北京邮电大学

计算机学院（示范性软件学院）

实验报告

课程名称： 区块链技术原理

项目名称： 以太坊溯源系统实验

项目完成人：

姓名： 孙津硕 学号： 2023140748

姓名： 赵振宇 学号： 2023110804

姓名： 学号：

姓名： 学号：

指导教师： 杨 谈

日 期： 2023 年 12 月 23 日

# 实验目的

本次实验的主要目的是通过编写部署智能合约及创建基于以太坊的溯源系统，让参与者深入理解以太坊的智能合约编写与部署流程，以及如何基于以太坊私链构建一个实际应用场景——溯源 Dapp。通过本实验，学会并掌握智能合约的编写、编译和测试方法，并能够运用以太坊平台创建并实现一个完整的区块链防伪溯源系统。

# 实验内容

随着区块链技术的快速发展，其在防伪溯源场景的应用价值日益凸显，受到众多科技巨头的关注与布局。其中，数据溯源的核心在于如何有效地构建数据模型，以确保源头信息的有效获取、存储及后期利用。为此，实验过程中需针对不同业务领域和应用场景抽象出合适的数据模型，并规范各类数据接入标准；同时，将复杂的业务流程细化成各个阶段，针对性地整合不同阶段产生的业务数据；最后，运用数据特征标识技术，实现在整个生命周期内追踪和查询数据的历史版本。

智能合约作为区块链技术的一大革新，其工作原理类似于传统应用程序中的条件触发机制——即if-then语句，当预设条件满足时，智能合约会自动执行相关条款，且由于其运行于去中心化的区块链网络之上，无需任何中介参与，确保了自动化和可信的执行过程。

以太坊作为一个建立在区块链和加密货币理念上的开源、图灵完备的区块链平台，为智能合约提供了安全可靠的运行环境。丰富的API接口使得开发者能轻松地在以太坊平台上编写、编译并部署各种由区块链技术支持的智能合约应用。

本次实验旨在通过实践探索区块链技术在防伪溯源领域的广泛应用潜力，并结合以太坊智能合约开发一个基于食品追溯的Dapp。实验内容具体涵盖了以下四个核心步骤：

1. 首先，参照实验一的指导搭建并配置以太坊私链环境，确保其成功启动运行，为后续实验提供稳定的区块链基础设施。

2. 在此基础上，设计并编写名为“FoodTransportation”的智能合约，该合约专门用于记录食品从生产至销售各环节的关键信息（如地点、负责人和时间戳等），实现对食品流转过程的透明化追溯。

3. 利用 Remix 这款在线 Solidity 编辑器完成智能合约的编译与测试工作，确保合约逻辑无误并在以太坊虚拟机上能够正确执行。

4. 接着，采用 truffle 框架和 webpack 打包工具构建并部署一个基于以太坊的溯源 Dapp。该 Dapp 集成了 MetaMask 插件，用户可以在 Chrome 浏览器中进行直观、便捷的交互操作。

综上所述，本次实验集合了四个关键组成部分：搭建以太坊区块链服务的虚拟机服务器、实现防伪溯源功能的FoodSafe智能合约源代码、集成在Chrome浏览器中的MetaMask插件以及用户友好的前端交互界面。整个溯源Dapp项目依托truffle框架进行整体管理，并借助webpack工具进行了高效的前端模块打包处理，从而实现了区块链技术在实际防伪溯源场景中的创新应用。

# 实验环境

本实验需要以下软硬件环境支持：

Linux 或 Windows 系统

Remix（在线 Solidity 编辑器）

Truffle 框架

Webpack 打包工具

以太坊 Geth 客户端

MetaMask 插件

# 附录

## 问题分析

### 目标

本实验的核心目标是利用区块链技术，尤其是以太坊智能合约的特性，构建一个去中心化、公开透明且不可篡改的食品溯源系统。通过“FoodTransportation”智能合约记录食品从生产、加工、运输到销售等各个环节的关键信息，确保食品供应链的全程可追溯性，从而提升食品安全监管效率，增强消费者对食品来源的信任度。

### 功能

### 

图1 以太坊食品溯源系统功能展示

#### 功能一：获取当前食品溯源信息

getCurrentLocation: function()

    {

        var contractAddress = document.getElementById("contractAddress").value;

        var info = "";

        try{

        var deployedFoodTransportation = FoodTransportationContract.at(contractAddress);

        var food;

        deployedFoodTransportation.getFoodName.call(function (error,foodName) {

            food = foodName;

        })

        deployedFoodTransportation.getLocationNum.call(function (error, locationNum){

            var tbFoodName = document.getElementById("table\_foodName");

            tbFoodName.innerText = "食品名称: "+food;

            var tb = document.getElementById("tb");

            resetTable(tb);

            while (locationNum >= 1){

                deployedFoodTransportation.getLocation.call(locationNum-1, function(error, returnValues){

                    addRow(tb,returnValues[0],returnValues[1],timestampToTime(returnValues[2]))

                })

                locationNum--;

            }

        })

        }catch(e){

            alert("食品信息地址不存在，请重新输入")

        }

    }

为了方便用户追踪食品流向，设计了getCurrentLocation查询函数。其中，locationNum

返回食品经过的所有流转位置总数；table\_foodName 获取食品的基础信息如名称；contractAddress则用于检索指定位置的详细流转记录。

通过点击获取当前食品溯源信息按钮，用户可以看到当前食品的信息地址、食品名称、位置名称和运输员名称信息。

#### 功能二：更新食品位置

    addNewLocation: function()

    {

        var contractAddress = document.getElementById("contractAddress").value;

        console.log("地址"+document.getElementById("contractAddress").value)

        try{

        var deployedFoodTransportation = FoodTransportationContract.at(contractAddress);

        var foodName = document.getElementById("foodName").value;

        var locationName = document.getElementById("locationName").value;

        var personName = document.getElementById("personName").value;

        deployedFoodTransportation.addNewLocation(foodName, locationName, personName, function(error){

            console.log(error);

        });

        document.getElementById("locationName").value = "";

        document.getElementById("personName").value = "";

        }catch(e){

            alert("食品信息地址不存在，请重新输入")

        }

    },

智能合约提供了一个名为 addNewLocation 的接口函数，允许参与者上传新的流转环节信息，包括地理位置、负责人、时间戳等关键数据，并将这些信息永久存储在区块链上。

#### 功能三：添加新食品

createContract: function()

    {

        var FoodTransportation = FoodTransportationContract.new(

            {

                from: web3.eth.accounts[0],

                data: '6080604...',

                gas: '4700000'

            }, function (e, myContract){

                if (myContract.address) {

                    console.log('Contract mined! address: ' + myContract.address + ' transactionHash: ' + myContract.transactionHash);

                    document.getElementById("contractAddress").value = myContract.address;

                    document.getElementById("addNewLocation").disabled=false;

                    document.getElementById("getCurrentLocation").disabled=false;

                    document.getElementById("foodName").value = "";

                    var tbFoodName = document.getElementById("table\_foodName");

                    tbFoodName.innerText = "";

                    var tb = document.getElementById("tb");

                    resetTable(tb);

                    // 合约发布成功后，才能调用后续的方法

                } else {

                    console.log("contract deploy transaction hash: " + myContract.transactionHash) //部署合约的交易哈希值

                }

            });

    },

为了实现对不同食品进行独立记录和追溯，智能合约中还设计了初始化食品信息的功能。通过调用特定的createContract函数，在系统中创建新的食品条目，并为其分配唯一的标识符。一旦食品的基础信息如名称、生产日期等被成功添加到区块链上，即可开始对该食品从生产至销售全过程的关键环节进行跟踪记录。

### 性能

#### 响应速度

考虑到以太坊网络的运行机制，智能合约的实际执行速度会受到网络状态（如区块生成速度、交易确认时间）的影响。在本地私链环境下进行开发和测试时，由于无需考虑网络延迟及共识过程，智能合约的操作具有较高的实时响应能力。

#### 存储效率与查询性能

在以太坊智能合约中，数据存储成本是需要考虑的重要因素。本溯源系统通过合理组织数据结构（如使用 mapping 结构存储流转记录），确保了高效利用存储空间的同时，也优化了查询速度。当用户查询某一食品的流转历史时，智能合约能够快速定位并返回对应的数据，从而保证了用户体验和系统的整体响应性能。

### 健壮性

#### 异常输入处理

为确保系统的稳定性和安全性，智能合约在设计中充分考虑了异常输入处理。例如，在调用 addNewLocation 函数时，对输入参数进行了严格的类型检查和边界控制，要求 locationNum 必须为有效整数，并且不能超过当前实际的位置记录数量。此外，针对可能出现的其他错误情况，智能合约还包含了相应的错误处理逻辑，以防止因非法操作导致的数据混乱或系统崩溃。

#### 异常状态处理与安全性

除了严格的输入验证，智能合约还针对各种可能的异常状态进行了预防和处理，例如网络中断导致的交易失败或者区块回滚等情况。通过监听相关的事件以及区块链状态变化，智能合约能够自动恢复至正确状态，确保数据的一致性和完整性。此外，为了保障系统的安全性，所有关键操作均需经过权限校验，只有具有相应权限的参与者才能执行更新食品位置等敏感操作，有效防止了恶意篡改或非法入侵。

#### 可扩展性

智能合约的设计充分考虑了未来的业务扩展需求。当食品供应链涉及更多复杂的流程时，可通过增加新的数据结构或接口函数来适应这些变化，而无需对现有架构进行大规模重构。同时，由于智能合约的公开透明特性，任何符合规范的新功能都能无缝融入当前的食品溯源系统中，增强了系统的可扩展性和可持续发展能力。

#### 数据不可篡改与透明性

在本实验中，首先通过编写 Solidity 语言的智能合约（如上述 FoodTransportation 合约），定义了食品流转信息的数据结构 Location，并设计了一系列公开函数接口，实现了食品溯源信息的存储、查询与更新功能。智能合约部署在以太坊私链上，确保数据的不可篡改性与透明性。

## 设计方案

本实验中的 Dapp 设计采用了前后端分离架构。前端负责用户交互界面的设计与实现，使用现代Web开发技术构建直观易用的UI界面，通过集成MetaMask插件完成与后端智能合约的交互。后端部分主要由FoodTransportation智能合约构成，作为数据存储和业务逻辑处理的核心模块，其所有操作均基于以太坊区块链进行。

所以整体设计上，本实验基于以太坊私链的溯源 Dapp 被划分为四个主要模块：环境搭建、智能合约开发与部署、前端交互界面构建以及数据交互处理。

### 环境搭建模块

#### 功能描述

此模块负责搭建以太坊私有链环境，包括安装和配置 geth 客户端，创建并初始化创世区块（genesis.json），启动私有链节点，并进行挖矿操作确保交易的有效确认。

#### 文件结构

主要包括配置文件（如 genesis.json）及相关 shell 脚本或命令行指令记录。

### 智能合约开发与部署模块

#### 功能描述

编写名为“FoodTransportation”的智能合约源代码，实现食品流转信息的存储、查询和更新等功能。编译并通过 truffle 框架将智能合约部署到私链上。

#### 文件结构

主要包括 Solidity 源码文件（如 FoodTransportation.sol），truffle 配置文件（如 truffle-config.js），迁移文件夹（migrations/）内包含部署脚本。

### 前端交互界面构建模块

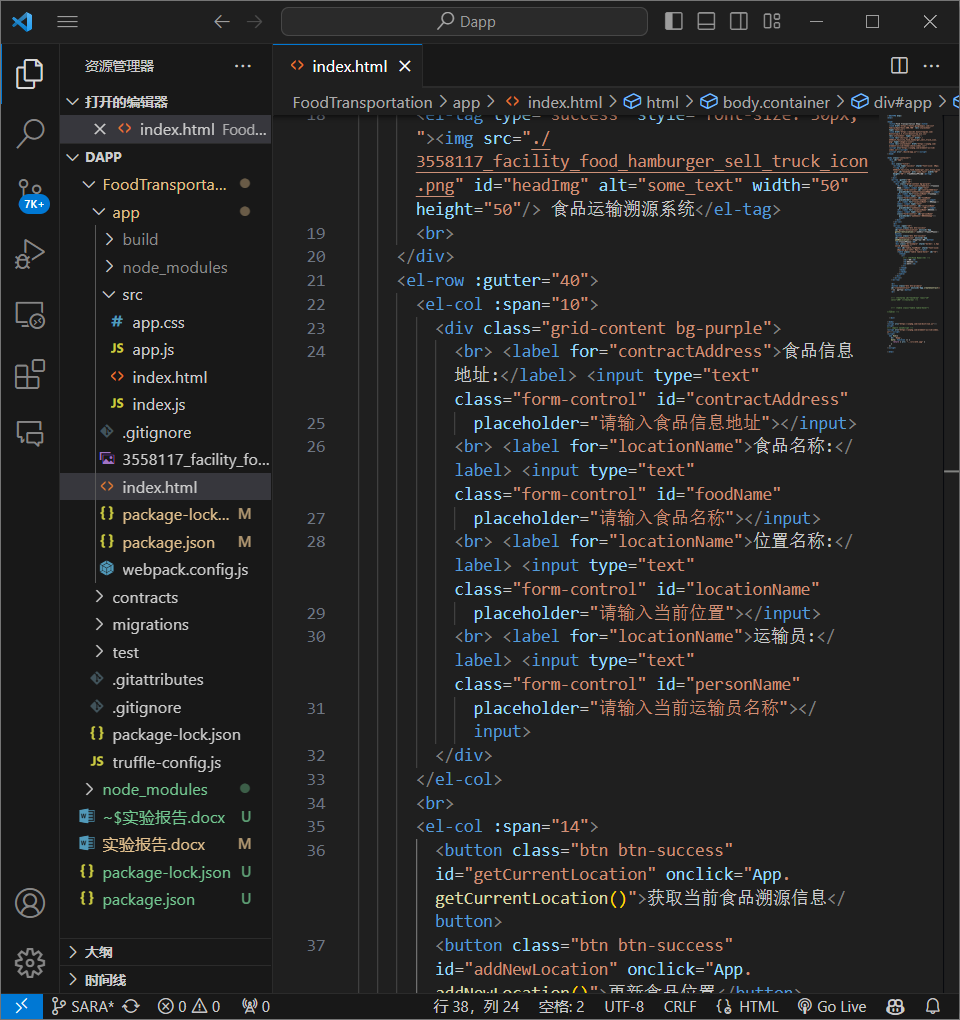


图2 前端交互界面构建

#### 功能描述

使用 webpack 打包工具，结合 HTML、CSS 和 JavaScript 构建用户友好的 Web 界面，允许用户查看食品溯源信息、添加新的流转记录等操作，并通过 MetaMask 插件连接至私链进行交互。

#### 文件结构

包含静态资源（HTML, CSS）、JavaScript 文件（业务逻辑处理和与智能合约交互部分）、打包配置文件（webpack.config.js）等。

### 数据交互处理模块

#### 功能描述

该模块是后端逻辑的核心部分，利用 web3.js 或 ethers.js 作为以太坊区块链的 JavaScript API，实现实现前端与智能合约之间的数据交换。例如，调用智能合约函数获取食品流转信息，发送交易更新位置信息等。

#### 函数说明

关键函数包括但不限于 addNewLocation（用于上传新流转环节信息到区块链），getLocationNum 和 getLocation（用于从智能合约中检索流转信息）。

## 程序整体流程图

开始

|---> 初始化阶段

|---> 搭建以太坊私链环境

|---> 编写并部署智能合约

|---> 运行阶段

|---> 用户通过前端界面触发操作

|---> 前端调用数据交互处理模块中的相应函数

|---> 数据交互处理模块通过 web3.js 向智能合约发起请求

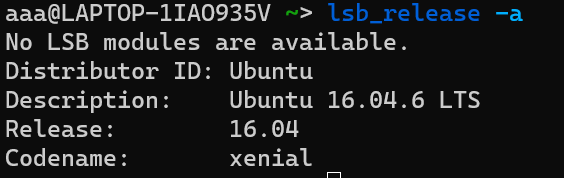
|---> 智能合约执行相关操作并返回结果给数据交互处理模块

|---> 数据交互处理模块将结果显示在前端界面上供用户查看

结束

## 实验结果

利用WSL2配置了Ubuntu16.04环境

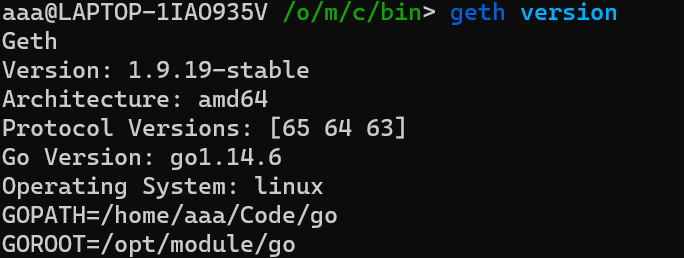


在Linux中下载安装geth

安装go1.14.6并配置环境变量。



安装geth 1.9.19-stable并配置环境变量。



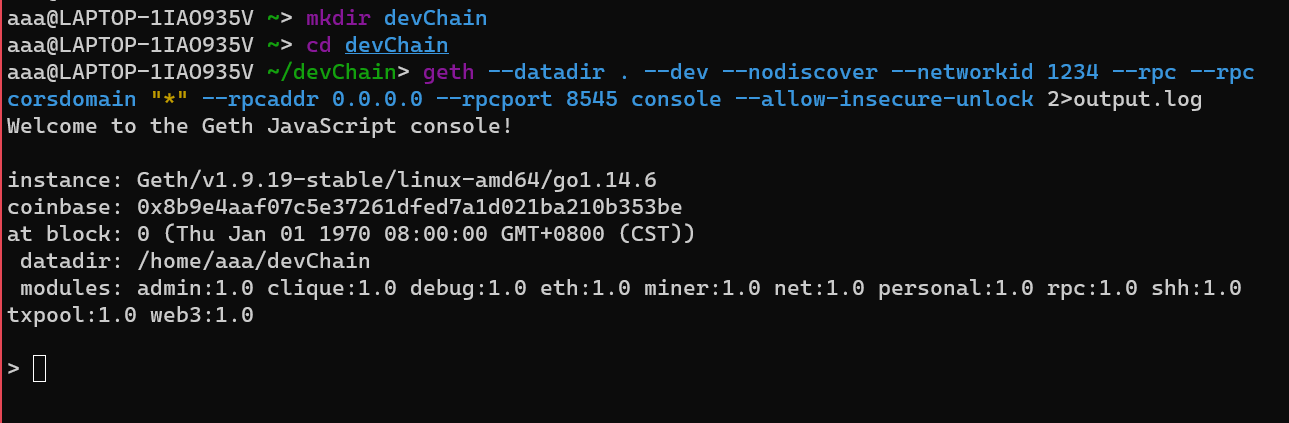
使用geth完成启动节点同步，启动开发者模式等操作。

执行geth --datadir . --syncmode fast 命令使用快速模式同步区块

图形用户界面, 文本, 应用程序, 聊天或短信

描述已自动生成

执行mkdir devChain ，cd devChain ，geth --datadir . --dev --nodiscover --networkid 1234 --rpc --rpccorsdomain "\*" --rpcaddr 0.0.0.0 --rpcport 8545 console --allow-insecure-unlock 2>output.log 命令，启动开发者模式。



新建mychain文件夹，创建genesis.json文件，写入各项配置



在mychain文件夹下，执行geth --datadir . init genesis.json命令，根据genesis.json中的各项配置初始化链。

文本

描述已自动生成

启动私链，networkid和rpcaddr的设置方法同开发者模式，networkid设置为1234，rpcaddr设置为0.0.0.0，输出日志设置为output.log。执行命令：geth --datadir . --nodiscover --networkid 1234 --rpc --rpccorsdomain "\*" --rpcaddr 0.0.0.0 --rpcport 8545 console --allow-insecure-unlock 2>output.log

文本

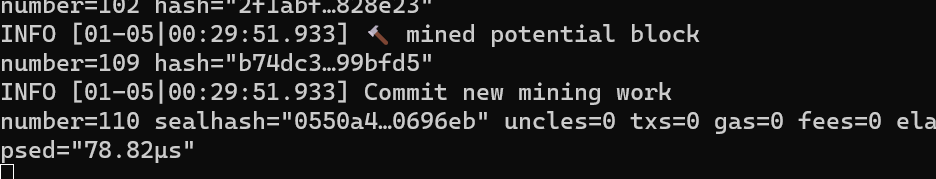
描述已自动生成

搭建私链后在私链上进行测试，实现账户创建、交易验证等基本功能。

执行eth.blockNumber命令查看块数量。

执行personal.newAccount("123")命令创建账户，账户密码为”123”。执行miner.start(1)命令开始挖矿。

通过查看output.log，当成功挖到新块后会有如下提示。



再查看blockNumber，发现已经挖到了许多块

图形用户界面, 文本, 应用程序

描述已自动生成

执行personal.newAccount("password")命令创建另一个账户用于交易。

执行me=eth.accounts[0]命令获得账户一。

执行itper=eth.accounts[1]命令per获得账户二。

执行personal.unlockAccount(my)命令解锁账户一（交易sender的账户）。

使用eth.getBalance(my)，eth.getBalance(other)命令查看交易账户的余额。

文本

描述已自动生成

执行eth.sendTransaction({from:me, to:it, value:10000})命令发起转账交易，提示交易哈希。

执行eth.getTransaction命令并输入生成的交易哈希查看指定交易的详细信息。



挖到新块后，执行eth.getBlock(0)命令通过区块号查看区块信息

文本

描述已自动生成

本次实验中，我们首先成功地搭建并配置了一套以太坊私链环境。利用 Remix 开发工具编写了名为 FoodTransportation 的智能合约，针对食品流转追溯场景进行了精细设计和实现，并通过编译与单元测试确保了其逻辑的正确性和无误性。

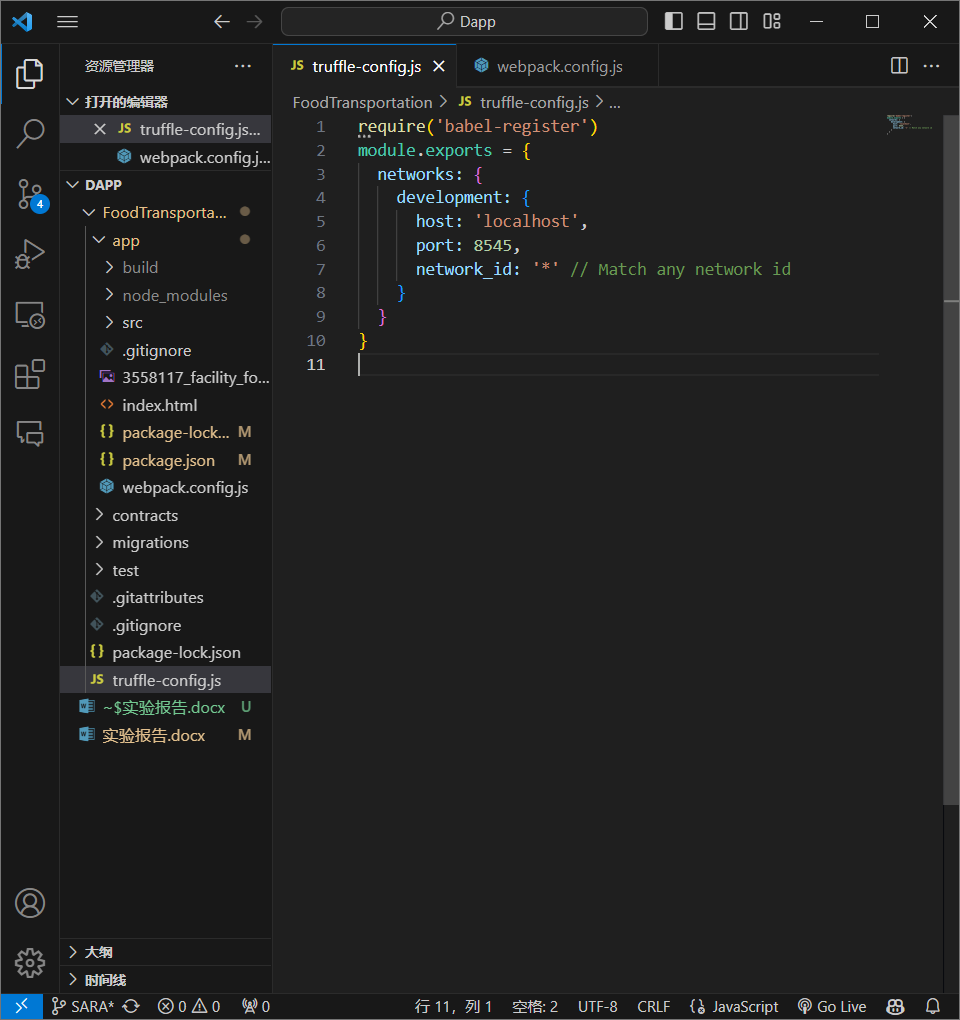


图3 Truffle 框架

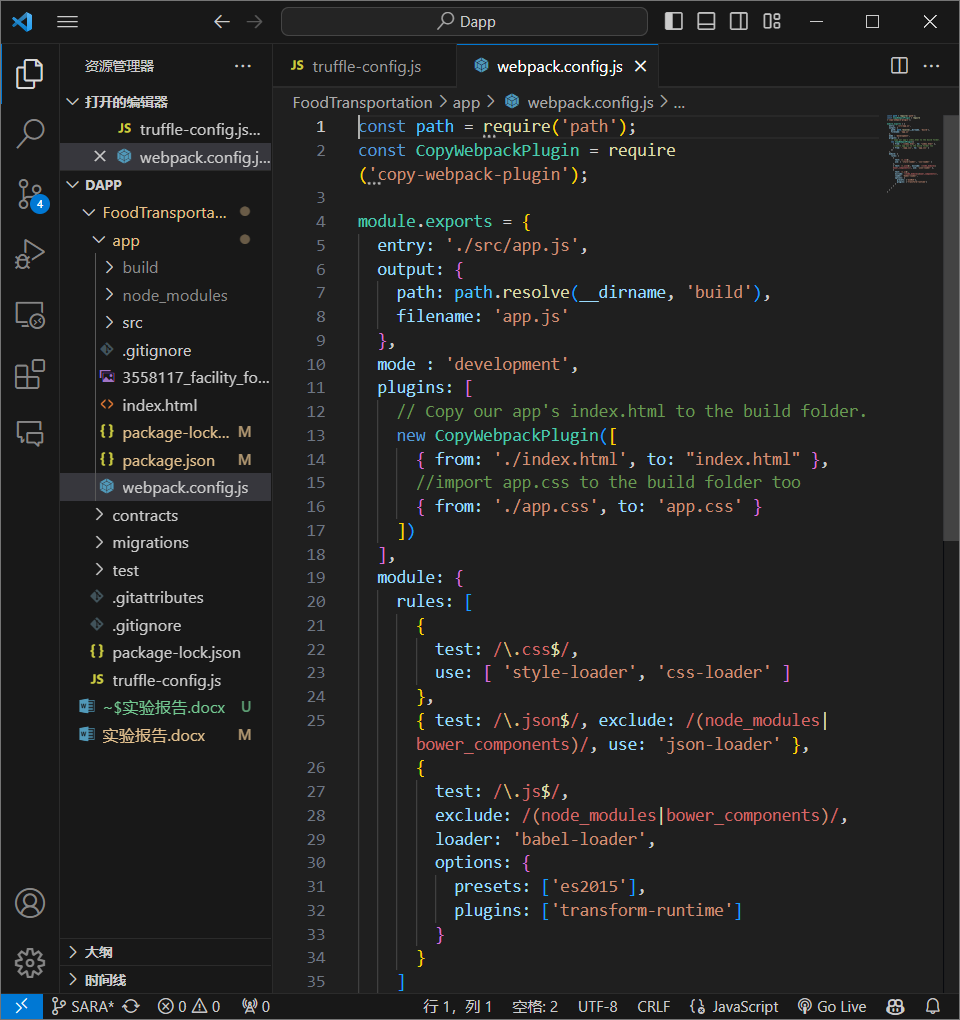


图4 Webpack打包

接下来，我们采用了 Truffle 框架进行智能合约的管理和部署，并结合 Webpack 进行了前端 Dapp 的打包构建，旨在实现一个高效、便捷的溯源 Dapp。此过程中，Dapp 成功集成了 MetaMask 浏览器插件，使得用户能够在浏览器环境下无缝地与智能合约进行交互操作。

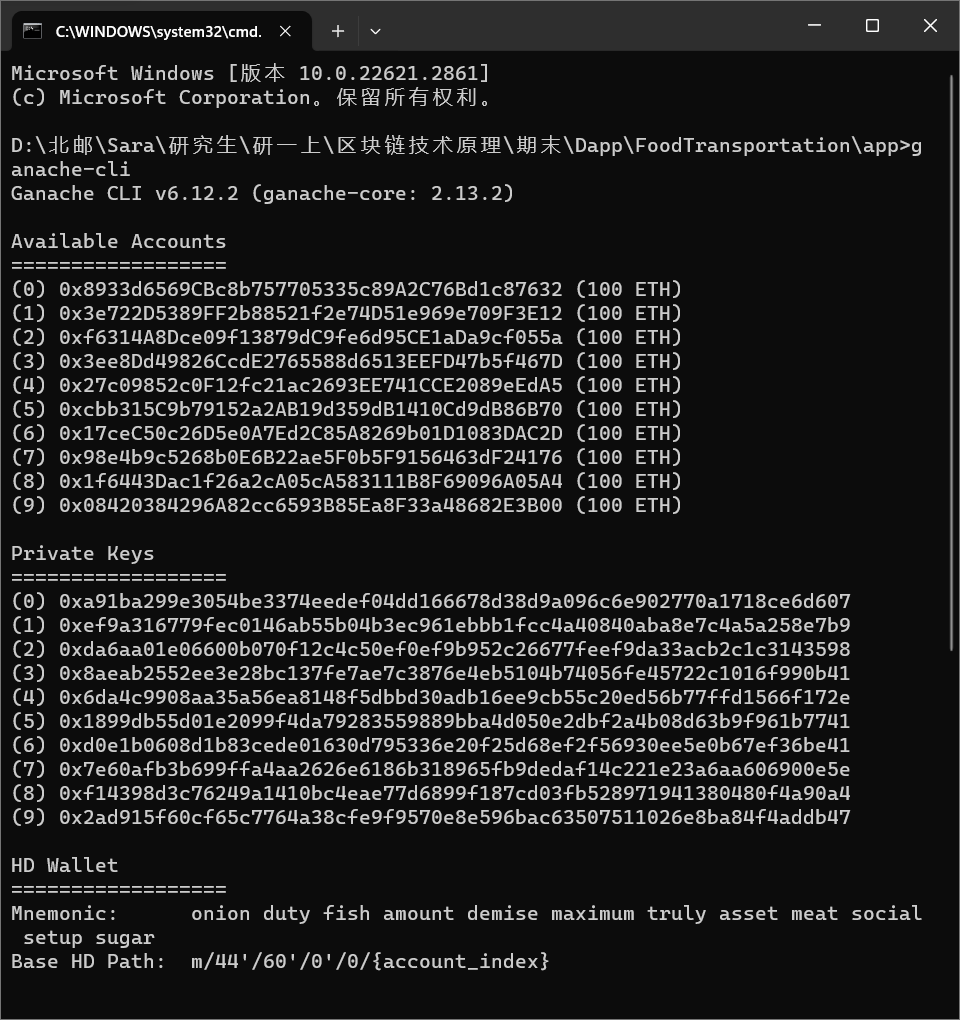


图5 合约部署 自动创建10个账户，可填入metamask中

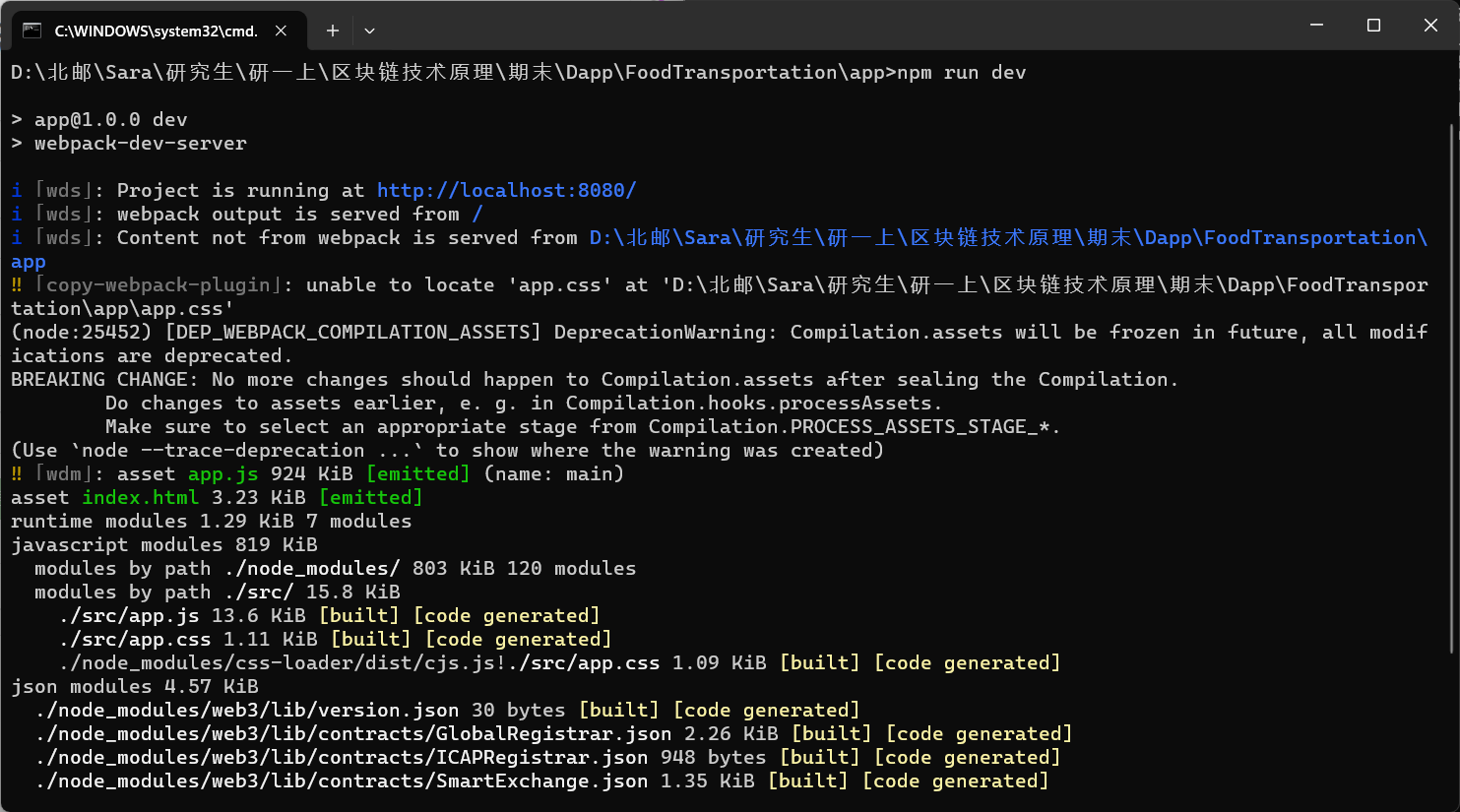


图6 运行打包项目 Dapp/app执行npm run dev

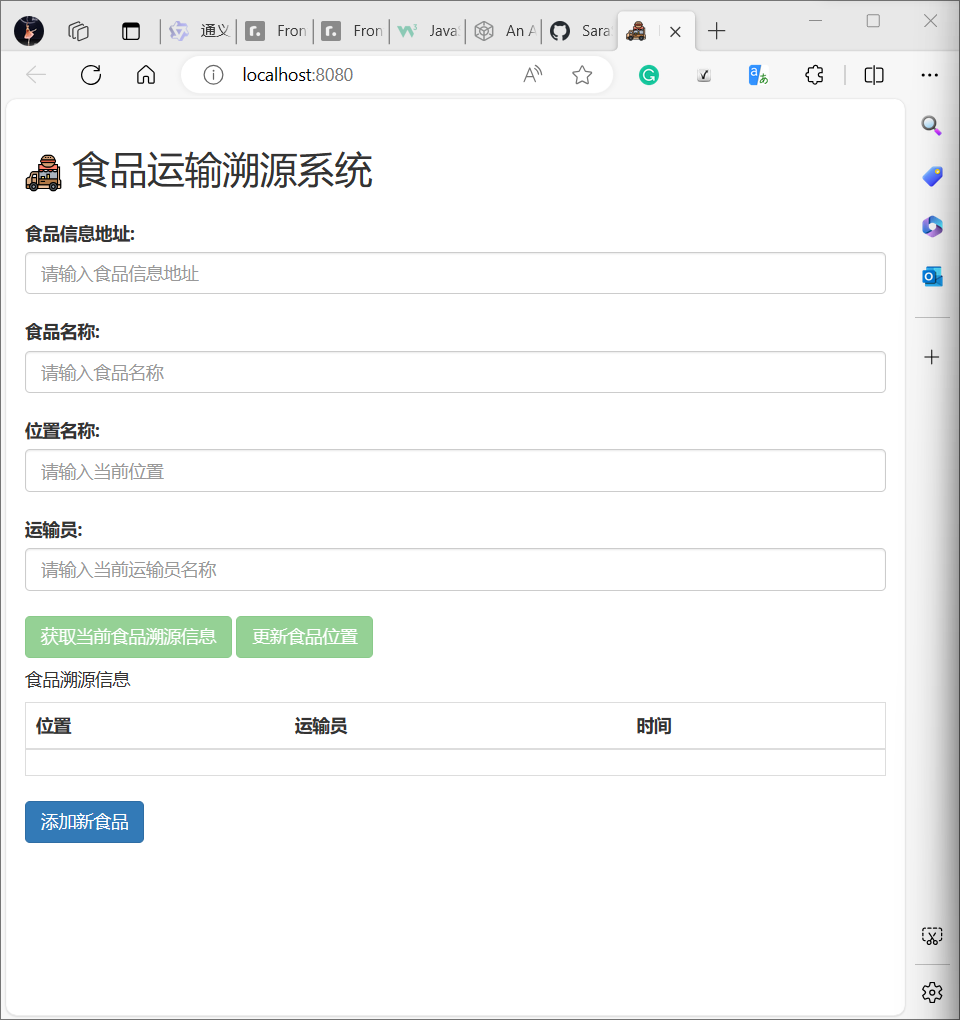


图7 食品运输溯源系统界面 成功运行食品运输溯源系统

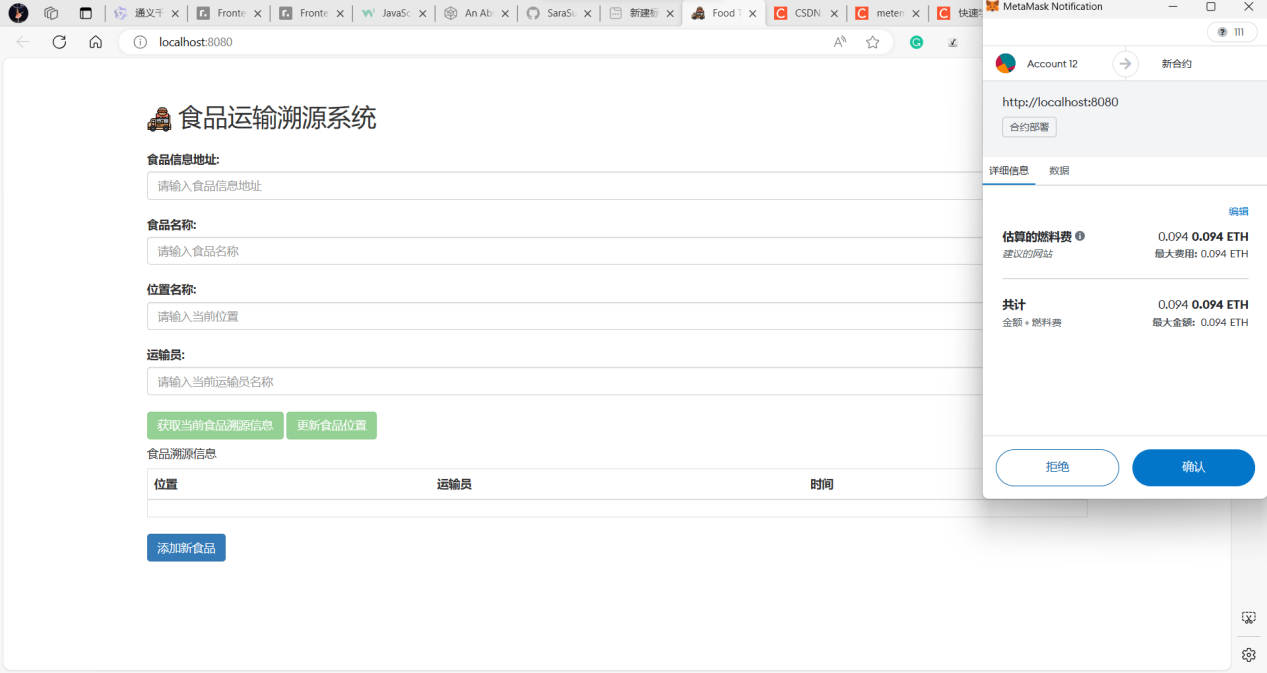


图8 添加新食品 签署合约后，可填入食品相关信息

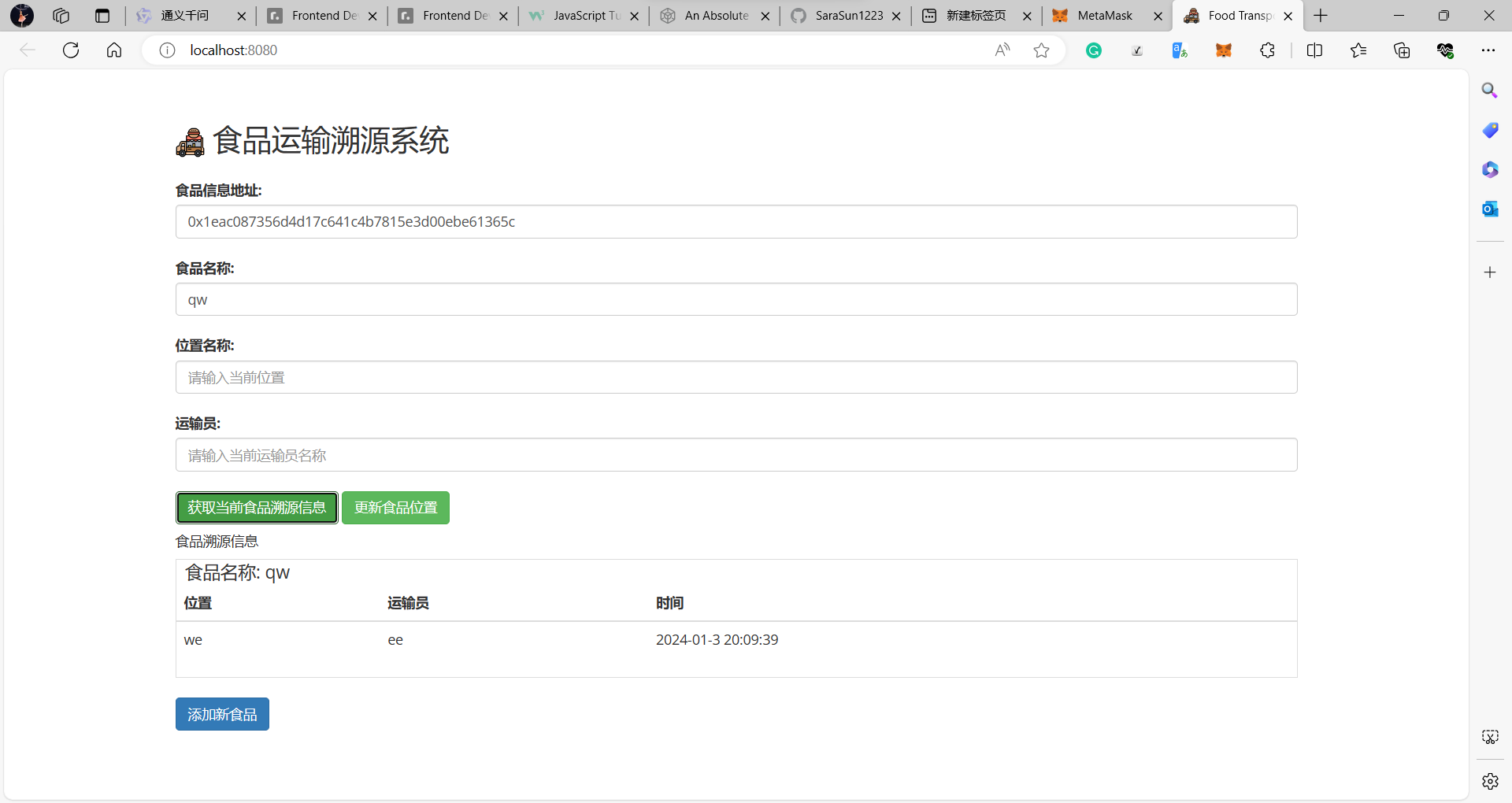


图9 获取当前食品溯源信息 可以获得当前食品的名称、位置、运输员和时间

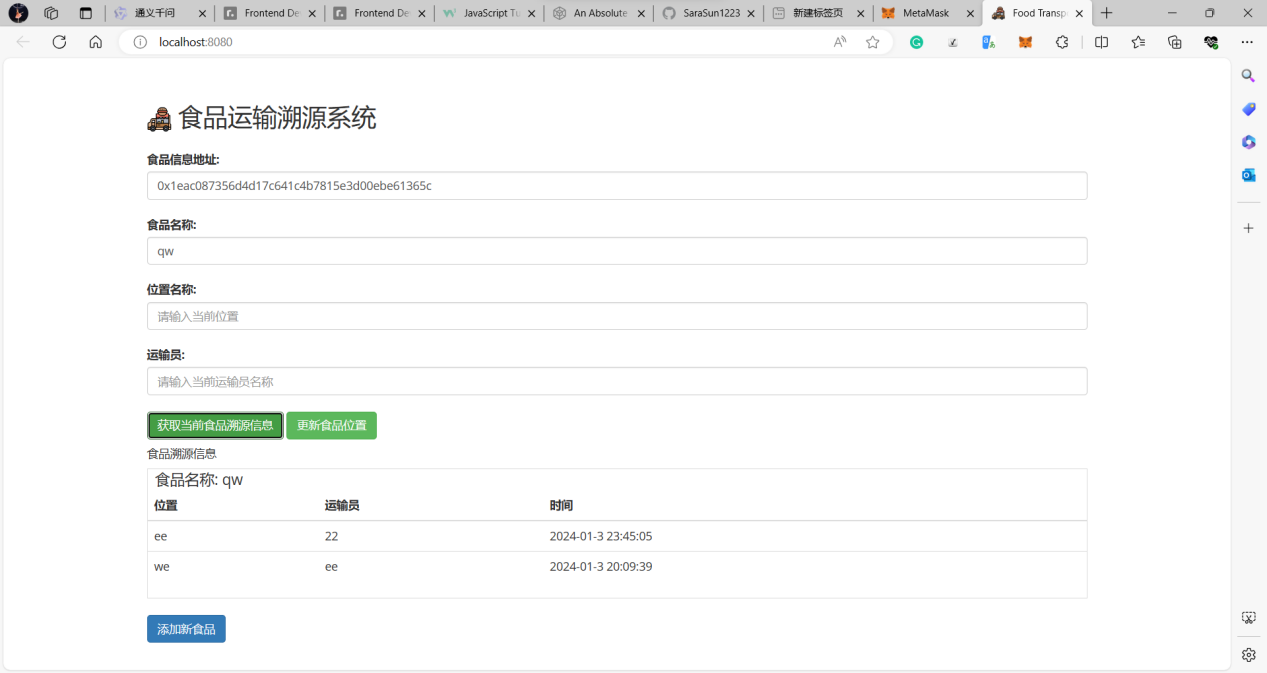
、

图10 更新食品位置 再次查询可获得当前食品位置和历史食品溯源记录

最终，在该自建的以太坊私链环境中，通过部署的 Dapp 进行了模拟的食品流转信息添加和查询等实际操作，验证了 FoodTransportation 智能合约的有效性以及整个基于以太坊的防伪溯源系统的可行性。这一系列步骤不仅充分检验了系统功能的完整性和正确性，而且达到了深入探究智能合约编写、部署技术，以及如何基于以太坊平台构建实际应用场景的教学目标。

## 调试心得

在调试过程中，深入理解了以太坊智能合约的工作原理，体会到了区块链技术在保证数据安全、透明方面的重要作用。同时，在处理前端与后端交互时，对 RPC 接口调用、事件监听等技术有了更实际的应用经验。此外，还注意到了在智能合约编程中，严谨的数据类型检查和边界控制对于提升系统健壮性的重要性。