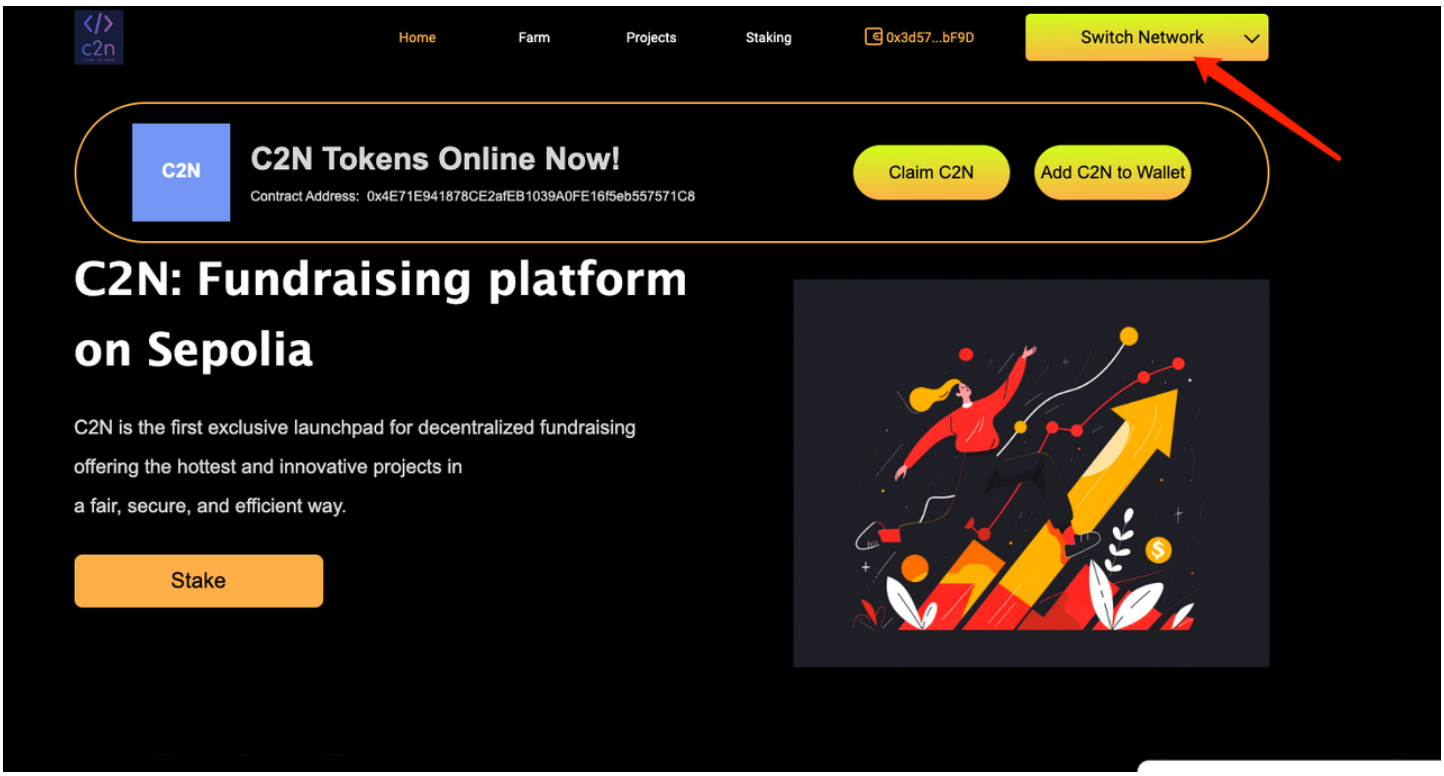
初始化

要进行下面的流程，需要提前准备sepolia的测试代币作为gas

1. 连接钱包(推荐metamask)

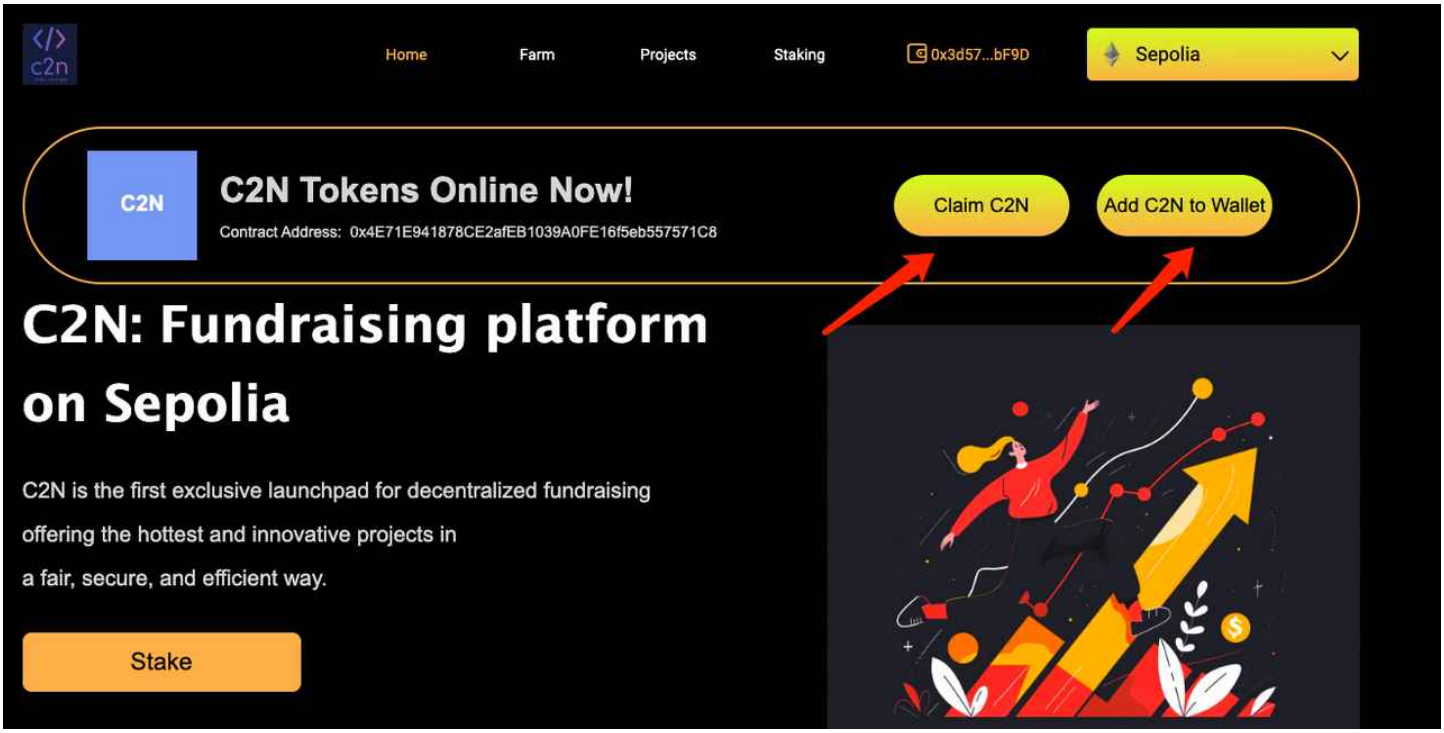


2.切换网络到sepolia，或者可以直接在钱包里面进行切换

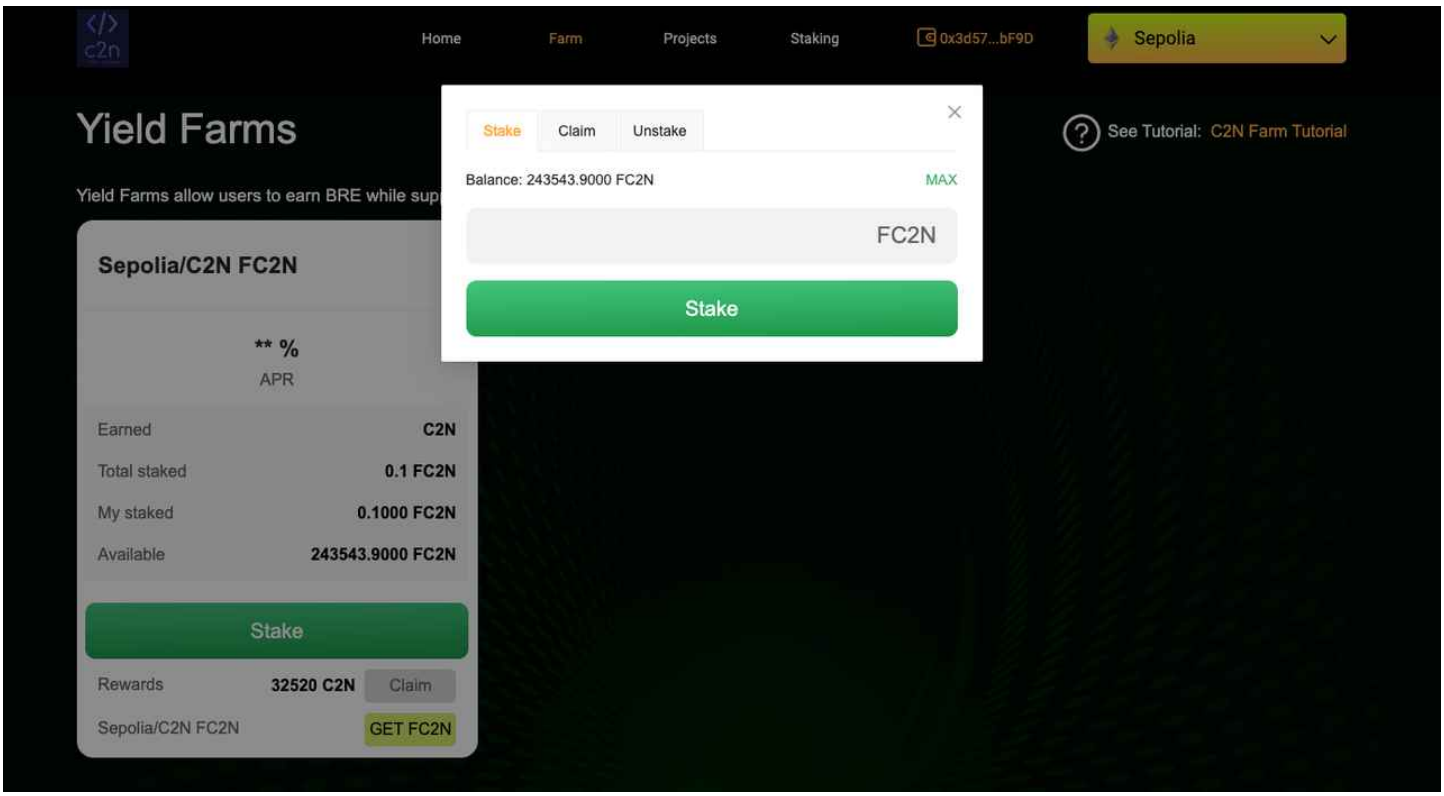


Farm 流程

1. Farm 流程需要用到我们的Erc20测试代币C2N, 可以在首页领取C2N(一个账户只能领取一次),并且添加到我们metamask，添加之后我们可以在metamask 看到我们领取的C2N 代币

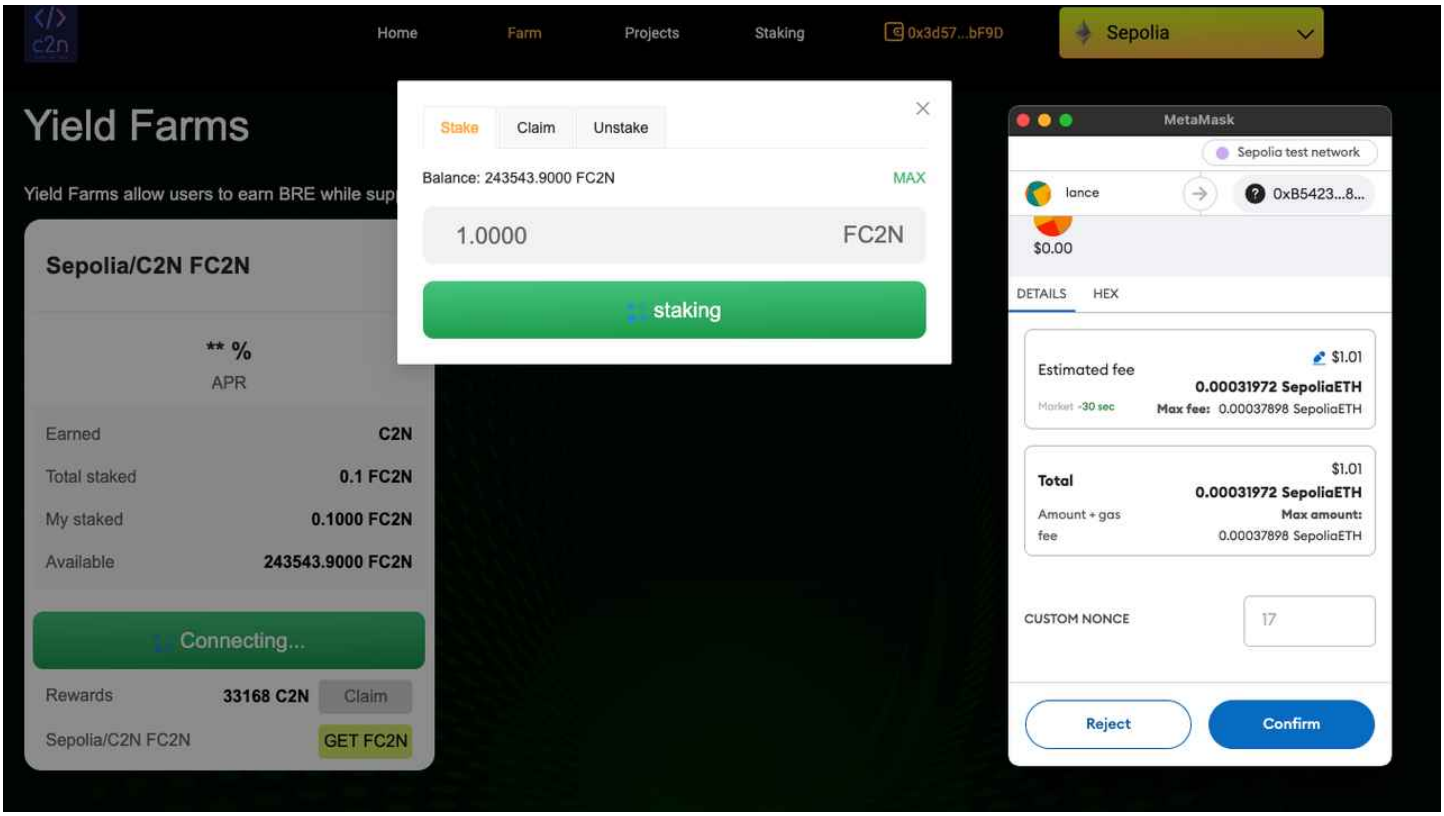


1. 在我们farm界面，我们可以质押fc2n 代币获取c2n, (方便大家操作，我们的测试网fc2n，c2n 是在上一步中领取的同一代币)，在这里我们有三个操作，stake:质押，unstake(witthdraw):撤回质押， 以及 claim:领取奖励;

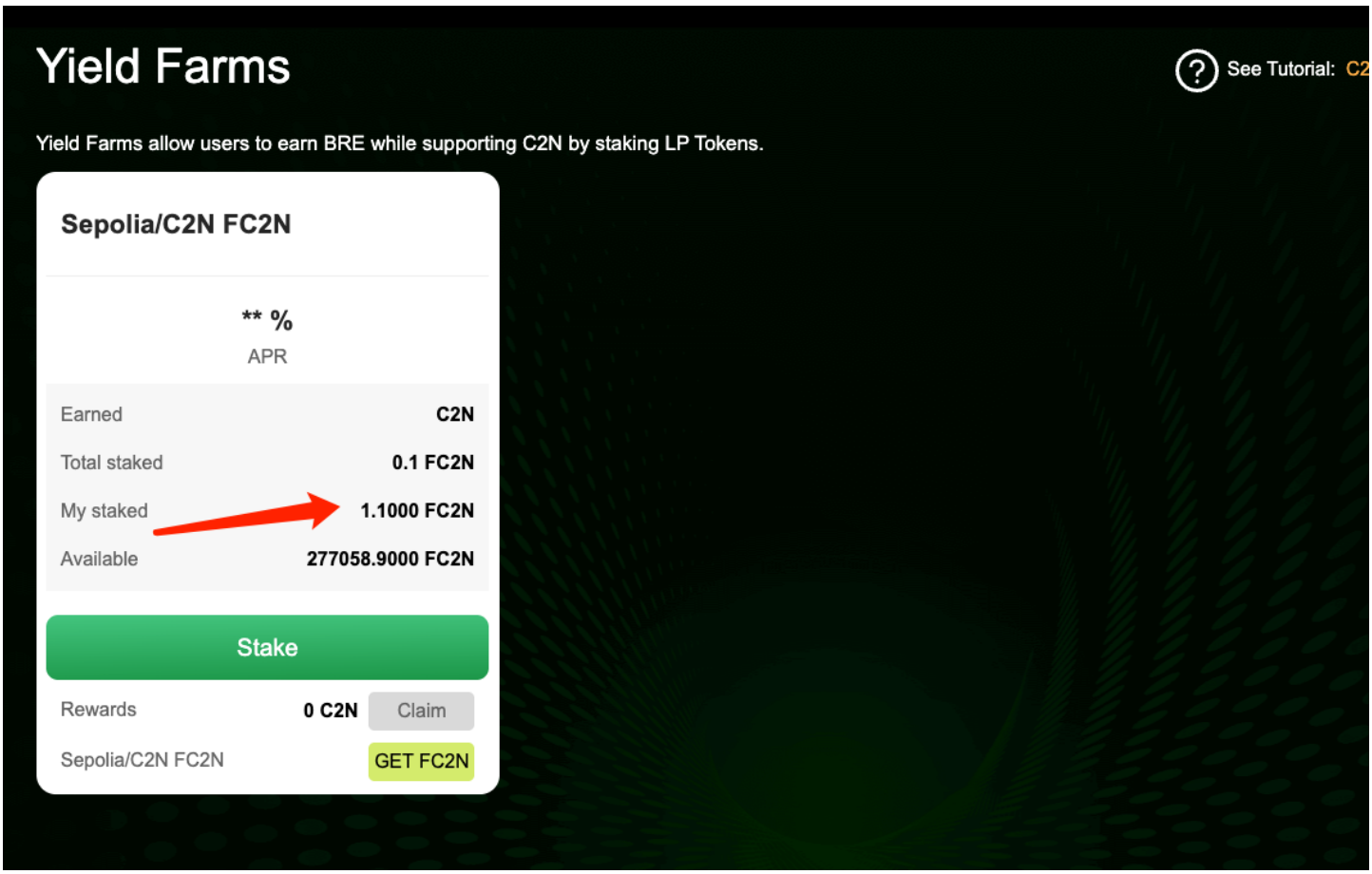


点击stake 或者claim 进入对应的弹窗，切换tab可以进行对应的操作;

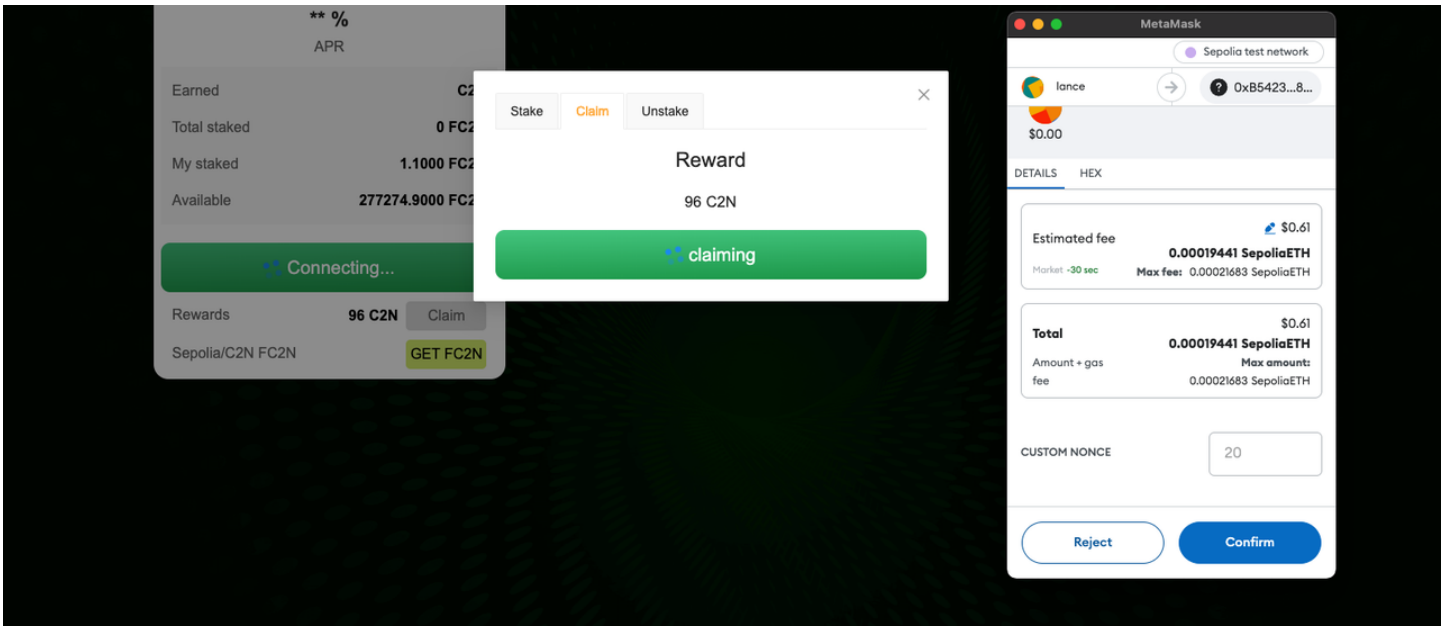
3. Stake ，输入要质押的FC2N代币数量，点击stake 会唤起钱包，在钱包中confirm，然后等待交易完成;



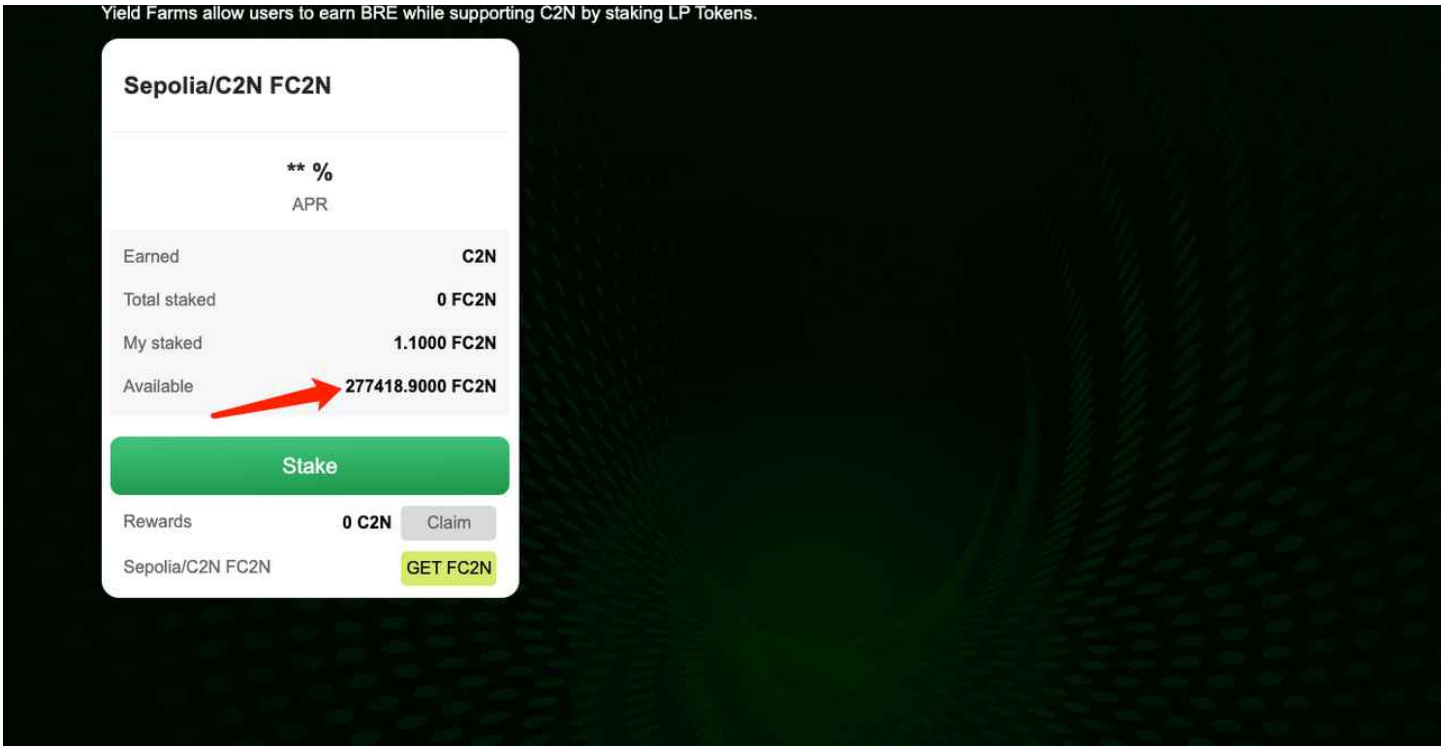
我们新增质押了1FC2N,交易完成之后我们会看到，My staked 从0.1 变成1.1;
Total staked 的更新是一个定时任务，我们需要等待一小段时间之后才能看到更新



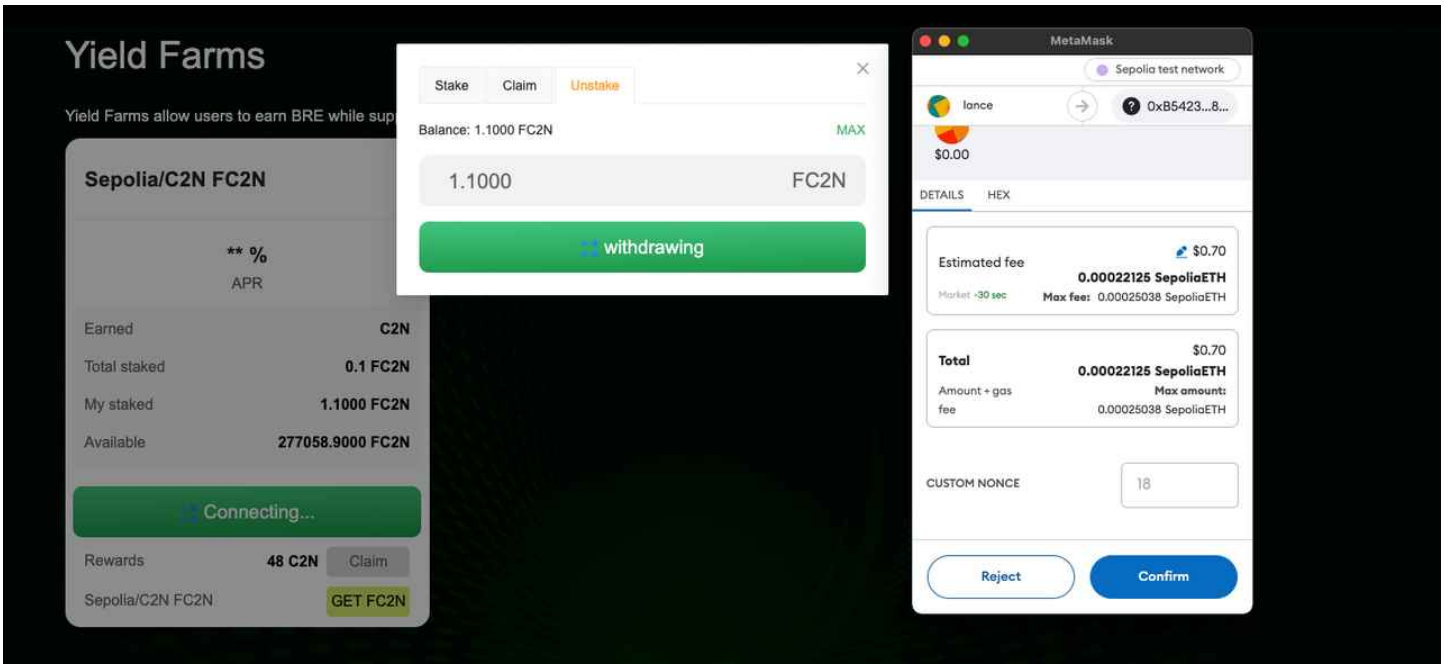
3.Claim 领取质押奖励的C2N,点击claim 并且在钱包确认



交易完成后我们会看到Available的FC2N数量增加了96，钱包里面C2N的代币数量同样增加了96



4.Unstake(withdraw),输入需要撤回的FC2N 数量(小于已经质押的Balance)，点击withdraw，并且在钱包确认交易



unstake 完成后我们可以看到my staked 的数量变为0

技术文档

部署流程

1. 复制.env.example 到.env,修改PRIVATE_KEY, 要求arbitrum sepolia上有测试eth
2. 部署c2n token
   `npx hardhat run scripts/deployment/deploy_c2n_token.js --network arb_sepoliaarb_sepolia`
3. 部署airdrop合约
   `npx hardhat run scripts/deployment/deploy_airdrop_c2n.js --network arb_sepolia`
4. 修改前端地址，运行前端测试airdrop功能

进入前端目录c2n-fe，安装依赖
`yarn`

修改token地址和airdrop 地址为合约之前部署的两个地址

c2n-fe/src/config/index.js 中的
`AIRDROP_TOKEN`
`AIRDROP_CONTRACT`

运行项目
`yarn dev`

6. farm

修改c2n-contracts/scripts/deployment/deploy_farm.js
第7行startTS为3分钟之后（必须是当前时间之后，考虑上链网络延迟）

修改 c2n-fe/src/config/farms.js
depositTokenAddress和earnedTokenAddress为AIRDROP_TOKEN的地址
修改stakingAddress为部署的farm合约地址

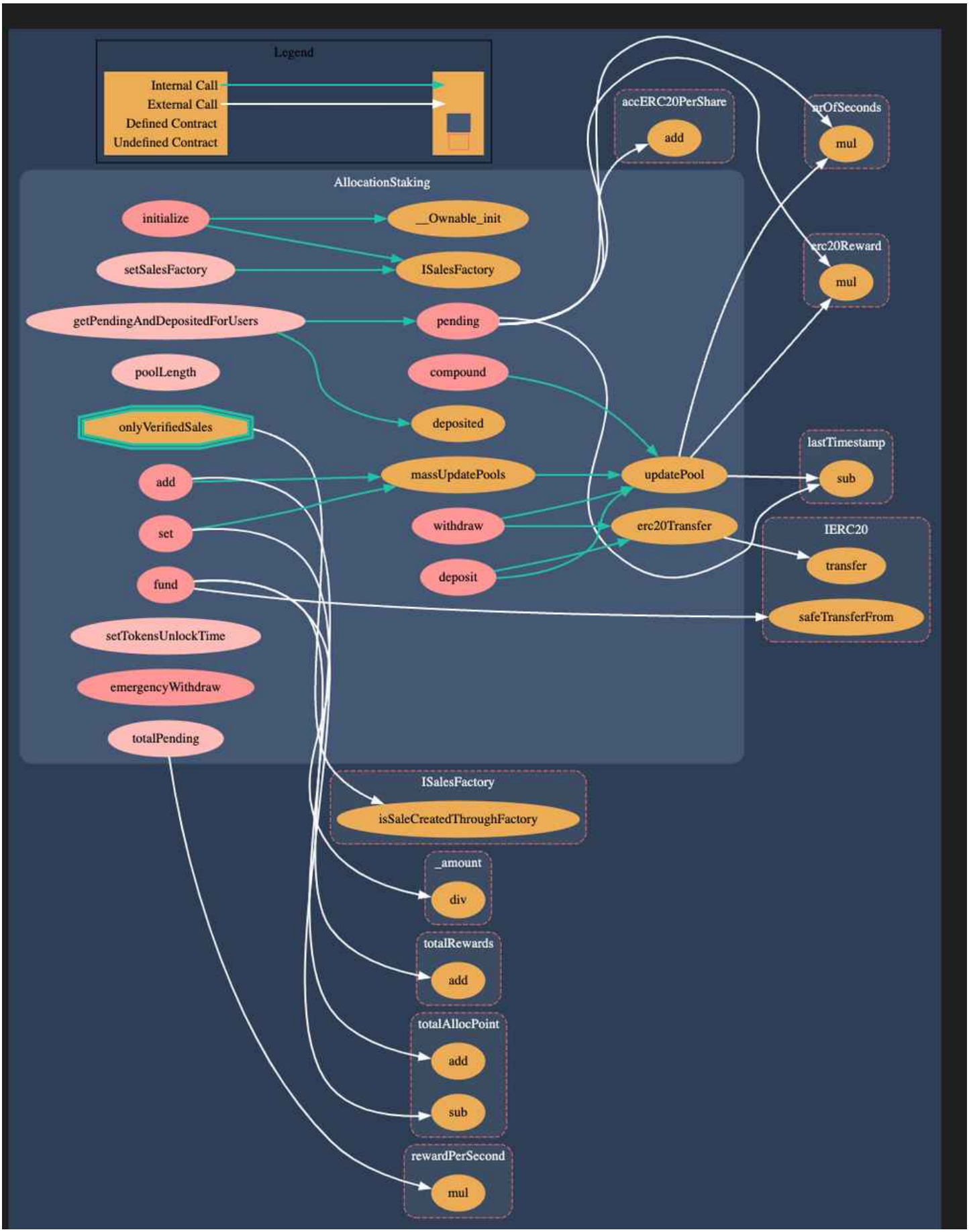
部署完毕，可以使用账号体验farm功能

合约开发说明

项目核心由两个合约组成，以下列出需要实现的函数功能

AllocationStaking.sol

关系调用

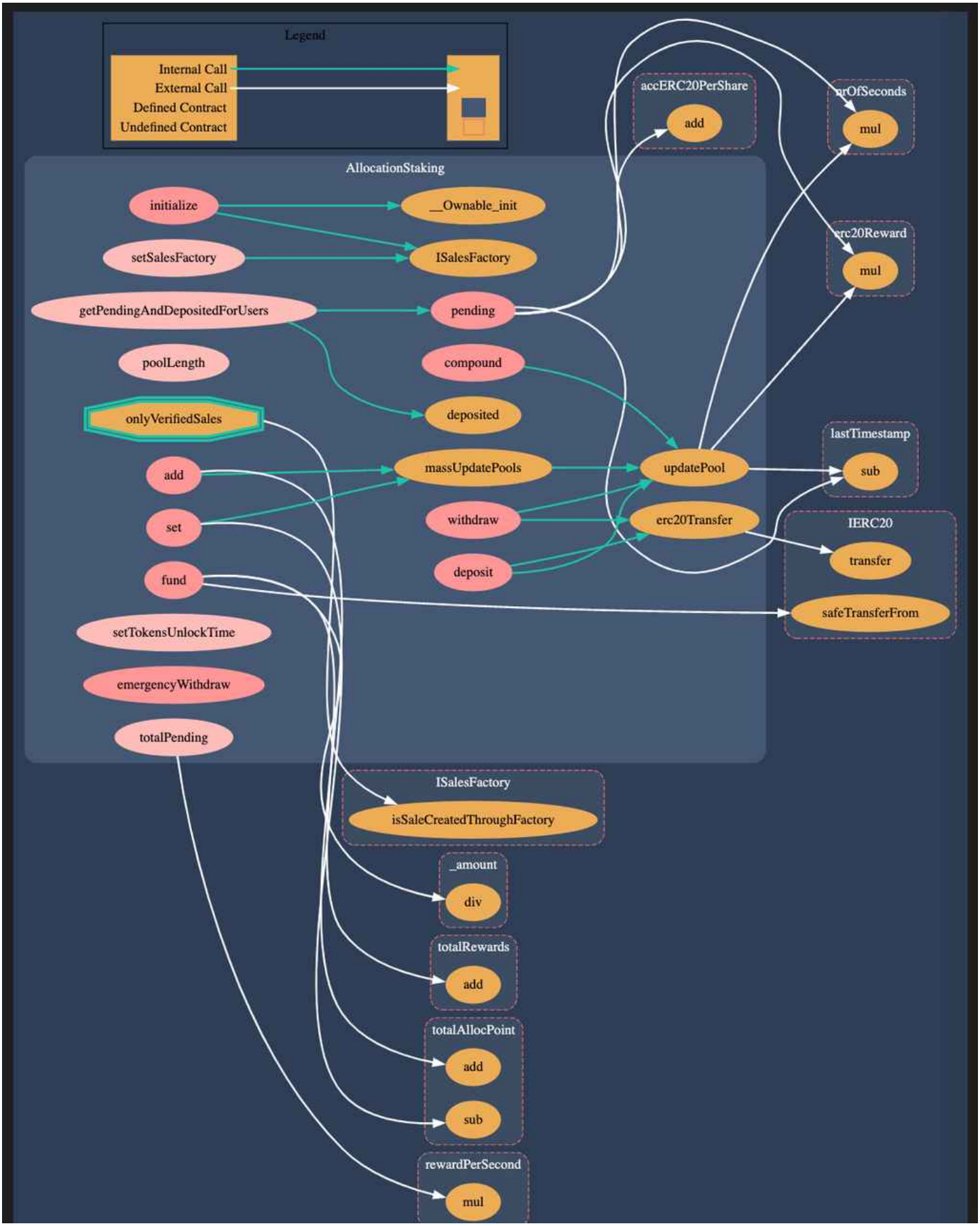


函数说明

暂时无法在飞书文档外展示此内容

BrewerySale.sol 功能

关系调用



函数调用

暂时无法在飞书文档外展示此内容

技术依赖

OpenZeppelin
OpenZeppelin库提供了一些安全的合约实现，如ERC20、SafeMath等。

前端开发
WIP

后端开发
数据库输入项目信息，配合合约sale显示项目进度和用户购买信息

学员任务

为了帮助学员逐步完成以太坊智能合约C2N Launchpad开发的学习任务，下面我将根据合约代码，拆分出一系列循序渐进的开发任务，并提供详细的文档。这将帮助学员理解并实践如何构建一个基于以太坊的农场合约（Farming contract），用于分配基于用户质押的流动性证明（LP tokens）的ERC20代币奖励。

概述

FarmingC2N合约是一个基于以太坊的智能合约，主要用于管理和分发基于用户质押的流动性证明(LP)代币的ERC20奖励。该合约允许用户存入LP代币，并根据质押的数量和时间来计算和分发ERC20类型的奖励。
开发任务拆分

任务一：了解基础合约和库的使用

1. 阅读和理解OpenZeppelin库的文档：熟悉IERC20、SafeERC20、SafeMath、Ownable这些库的功能和用途。
2. 创建基础智能合约结构：根据openZeppelin 库，导入上述合约。

任务二：用户和池子信息结构定义

1. 定义用户信息结构（UserInfo）：

- 学习如何在Solidity中定义结构体。
- 定义uint256类型的 amount,和uint256 rewardDebt字段
  在后续实现中会根据用户信息进行一些数学计算。

```
说明：在任何时间点，用户获得但还尚未分配的 ERC20 数量为：
pendingReward = (user.amount * pool.accERC20PerShare) - user.rewardDebt
每当用户向池中存入或提取 LP 代币时，会发生以下情况：
1. 更新池的 `accERC20PerShare`（和 `lastRewardBlock`）。
2. 用户收到发送到其地址的待分配奖励。
3. 用户的 `amount` 被更新。
4. 用户的 `rewardDebt` 被更新。
```

2. 定义池子信息结构（PoolInfo）：

- 理解并定义池子信息，包括LP代币地址、分配点、最后奖励时间戳等。

参考答案：

```
struct UserInfo {
    uint256 amount;
    uint256 rewardDebt;
}
struct PoolInfo {
    IERC20 lpToken;             // Address of LP token contract.
    uint256 allocPoint;         // How many allocation points assigned to this pool. ERC20s to distribute per block.
    uint256 lastRewardTimestamp;    // Last timstamp that ERC20s distribution occurs.
    uint256 accERC20PerShare;   // Accumulated ERC20s per share, times 1e36.
    uint256 totalDeposits; // Total amount of tokens deposited at the moment (staked)
}
```

任务三：合约构造函数和池子管理

首先我们先定义一些状态变量

- erc20：代表ERC20奖励代币的合约地址。
- rewardPerSecond：每秒产生的ERC20代币奖励数量。
- totalAllocPoint：所有矿池的分配点总和。
- poolInfo：所有矿池的数组。
- userInfo：记录每个用户在每个矿池中的信息。
- startTimestamp和endTimestamp：奖励开始和结束的时间戳。
- paidOut：已经支付的奖励总额。
- totalRewards：总的奖励额。

1. 编写合约的构造函数：

- 初始化ERC20代币地址、奖励生成速率和起始时间戳。

2. 实现添加新的LP池子的功能（add函数）：

- 按照poolInfo的结构，添加一个pool，并指定是否需要批量update合约资金信息

- 注意判断lastRewardTimestamp逻辑，如果大于startTimestamp，则为当前块高时间，否则还未开始发放奖励，设置为startTimestamp
- 学习权限管理，确保只有合约拥有者可以添加池子。

参考答案：

```
constructor(IERC20 _erc20, uint256 _rewardPerSecond, uint256 _startTimestamp) public {
    erc20 = _erc20;
    rewardPerSecond = _rewardPerSecond;
    startTimestamp = _startTimestamp;
    endTimestamp = _startTimestamp;
}

function add(uint256 _allocPoint, IERC20 _lpToken, bool _withUpdate) public onlyOwner {
    if (_withUpdate) {
        massUpdatePools();
    }
    uint256 lastRewardTimestamp = block.timestamp > startTimestamp ? block.timestamp : startTimestamp;
    totalAllocPoint = totalAllocPoint.add(_allocPoint);
    poolInfo.push(PoolInfo({
    lpToken : _lpToken,
    allocPoint : _allocPoint,
    lastRewardTimestamp : lastRewardTimestamp,
    accERC20PerShare : 0,
    totalDeposits : 0
    }));
}
```

任务四：fund功能实现

合约的所有者或授权用户可以通过此函数向合约注入ERC20代币，以延长奖励分发时间。
需求：

1. 确保合约在当前时间点仍可接收资金，即未超过奖励结束时间
2. 从调用者账户向合约账户安全转移指定数量的ERC20代币
3. 根据注入的资金量和每秒奖励数量，计算并延长奖励发放的结束时间
4. 更新合约记录的总奖励量
   参考答案

```
function fund(uint256 _amount) public {
    require(block.timestamp < endTimestamp, "fund: too late, the farm is closed");
    erc20.safeTransferFrom(address(msg.sender), address(this), _amount);
    endTimestamp += _amount.div(rewardPerSecond);
    totalRewards = totalRewards.add(_amount);
}
```

任务五：核心功能开发，奖励机制的实现

编写更新单个池子奖励的函数（updatePool）：

- 理解如何计算每个池子的累计ERC20代币每股份额。
- 需求说明: 该函数主要功能是确保矿池的奖励数据是最新的，并根据最新数据更新矿池的状态，需要实现以下功能：
  a. 更新矿池的奖励变量
  updatePool需要针对指定的矿池ID更新矿池中的关键奖励变量，确保其反映了最新的奖励情况。这包括：
  - 更新最后奖励时间戳： 如果池子还未结束，将矿池的lastRewardTimestamp更新为当前时间戳，以确保奖励的计算与时间同步，否则lastRewardTimestamp = endTimestamp
  - 计算新增的奖励：根据从上次奖励时间到现在的时间差，结合矿池的分配点数和全局的每秒奖励率，计算此期间应该新增的ERC20奖励量。

  b. 累加每股累积奖励
  根据新计算出的奖励量，更新矿池的accERC20PerShare（每股累积ERC20奖励）：
  - 奖励分配：将新增的奖励量按照矿池中当前LP代币的总量（totalDeposits）进行分配，计算出每份LP代币所能获得的奖励，并更新accERC20PerShare。

  c. 确保时间和奖励的正确性
  处理边界条件，确保在计算奖励时，各种时间点和奖励量的处理是合理和正确的：
  - 时间边界处理：如果当前时间已经超过了奖励分配的结束时间（endTimestamp），则需要相应调整逻辑以防止奖励超发。
  - LP代币总量检查：如果矿池中没有LP代币（totalDeposits为0），则不进行奖励计算，直接更新时间戳。
  参考实现：

```
function updatePool(uint256 _pid) public {
    PoolInfo storage pool = poolInfo[_pid];
    uint256 lastTimestamp = block.timestamp < endTimestamp ? block.timestamp : endTimestamp;

```

```
    if (lastTimestamp <= pool.lastRewardTimestamp) {
        return;
    }
    uint256 lpSupply = pool.totalDeposits;

    if (lpSupply == 0) {
        pool.lastRewardTimestamp = lastTimestamp;
        return;
    }

    uint256 nrOfSeconds = lastTimestamp.sub(pool.lastRewardTimestamp);
    uint256 erc20Reward =
nrOfSeconds.mul(rewardPerSecond).mul(pool.allocPoint).div(totalAllocPoint);

    pool.accERC20PerShare =
pool.accERC20PerShare.add(erc20Reward.mul(1e36).div(lpSupply));
    pool.lastRewardTimestamp = block.timestamp;
}
```

1. 实现用户存入和提取LP代币的功能（deposit和withdraw函数）：

- 理解如何更新用户的amount和rewardDebt。
  - Deposit: 函数允许用户将LP代币存入指定的矿池，以参与ERC20代币的分配。
    - 更新矿池奖励数据：调用updatePool函数，保证矿池数据是最新的，确保奖励计算的正确性。
    - 计算并发放挂起的奖励：如果用户已有存款，则计算用户从上次存款后到现在的挂起奖励，并通过erc20Transfer发放这些奖励。
    - 接收用户存款：通过safeTransferFrom函数，从用户账户安全地转移LP代币到合约地址。
    - 更新用户存款数据：更新用户在该矿池的存款总额和奖励债务，为下次奖励计算做准备。
    - 记录事件：发出Deposit事件，记录此次存款操作的详细信息。
  - Withdraw
    - 更新矿池奖励数据：调用updatePool函数更新矿池的奖励变量，确保奖励的准确性。
    - 计算并发放挂起的奖励：计算用户应得的挂起奖励，并通过erc20Transfer将奖励发放给用户。
    - 提取LP代币：安全地将用户请求的LP代币数量从合约转移到用户账户。
    - 更新用户存款数据：更新用户的存款总额和奖励债务，准确记录用户的新状态。
    - 记录事件：发出Withdraw事件，记录此次提款操作的详细信息。
      参考答案：

```
// Deposit LP tokens to Farm for ERC20 allocation.
function deposit(uint256 _pid, uint256 _amount) public {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];

    updatePool(_pid);

    if (user.amount > 0) {
        uint256 pendingAmount =
user.amount.mul(pool.accERC20PerShare).div(1e36).sub(user.rewardDebt);
        erc20Transfer(msg.sender, pendingAmount);
    }

    pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);
    pool.totalDeposits = pool.totalDeposits.add(_amount);

    user.amount = user.amount.add(_amount);
    user.rewardDebt = user.amount.mul(pool.accERC20PerShare).div(1e36);
    emit Deposit(msg.sender, _pid, _amount);
}

// Withdraw LP tokens from Farm.
function withdraw(uint256 _pid, uint256 _amount) public {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];
    require(user.amount >= _amount, "withdraw: can't withdraw more than
deposit");
    updatePool(_pid);

    uint256 pendingAmount =
user.amount.mul(pool.accERC20PerShare).div(1e36).sub(user.rewardDebt);

    erc20Transfer(msg.sender, pendingAmount);
    user.amount = user.amount.sub(_amount);
    user.rewardDebt = user.amount.mul(pool.accERC20PerShare).div(1e36);
    pool.lpToken.safeTransfer(address(msg.sender), _amount);
    pool.totalDeposits = pool.totalDeposits.sub(_amount);

    emit Withdraw(msg.sender, _pid, _amount);
}
```

任务六：紧急提款和奖励分配

1. 实现紧急提款功能（emergencyWithdraw函数）：

- 让用户在紧急情况下提取他们的LP代币，但不获取奖励。

2. 实现ERC20代币转移的内部函数（erc20Transfer）：

- 确保奖励正确支付给用户。
  参考答案：

```
// Withdraw without caring about rewards. EMERGENCY ONLY.
function emergencyWithdraw(uint256 _pid) public {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];
    pool.lpToken.safeTransfer(address(msg.sender), user.amount);
    pool.totalDeposits = pool.totalDeposits.sub(user.amount);
    emit EmergencyWithdraw(msg.sender, _pid, user.amount);
    user.amount = 0;
    user.rewardDebt = 0;
}

// Transfer ERC20 and update the required ERC20 to payout all rewards
function erc20Transfer(address _to, uint256 _amount) internal {
    erc20.transfer(_to, _amount);
    paidOut += _amount;
}
```

任务七：合约测试和部署

1. 编写测试用例：

- 使用如Truffle或Hardhat的框架进行合约测试。

2. 部署合约到测试网络（Sepolia）：

- 学习如何在公共测试网络上部署和管理智能合约。

任务七：前端集成和交互

1. 开发一个简单的前端应用：

- 使用Web3.js或Ethers.js与智能合约交互。

2. 实现用户界面：

- 允许用户通过网页界面存入、提取LP代币，查看待领取奖励。

任务重难点分析

在上述的智能合约代码中，奖励机制的核心功能围绕着分配ERC20代币给在不同流动性提供池（LP pools）中质押LP代币的用户。这个过程涉及多个关键步骤和计算，用以确保每个用户根据其质押的LP代币数量公平地获得ERC20代币奖励。下面将详细解释这个奖励机制的实现过程。

奖励计算原理

1. 用户信息（UserInfo）和池子信息（PoolInfo）：

- UserInfo 结构存储了用户在特定池子中质押的LP代币数量（amount）和奖励债务（rewardDebt）。奖励债务表示在最后一次处理后，用户已经计算过但尚未领取的奖励数量。
- PoolInfo 结构包含了该池子的信息，如LP代币地址、分配点（用于计算该池子在总奖励中的比例）、最后一次奖励时间戳、累计每股分配的ERC20代币数（accERC20PerShare）等。

2. 累计每股分配的ERC20代币（accERC20PerShare）的计算：

- 当一个池子接收到新的存款、提款或奖励分配请求时，系统首先调用updatePool函数来更新该池子的奖励变量。
- 计算从上一次奖励到现在的时间内，该池子应分配的ERC20代币总量。这个总量是基于时间差、池子的分配点和每秒产生的奖励量来计算的。
- 将计算出的奖励按照池子中总LP代币数量平分，更新accERC20PerShare，确保每股的奖励反映了新加入的奖励。

3. 用户奖励的计算：

- 当用户调用deposit或withdraw函数时，合约首先计算用户在这次操作前的待领取奖励。
- 待领取奖励是通过将用户质押的LP代币数量乘以池子的accERC20PerShare，然后减去用户的rewardDebt来计算的。这样可以得到自上次用户更新以来所产生的新奖励。
- 用户完成操作后，其amount（如果是存款则增加，如果是提款则减少）和rewardDebt都将更新。新的rewardDebt是用户更新后的LP代币数量乘以最新的accERC20PerShare。

奖励发放

- 在用户进行提款（withdraw）操作时，计算的待领取奖励会通过erc20Transfer函数直接发送到用户的地址。
- 这种奖励分配机制确保了用户每次质押状态变更时，都会根据其质押的时间和数量公平地获得相应的ERC20代币奖励。
  通过这种设计，智能合约能够高效且公平地管理多个LP池子中的奖励分配，使得用户对质押LP代币和领取奖励的过程感到透明和公正。

InComing

AllocationStaking和c2nSale 正在开发中。。。