

Lab7 DAPP

苏家齐 2021200834

本实验的所有代码均已同步在<https://github.com/AiRAM-S/ConferenceEnrollmentDAPP>。其中lab8文件夹为使用truffle生成的私链项目，核心合约 Enrollment.sol 实现在contracts文件夹内，test文件夹内编写了一个简单的测试合约，仅为尝试编写测试而用，其测试效果较弱。lab7-frontend文件夹为基于下发前端项目进行完善与修改的前端DAPP应用。

实验1 会议报名登记系统的基本功能与实现

本阶段需要初步编写会议报名系统的solidity代码。函数接口与逻辑与实验指导一致，在此不赘述实现逻辑，仅对指导中的问题进行回答：

- 1) 在合约的construct函数指定管理员身份。合约中有administrator成员，将该成员设置为msg.sender
- 2) 在通过 newConference 发起新会议时，首先，该函数被添加了修饰符 onlyAdministrator，该修饰符会使用 require 检查 msg.sender 是否与合约中的 administrator 一致，若一致才可以继续执行 newConference 中的内容；

require assert revert 三者都可以用于进行错误处理：

require：用于检查函数的输入参数或合约状态是否符合预期条件，只有条件满足了才会继续执行，否则会抛出异常并撤销交易，并可以输出自定义的错误消息。

assert：用法和 require 类似，用于检查程序中的不变条件，通常用于检测合约的内部逻辑错误，或者检查是否出现了异常。如果条件不满足，assert会抛出异常并撤销交易。一个 assert 和 require 的使用例子为：

```
// 转账函数：检查条件，执行转账
function transfer(address _to, uint _amount) public {
    // 使用 require() 确保转账金额有效
    require(_to != address(0), "Invalid recipient address");
    require(_amount > 0, "Amount must be greater than zero");
    require(balances[msg.sender] >= _amount, "Insufficient balance");

    // 执行转账
    balances[msg.sender] -= _amount;
    balances[_to] += _amount;

    // 触发转账事件
    emit Transfer(msg.sender, _to, _amount);
}

// 检查余额是否符合条件，使用 assert()
function checkBalance(address _account) public view returns (uint) {
    // 使用 assert() 检查余额是否为正数
    assert(balances[_account] >= 0); // 这其实是多余的，因为余额不可能是负数
    return balances[_account];
}
```

`revert`：顾名思义，该函数负责显式地撤销当前的交易并返回错误信息，但函数本身并不执行条件判断。其用法一般为

```
if(condition){  
    revert("Error:....");  
}
```

3) 简述合约中用 `memory` 和 `storage` 声明变量的区别：

`memory`存储在虚拟机的临时内存中，仅在函数生命周期内存在，函数调用结束后就会被销毁；`memory`变量可以在函数执行期间内修改，修改`memory`变量的gas消耗更低；

`storage`存储在区块链/合约的永久存储中，在合约的生命周期中存在，直到合约被销毁；`storage`变量的修改gas消耗更高。

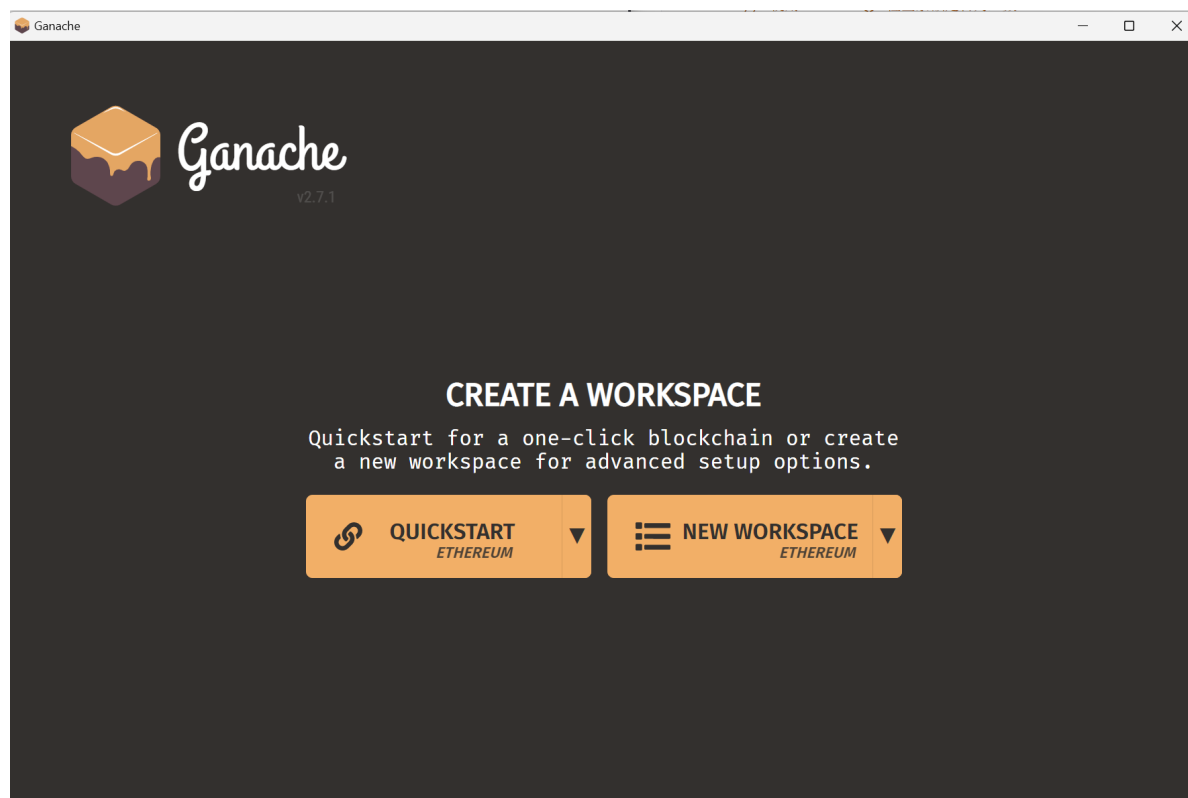
实验2 学习用Truffle 组件部署和测试合约。

本阶段需要使用Ganache搭建本地私链，并使用truffle将之前实现在remix的合约代码部署在Ganache私链上。

首先在wsl2 ubuntu20.04安装了node.js以及truffle：

```
● suzuki@LAPTOP-JTG33IP5:/mnt/d/learn/2024_Autumn/lab7dapp/instruction$ npm -v  
10.5.0  
  
● suzuki@LAPTOP-JTG33IP5:/mnt/d/learn/2024_Autumn/lab7dapp/instruction$ truffle -v  
Truffle v5.11.5 (core: 5.11.5)  
Ganache v7.9.1  
Solidity v0.5.16 (solc-js)  
Node v18.20.2  
Web3.js v1.10.0
```

而后在宿主机安装了ganache windows版本：



在lab7dapp文件夹下使用truffle初始化lab8项目：

```
● suzuki@LAPTOP-JTG33IP5:/mnt/d/learn/2024_Autumn/lab7dapp$ truffle init lab8
```

```
Starting init...
```

```
=====
```

```
> Copying project files to lab8
```

```
Init successful, sweet!
```

```
Try our scaffold commands to get started:
```

```
$ truffle create contract YourContractName # scaffold a contract
```

```
$ truffle create test YourTestName # scaffold a test
```

```
http://trufflesuite.com/docs
```

但此时truffle并没有按照实验指导生成对应的 Migrations.sol 和 1_initial_migration.js 文件。查看truffle官方的示例项目 metacoin，发现其目录中同样没有这两个文件，可能的原因是目前的版本已经不需要Migrations合约进行迁移了。

将之前编写的 Enrollment.sol 复制到contracts文件夹下，并为其编写了测试合约

TestEnrollment.sol。按照实验指导编写了 1_deploy_contracts.js。由于在这里还用到了 ConvertLib合约，保险起见在 contracts 文件夹下参考 metacoin 编写了 ConvertLib.sol（之后其实并没有用到）：

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.13;

// A library is like a contract with reusable code, which can be called by other
contracts.
// Deploying common code can reduce gas costs.
library ConvertLib{
    function convert(uint amount, uint conversionRate) public pure returns (uint
convertedAmount)
    {
        return amount * conversionRate;
    }
}
```

而后要尝试将truffle中的代码部署到ganache的私链上。由于我的truffle运行在WSL2上，而我的ganache运行在宿主机上，所以首先需要在宿主机上允许来自WSL和7545的流量（以下以管理员身份启动powershell并执行）。

```
New-NetFirewallRule -DisplayName "Allow Ganache 7545" -Direction Inbound -
LocalPort 7545 -Protocol TCP -Action Allow
```

```
New-NetFirewallRule -DisplayName "Ganache on WSL" -Direction Inbound -LocalPort
7545 -Protocol TCP -Action Allow
```

而后修改truffle-config.js中的host，将其从localhost修改为WSL的IPv4地址，可以在宿主机上运行 ipconfig 查看：

以太网适配器 vEthernet (WSL (Hyper-V firewall)):

连接特定的 DNS 后缀 :
本地链接 IPv6 地址. : fe80::350a:6b8f:66b4:c8d%64
IPv4 地址 : 172.27.208.1
子网掩码 : 255.255.240.0
默认网关. :

最后得到 `truffle-config.js` 文件如下（仅保留了修改的部分，将solc从0.8.20降级至了0.8.13，用于解决'hit-an-invalid-opcode-while-deploying'报错）：

```
module.exports = {  
  
  networks: {  
    development: {  
      // host: "127.0.0.1",      // Localhost (default: none)  
      host: "172.27.208.1",  
      port: 7545,              // Standard Ethereum port (default: none)  
      network_id: "*",        // Any network (default: none)  
    },  
  },  
  ...  
  // Configure your compilers  
  compilers: {  
    solc: {  
      version: "0.8.13",      // Fetch exact version from solc-bin (default:  
truffle's version)  
    }  
  },  
};
```

在Ganache利用 `truffle-config.js` 运行了一个以太坊私链：

The screenshot shows the Ganache application window. At the top, there's a navigation bar with icons for ACCOUNTS, BLOCKS, TRANSACTIONS, CONTRACTS, EVENTS, and LOGS. Below this is a status bar displaying various metrics like CURRENT BLOCK, GAS PRICE, GAS LIMIT, HARDFORK, NETWORK ID, RPC SERVER, MINING STATUS, and WORKSPACE. The main area shows the MNEMONIC and HD PATH. Below that is a table with 7 rows, each representing an account with its ADDRESS, BALANCE, TX COUNT, INDEX, and a key icon.

ADDRESS	BALANCE	TX COUNT	INDEX	
0x6CF5c8ab886f04efC178B616676629C9c6e5B586	100.00 ETH	0	0	🔑
0x1D6205658a394F3aD24185026665a2e94291Bdac	100.00 ETH	0	1	🔑
0x0F4707Ab5EF4C5Fac6B55b56a717Ca282ED530d2	100.00 ETH	0	2	🔑
0xA75602004Ea89A249a5aE8ea8b0293dacdf01Faf	100.00 ETH	0	3	🔑
0xd21418c21454c89117896379c42FAA0f6860582A	100.00 ETH	0	4	🔑
0xf9CbE866B02374DF6D9787fCE97a650aFFAc3B08	100.00 ETH	0	5	🔑
0xD1b9bcee92BE0BbD5fb5e3DD25ACbCfaBA6907B	100.00 ETH	0	6	🔑

接下来使用 `truffle migrate` 命令将编写好的合约部署到ganache私链上。

```
● suzuki@LAPTOP-JTG33IP5:/mnt/d/learn/2024_Autumn/lab7dapp/lab8$ truffle migrate
```

```
Compiling your contracts...
=====
> Compiling ./contracts/ConvertLib.sol
> Compiling ./contracts/Enrollment.sol
> Artifacts written to /mnt/d/learn/2024_Autumn/lab7dapp/lab8/build/contracts
> Compiled successfully using:
  - solc: 0.8.13+commit.abaa5c0e.Emscripten.clang

Starting migrations...
=====
> Network name:    'development'
> Network id:     5777
> Block gas limit: 6721975 (0x6691b7)
```

```
Deploying 'ConvertLib'
-----
> transaction hash: 0x17e7e040f9960c98c361a6df584e027aca029c2b1c2227c7534a54dd013b5254
> Blocks: 0
> contract address: 0x16D225D6837Fe56498B399f6F8375b938E0B9D86
> block number: 3
> block timestamp: 1733457518
> account: 0x6CF5c8ab886f04efC178B616676629C9c6e5B586
> balance: 99.998505181049176272
> gas used: 157568 (0x26780)
> gas price: 3.178361405 gwei
> value sent: 0 ETH
> total cost: 0.00050080804986304 ETH
```

```
Deploying 'Enrollment'
-----
> transaction hash: 0x4aeff6f8e93fdbcd7d9ab76a046c77ddb1c022554659015dd396c11562c462bf4
> Blocks: 0
> contract address: 0xA5e02608d886394873C9eed3EB7664EB6772FAA1
> block number: 4
> block timestamp: 1733457518
> account: 0x6CF5c8ab886f04efC178B616676629C9c6e5B586
> balance: 99.991288284035544064
> gas used: 2329879 (0x238d17)
> gas price: 3.097541552 gwei
> value sent: 0 ETH
> total cost: 0.007216897013632208 ETH
```

```
> Saving artifacts
-----
```

```
> Total cost: 0.007717705063495248 ETH
```

```
Summary
```

```
=====
```

```
> Total deployments: 2
> Final cost: 0.007717705063495248 ETH
```

可以看到两个合约都被成功部署，下面利用 `truffle test` 简单测试几个函数的功能：

```
● suzuki@LAPTOP-JTG33IP5:/mnt/d/learn/2024_Autumn/lab7dapp/lab8$ truffle test
Using network 'development'.
```

```
Compiling your contracts...
=====
> Compiling ./contracts/Enrollment.sol
> Compiling ./test/TestEnrollment.sol
> Artifacts written to /tmp/test--33564-XUYmsZJAHqME
> Compiled successfully using:
  - solc: 0.8.13+commit.abaa5c0e.Emscripten.clang
```

```
TestEnrollment
  ✓ testSignUp (209ms)
  ✓ testDelegate (154ms)
  ✓ testEnroll (262ms)
```

```
3 passing (7s)
```

练习2：观察合约的部署过程

方便起见，这里利用ganache重新搭建了一条私链，而后使用truffle migrate进行合约部署。

```
● suzuki@LAPTOP-JTG33IP5: /mnt/d/learn/2024_Autumn/lab7dapp/lab8$ truffle migrate

Compiling your contracts...
=====
> Compiling ./contracts/Enrollment.sol
> Artifacts written to /mnt/d/learn/2024_Autumn/lab7dapp/lab8/build/contracts
> Compiled successfully using:
   - solc: 0.8.13+commit.abaa5c0e.Emscripten.clang

Starting migrations...
=====
> Network name:    'development'
> Network id:      5777
> Block gas limit: 6721975 (0x6691b7)

1_deploy_contracts.js
=====

  Replacing 'ConvertLib'
  -----
  > transaction hash: 0xa3ada156ac442844a8994e93d7708ef6010a18e005457e298ff4011bca628415
  > Blocks: 0        Seconds: 0
  > contract address: 0xd050e152C4A3F2809FF105c35E54bC888AD35F0e
  > block number:     1
  > block timestamp:  1733463290
  > account:          0xC47713610d90d8E4f0e28Ed723dD0e1bA697497a
  > balance:          99.999468208
  > gas used:          157568 (0x26780)
  > gas price:         3.375 gwei
  > value sent:        0 ETH
  > total cost:        0.000531792 ETH

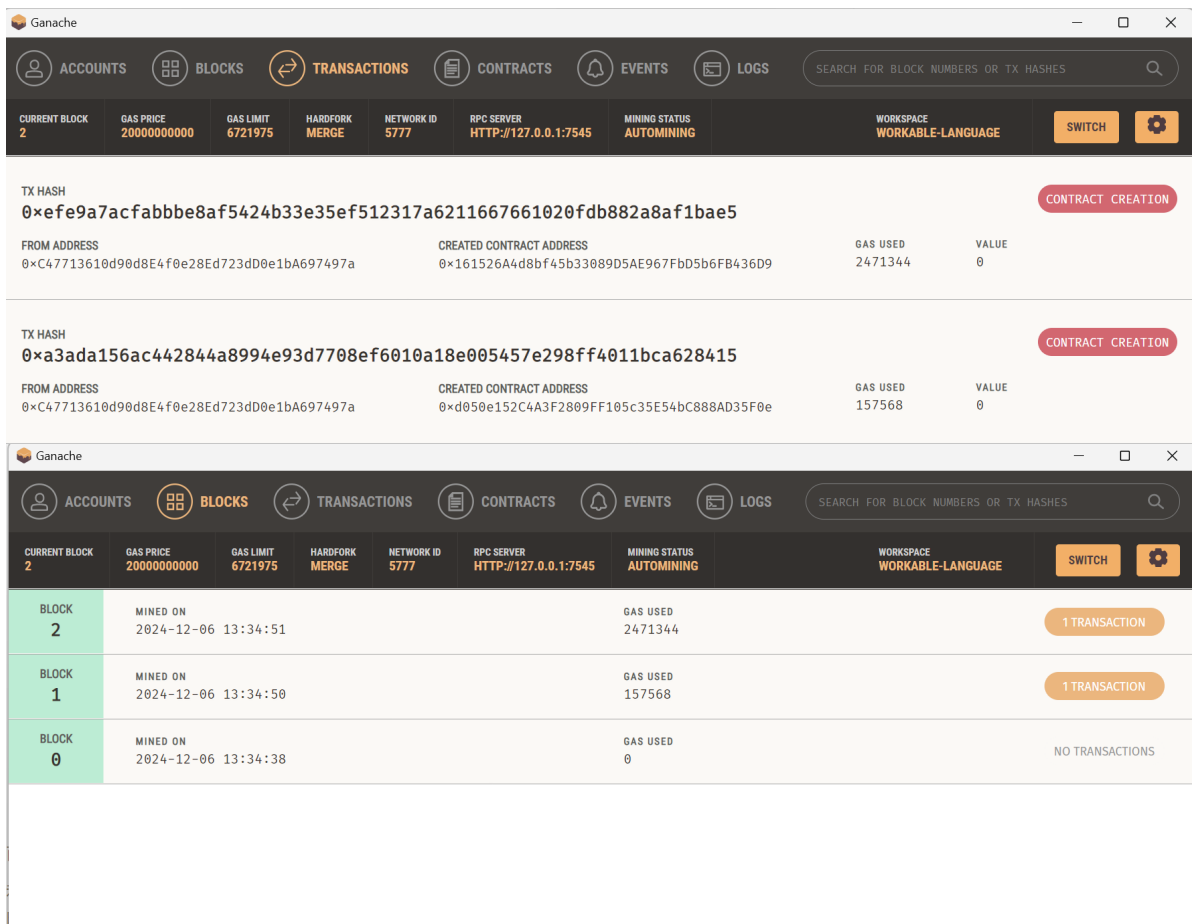
  Replacing 'Enrollment'
  -----
  > transaction hash: 0xef9a7acfabbbe8af5424b33e35ef512317a6211667661020fdb882a8af1bae5
  > Blocks: 0        Seconds: 0
  > contract address: 0x161526A4d8bf45b33089D5AE967FbD5b6FB436D9
  > block number:     2
  > block timestamp:  1733463291
  > account:          0xC47713610d90d8E4f0e28Ed723dD0e1bA697497a
  > balance:          99.991385053035753616
  > gas used:          2471344 (0x25b5b0)
  > gas price:         3.270752661 gwei
  > value sent:        0 ETH
  > total cost:        0.008083154964246384 ETH

  > Saving artifacts
  -----
  > Total cost:        0.008614946964246384 ETH

Summary
=====
> Total deployments:  2
> Final cost:         0.008614946964246384 ETH
```

两个合约的部署交易被打包在了1号和2号两个区块里。

利用truffle和ganache的部署过程包括：truffle对需要部署的合约进行编译，生成字节码后，会从Ganache的当前地址向0x0发送包含字节码的一笔交易，这笔交易会被（Ganache自动）打包进区块，而后得到合约的唯一地址。下图是Ganache上对应的这两笔交易，以及它们对应的区块。



通过web3.js连接前端

这一阶段的实验目标是使用下发的lab7-frontend框架，将前端用户行为与合约调用串联起来。下发框架是一个基于 `create-react-app` 搭建的react前端界面，核心修改包括：

1. `src/components`：为前端上每一个组件都设置了一个文件夹。对于每一个组件，需要在 `mapDispatchToProps` 函数中实现 `submit` 函数，将表单中填入的数据进行处理后，调用 `contract.methods.xxx.send()` 函数（表单类为send，非表单为call）调用对应的合约函数；
2. `contracts/contract.js`，需要在其中指定合约的部署地址，以及合约的ABI。

实验过程中，由于版本更新，对上述文件夹中部分函数依据报错信息进行了重构。例如，下发代码中使用 `window.web3.eth.accounts[0]` 获取当前用户的地址，用于支付交易费用等，然而web3在 `contract.js` 中的初始化方法已经被弃用，这会导致metamask钱包无法连接到前端app上，并使得web3无法正常初始化，进而影响所有web3的方法调用，同时使用accounts直接获取地址的方式也不被支持。因此在 `contract.js` 里修改了web3的初始化方法，改用 `window.ethereum` 进行初始化：

```
// window.web3.currentProvider为当前浏览器的web3 Provider
const web3 = new Web3(window.ethereum);
try {
  window.ethereum.request({ method: "eth_requestAccounts" });
  console.log("Ethereum accounts authorized");
} catch (error) {
  console.error("User denied account access:", error);
}
// 导出合约实例
export default new web3.eth.Contract(abi, address);
export { web3 };
```

同时修改获取地址的方法为(以delegate为例) 使用 `getAccounts` 获取，并逐步对Promise类型的返回值进行处理：

```

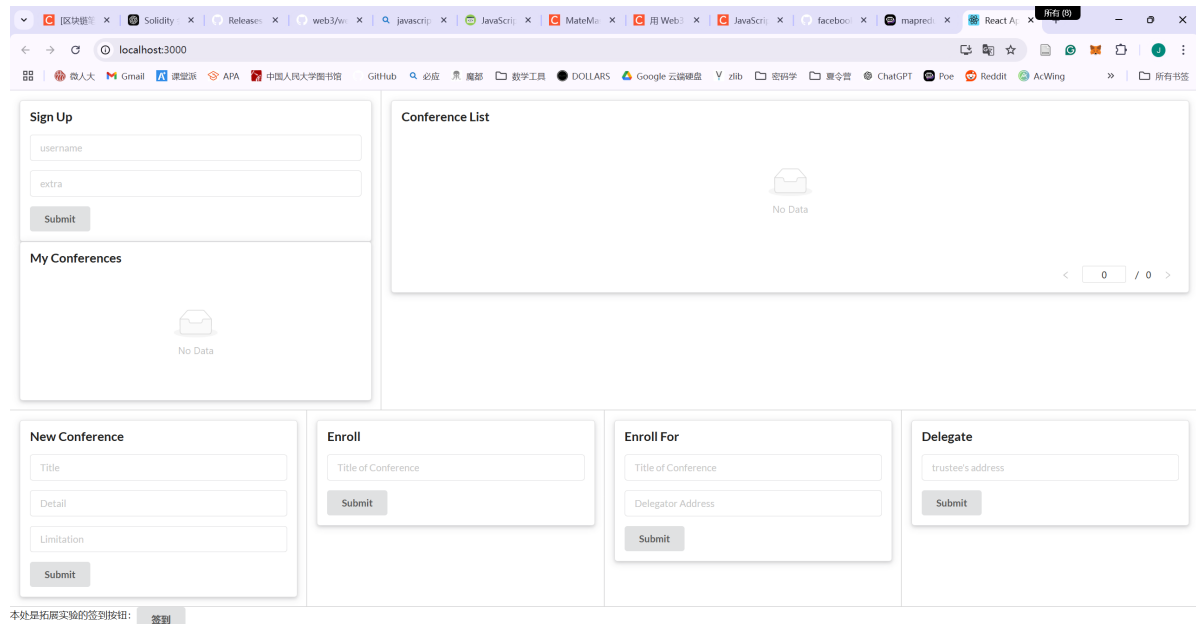
submit(address){
  web3.eth.getAccounts()
  .then(accounts => {
    if(accounts.length === 0) {
      throw new Error('No accounts found');
    }
    const fromAddress = accounts[0];
    console.log("Send Delegate from " + fromAddress);
    return contract.methods.delegate(address).send({from : fromAddress});
  })
  .then((res) => console.log(res));
  dispatch({
    type: 'submit_delegate'
  });
},

```

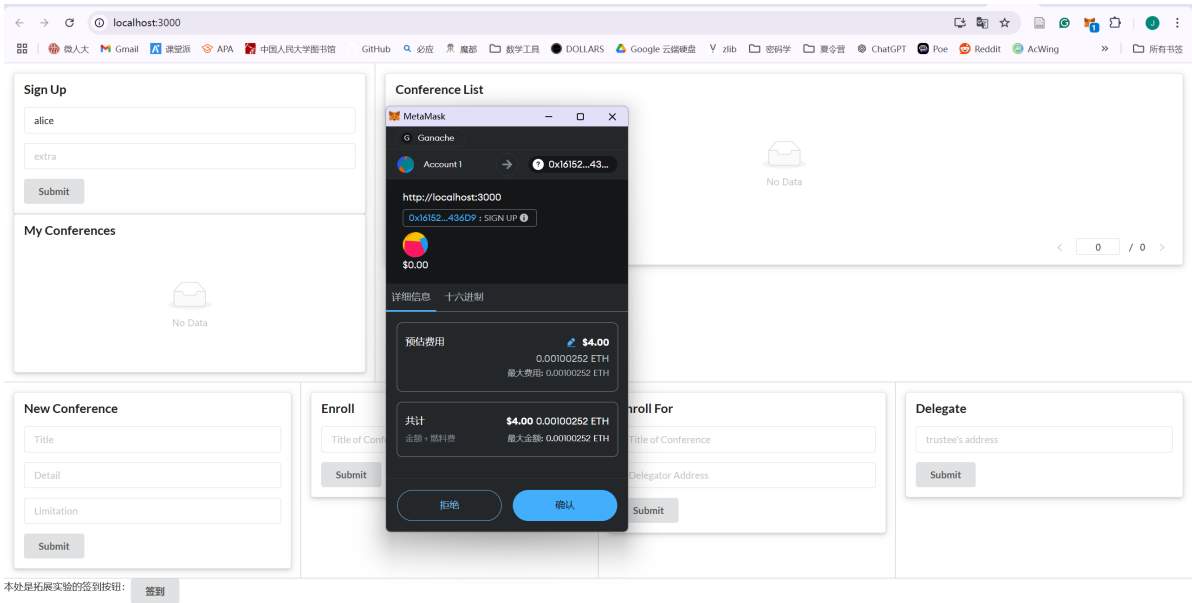
类似此处的改动在每个组件对应的 `index.js` 均有，此处不做赘述，但所有的修改都局限在前述的两个文件（夹）内。

除此之外，在提供的前端接口方面，修改了enroll for的接收参数，原先是通过 `Title of Conference` 和 `Username` 确定代为哪个用户报名会议，然而这将导致一个问题：确定代理关系时，使用的是被代理方的地址，同时被代理方未必通过SignUp注册了用户名，因此如果通过 `Username` 定位 `Enroll For` 的报名人是不太合理的，在这里改为了使用被代理方的地址作为输入参数。（这里其实还有一个问题：当用户A代为用户B报名某个会议后，用户B能否在前端界面上看到这个会议呢？按照目前的设计是看不到的，因为用户B没有主动SignUp，因此在Participants里不会有用户B的信息，也就没有用户B的报名会议信息）。

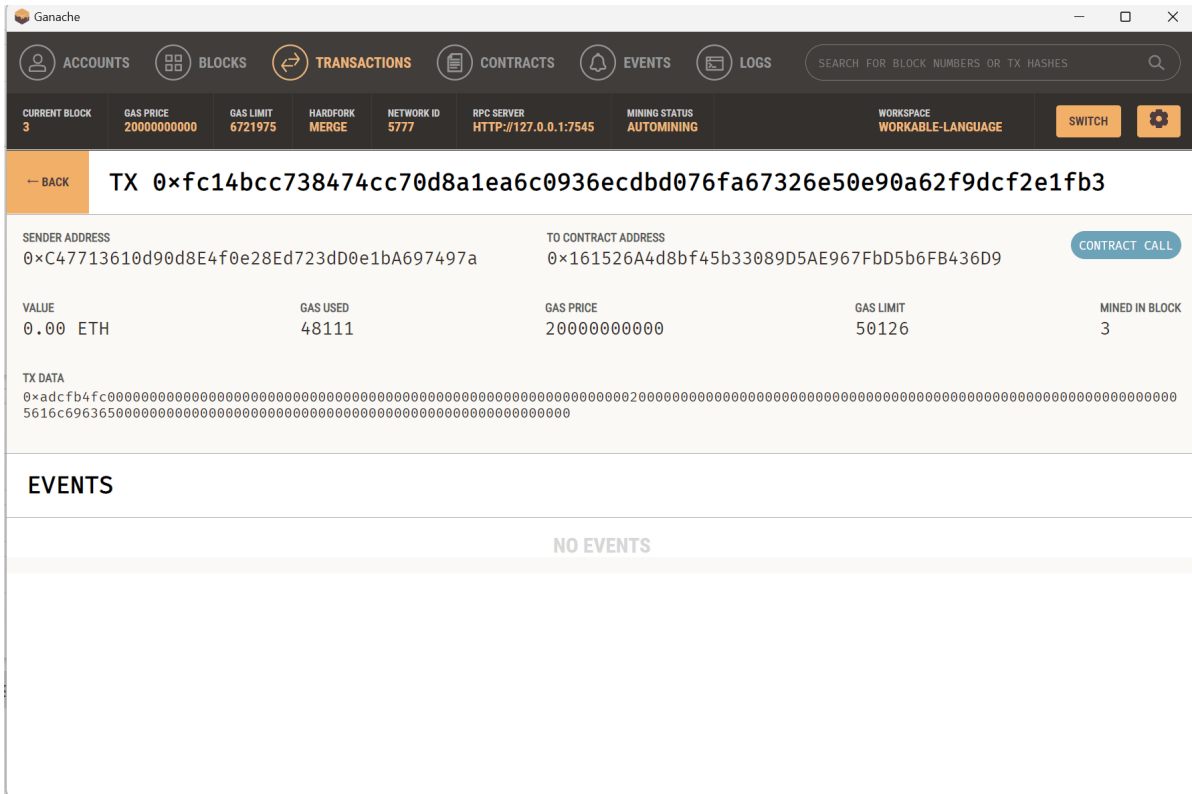
做完以上修改后，使用 `npm start` 运行前端APP。这里我使用的node版本为16.20.2，过高的版本会导致错误。最终运行的前端界面如下（此前，在右上角的metamask插件中已经利用RPC SERVER等信息连接到Ganache私链，并登录上了Ganache上的第一个账户）：



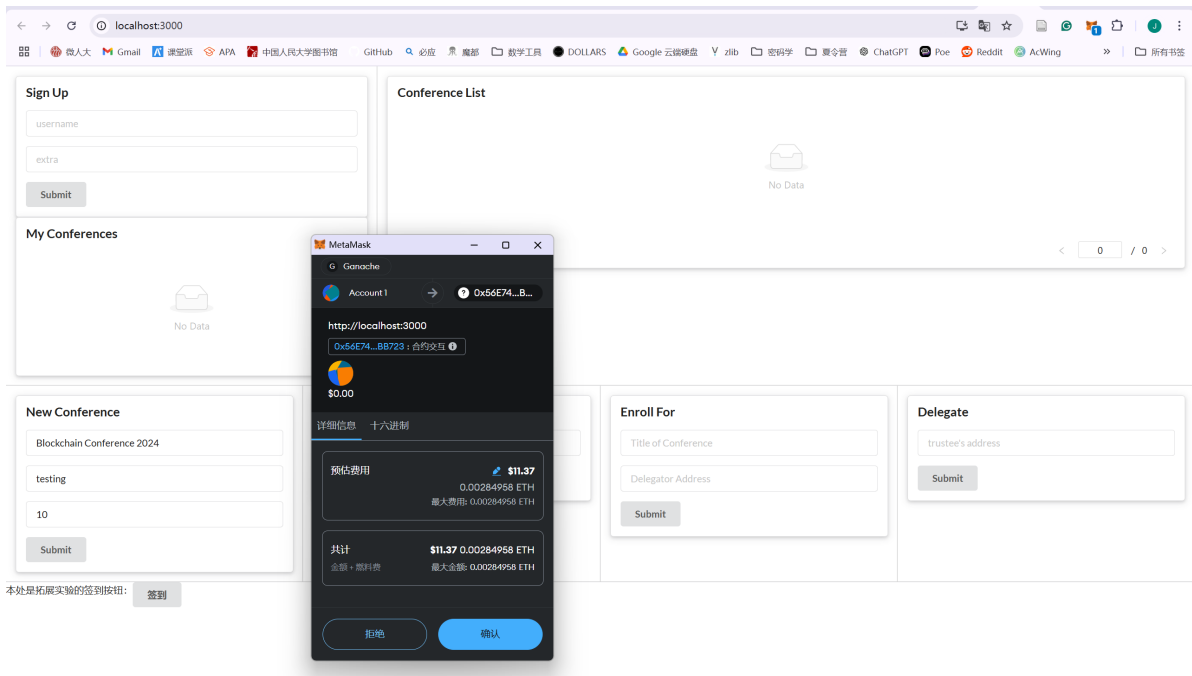
尝试使用SignUp组件进行注册，可以看到会弹出Metamask的交易插件，点击确认即签署交易，调用SignUp函数。



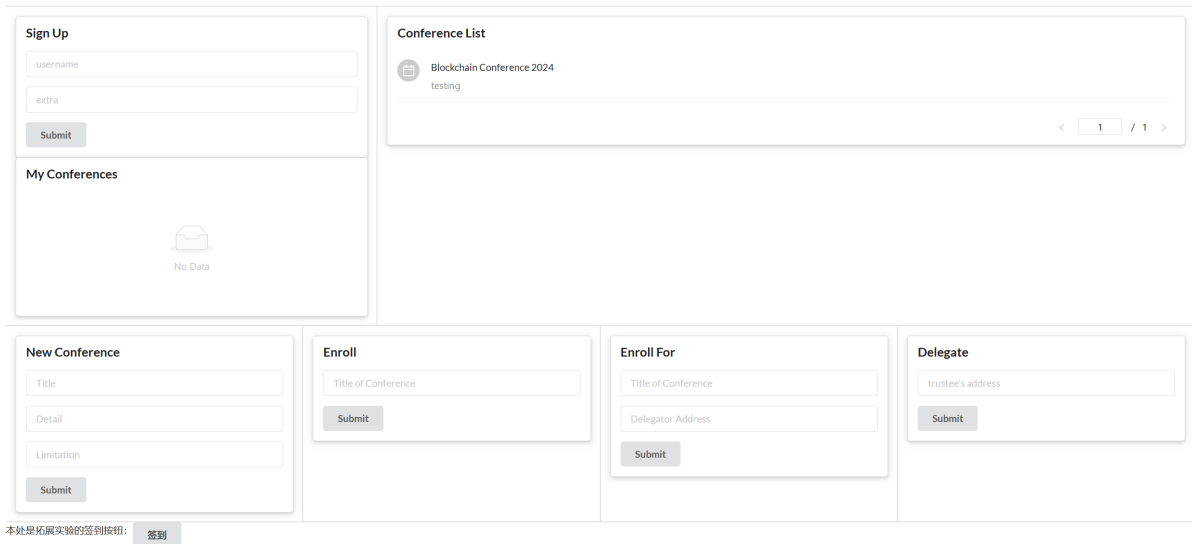
这笔交易的详细信息可以在Ganache中查找到：



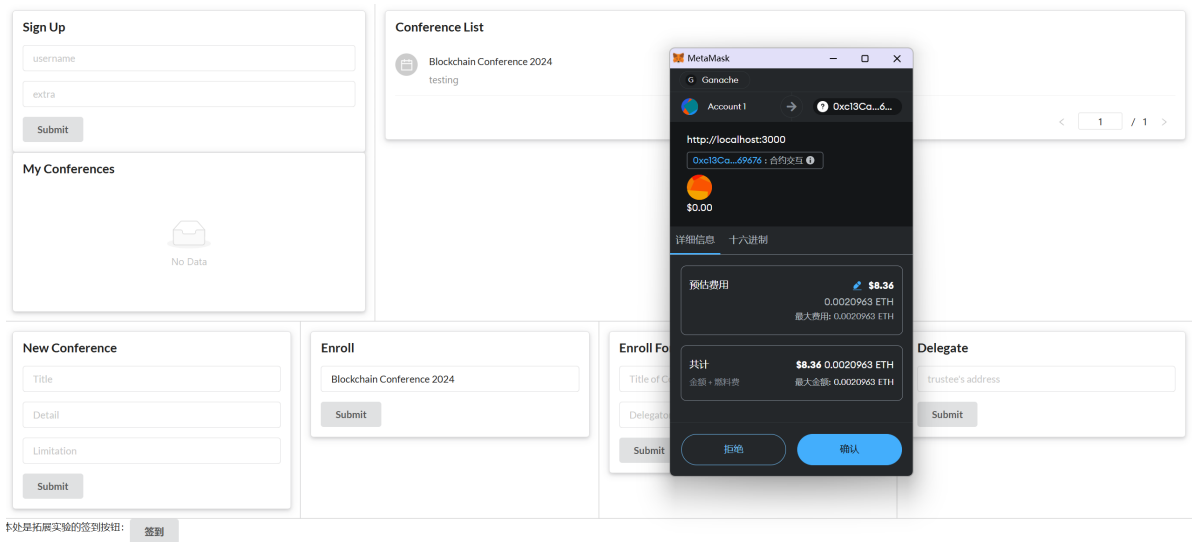
而后尝试创建新的会议，同样弹出交易确认界面，按确认签署交易：



交易完成后，由于设置了对 `NewConference` 事件的监听，“Conference List”组件立刻进行了刷新，并显示出这个新创建的会议：



此处尝试报名刚刚创建的会议：



同样地，MyConferences组件监听到了报名会议的事件，进行刷新，显示当前报名的会议。这里可以看到有一个bug，即没有对多次报名同一会议进行检查，一方面会导致MyConferences里出现多个同一会议，另一方面也可能导致会议的报名人数被重复统计。受限于时间，这一bug暂时没有修复。

