区块链设计 - 产品溯源

专　　业：软　件　工　程

指　导　教　师：尹　长　青

年　　级：大学三年级

学　　号：2153061

姓　　名：谢　嘉　麒

1.区块链设计思路

1.1.总体设计思路

设置区块链实现溯源功能，本质上还是需要将设计落实到区块链的去中心化功能。区块链的本质是一个去中心化的分布式账本，记录着所有交易的历史，通过加密技术保证数据的不可篡改性和安全性。在设计本系统时，首先应考虑区块的数据结构、加密方式、共识机制、网络节点的组织方式。

1. **区块设计**：每个区块内部包含哈希值，上一区块的哈希值，`merkle`根，时间戳和随机数。通过哈希值确保区块之间的连贯性和整个链的完整性，必要时在区块内添加交易信息。
2. **交易信息**：由于这是一个产品溯源系统，因此交易信息内要包括交易`id`，发送者密钥，发送者名称，接收者密钥，接收者名称，交易金额，交易信息和签名。
3. **加密技术**：使用SHA-256加密算法来保证交易数据的安全和区块的不可篡改性。此外，使用公钥私钥机制来进行交易验证，确保交易双方的身份验证和非抵赖性。
4. **共识机制**：设计工作量证明PoW共识机制，以实现网络中不同节点间的数据一致性。这是区块链网络能够稳定运行的关键。
5. **交易过程**：由于比特币系统没有余额的概念，故使用其UTXO模型，用未使用过的交易输出来存储系统虚拟币余额。
6. **网络节点管理**：节点应能自动同步最新的区块链数据，并参与区块的验证和生成过程。节点的去中心化配置有助于提高网络的抗攻击能力和故障容错性。
7. **溯源功能实现**：在设计区块链时，考虑利用区块链交易信息记录交易链条，实现产品从生产到销售的全过程透明化，确保信息的真实性和可追踪性。

本系统将整个区块链分为区块类，区块链类，参与者类，交易类和进账出账类实现，本说明文档将按照类为框架区分，从数据结构和算法两个方面介绍设计方案。

1.2.区块单元设计 - Block

1.2.1.区块单元数据结构设计

由于区块链本质上是类似于链表，将区块进行联系的集合，因此优先对区块单元进行设计。

系统设定每一个区块中，需要包含自己的哈希值，上一个区块的哈希值，`merkle`根，时间戳和随机数和交易信息。其中：

1. **自己的哈希值（hash）**：使用`SHA256`算法，加密前一个区块的哈希值，时间戳，工作量和梅克尔根的和，因此如果前一个区块的哈希值，区块时间戳，工作量和梅克尔根任意一个发生变化或被篡改，整个区块的哈希值就会改变，后续所有区块的哈希值也会变化，甚至产生“断链”，因此计算和对比哈希值是检查当前区块链是否合法的重要方法，同时因为篡改数据会改变区块哈希值并破坏整个区块链，这种设置也避免数据被恶意篡改，

2. **前一个区块的哈希值（previousHash）**：在区块中定义前一个区块的哈希值，一方面是为了作为计算自己哈希值的参数进行加密，提高区块链安全性和稳定性；另一方面也作为区块链串联的依据。
3. **梅克尔根（MerkleRoot）**：用于表示区块中所有交易的数据完整性。它是由区块中所有交易的哈希值构成的树状数据结构中的根节点的哈希值。这个树状结构称为梅克尔树（Merkle Tree），也被称为哈希树。通过将区块中的所有交易哈希值两两配对，然后对这些配对的哈希值再次进行哈希运算，直到最后只剩下一个哈希值，即梅克尔根。这个过程确保了每个区块中的交易都被包含在一个可验证的数据结构中，并且保证了区块数据的完整性。梅克尔根的存在使得任何人都可以轻松验证特定交易是否包含在一个区块中，同时还能够确保区块链网络中的数据没有被篡改。
4. **时间戳（timeStamp）**：用于计算区块哈希值，保证区块链安全性。
5. **随机数（nonce）**：随机数用于在工作量证明算法中使用，寻找一个合适的随机数值，使区块的哈希值满足一定条件。其作用在于增加区块哈希的随机性，增加寻找符合条件的哈希值的难度，防止恶意篡改区块的行为，保证的区块链的安全性和不可篡改性。
6. **交易信息（transactions）**：为区块添加交易信息，进行记录。交易信息类需要单独定义。

1.2.2.区块单元方法设计

为了保证区块单元能够正常工作，需要设定包括加密算法，梅克尔根计算，交易信息添加等方法。

1. **加密算法**：由于本项目使用 `JAVA` 语言进行后端实现，而其加密方式有包括 `BASE`， `MD` 等在内的多种。本项目选择使用安全散列算法即 `SHA256` 进行加密。这种算法具有数据少量更改会导致哈希值大规模更改的特点，而且值大小为256位，因此这一加密算法一方面大小合适，另一方面特点也适合区块链的加密工作。使用 `MessageDigest.getInstance("SHA-256")` 获取库加密算法，并使用 `.digest(input.getBytes(StandardCharsets.UTF_8))` 获取字节数组的加密哈希，最终利用 `HexFormat.of().formatHex(encodedhash)` 将其转化为可读格式。
2. **梅克尔根算法**：首先，将每个交易的ID提取并存储在一个列表中。然后，这个列表的元素（交易ID）通过配对并进行哈希处理来逐层简化。对于列表中的每对相邻元素，将它们的ID连接起来，并使用SHA-256算法进行哈希处理。如果列表中的元素数量为奇数，最后一个元素会被复制并与自身哈希处理以形成新的对。这个过程重复进行，直到只剩下一个元素，即Merkle树的根哈希值，最终返回这个值。这种方法有效地确保了数据的完整性和不可篡改性。
3. **挖矿算法**：查阅资料得知，普通电脑对于挖矿难度1和2可以轻松算出，但难度3则需要几秒时间，在莱特币系统中难度大概在442592左右，而比特币挖矿则需要十分钟左右。因此，系统设定挖矿难度为3，首先计算区块交易的梅克尔根，根据传入的难度参数，创建一个目标哈希值，要求新区块的哈希值前几位必须为0。之后，在一个循环中，通过不断尝试增加随机数值 `nonce`，计算哈希值，直到找到符合要求的哈希值，其前缀必须有足够数量的零，满足挖矿的难度要求，确保新的区块被添加到区块链中符合安全性和共识机制的要求。
4. **交易添加算法**：需要为区块添加交易，在添加时对交易进行计算（如金额进账与出账），完成交易过程后将这一次的交易信息添加到区块的交易信息数据结构中。

1.3.区块链类设计 - BlockChain

1.3.1.区块链数据结构设计

区块链数据结构较为简单，在`JAVA`语言实现时只需要定义一个`ArrayList<Block>`类型的变量即可。为了使用比特币的UTXO交易机制，还需要定义一个类型为`HashMap<String, OutputCase>`的`HashMap`数据，此处不再赘述，后文区块交易部分会进行介绍。

1.3.2.区块链方法设计

区块链方法主要包括，在链中添加区块，找到区块链末尾块的哈希值和检查区块链合法性的方法。

1. **添加区块方法**：将区块添加到区块链中，需要通过挖矿来找到这个区块的哈希值，之后利用`ArrayList`库的`add`方法将区块添加到区块链中即可。
2. **找到区块链末尾**：由于区块添加时需要将区块链末尾的最后一个区块的哈希值，填写到新区块的"上一区块哈希值"这一数据中，因此定位区块链末尾即定义末尾块的哈希值，通过`blockchain.get(blockchain.size() - 1).hash`语句直接得到答案即可。
3. **检查区块链合法性**：需要通过区块的哈希值来定位链是否被篡改，即下一个区块中记载的前一区块的哈希值是否能和前一区块目前的哈希值对应，通过逐对比较来得到结果。更进一步地，还可以对梅克尔根，交易信息等数据类型进行校验，保证区块链的合法性和安全性。

1.4.参与者类设计 - Participants

1.4.1.参与者数据结构设计

参与者类数据结构也较为简单，重点在于设定参与者的私钥与公钥，以及UTXO机制数据。为了满足产品溯源的需求，追踪产品流通过程中的参与者，还为其添加了参与者名称这一数据。其中：

1. **公钥publicKey**：是参与者的唯一标识地址，在交易过程中通过出示公钥来获取他人的付款，产品等。在交易时需要发送公钥来验证签名有效，是可公开的部分。

1. **私钥privateKey**：用于对交易进行签名，保证自己的交易金额的权限，是不可公开的部分。
2. **参与者名称name**：用于记录参与者的名称，为溯源展示提供更人性化的可理解内容。
3. **未花费的交易输出UTXO**：使用`HashMap<String, OutputCase>`数据结构，用于参与UTXO机制，记录未花费的交易输出。

1.4.2.参与者方法设计

在参与者类中，需要设定公钥私钥生成算法，获取参与者余额的方法和创建交易方法。其中：

1. **密钥生成算法generateKeys()**：这个方法用于生成一对ECDSA密钥，包括一个私钥和一个公钥。使用`KeyPairGenerator`生成密钥对，选择`ECDSA`算法和`prime192v1`椭圆曲线规

格。使用 `JAVA` 库函数 `KeyPairGenerator.getInstance("ECDSA","BC")` 和 `SecureRandom.getInstance("SHA1PRNG")` 可以直接得到。生成的密钥通过 `Base64.getEncoder().encodeToString` 语句以Base64编码格式存储为 `String` 类型，便于后续的使用和传输。该方法确保安全地生成并存储密钥对，是区块链身份验证和交易签名的基础。

2. **获取余额算法getBalance()**：此方法计算和返回钱包的当前余额。它遍历所有未花费的交易输出（UTXOs），检查公钥与交易接收者匹配的输出，并累计这些输出的值作为总余额。这一方法一方面用于为前端和余额查询进行铺垫，另一方面也是进行交易前检查参与者能否负担交易的重要方法。
3. **创建交易方法**：这个方法用于创建新的交易，将指定金额的资金从当前用户发送到指定的接收者。首先验证当前账户余额是否足以支付交易金额。如果足够，它将收集足够的UTXOs作为输入，并创建新交易。然后，为新交易生成数字签名以验证其有效性，并在创建新交易后更新UTXO池，移除已使用的UTXOs。这个方法是实际执行资金转移和确保交易安全的核心功能。
4. **备注**：最初并不希望将密钥存储为 `String` 类型，但是密钥类型在 `SpringBoot` 反序列化传递给前端时由于较为复杂会导致序列化失败，但本项目作为区块链学习实践项目，希望能够在前端对密钥进行可视化处理。因此，将生成的密钥转换存储为 `String` 类型，是为了方便反序列化传递给前端，在区块链实际应用中不需要多此一举。

1.5.交易信息类设计 - Transaction

1.5.1.交易信息数据结构设计

由于本次区块链实现的最终目的是实现产品溯源系统，因此在交易信息类中需要定义交易id，发送者密钥，接收者密钥，发送者名称，接收者名称，交易金额，交易备注，交易签名等信息，同时需要在交易信息中添加两个 `ArrayList` 数组，用于存储金额的进账出账信息。

1. transactionId：交易的唯一标识符，是交易内容（如交易参与者、金额等）的哈希值，用于唯一地识别每笔交易。
2. sender：发送者的公钥，用于指明交易的发起方。
3. receiver：接收者的公钥，用于指明交易的接收方。
4. senderName：发送者的名称，用于显示或记录交易发起人的姓名或别名，使交易记录更易于理解。
5. receiverName：接收者的名称，用于显示或记录交易接收人的姓名或别名。
6. value：交易涉及的金额。这个数值表明了从发送者到接收者转移的资金或资产的数量。
7. info：交易的附加信息。这个字段可以包含任何额外的数据，如交易目的、备注等，用于提供交易的更多背景信息。
8. signature：交易的数字签名。这是使用发送者的私钥生成的，用于验证交易的真实性和确保交易不可抵赖。数字签名保证了只有交易的发起者才能发起这笔交易，并且一旦发起，签名就可以用来验证交易内容未被篡改。

1.5.2.交易信息方法设计

交易信息方法则较为复杂，其中需要定义公私钥解密算法，交易过程算法，`UTXO`池更新算法，交易输入获取算法，交易签名算法等。

除此之外，由于上文提到的原因，公私钥存储为`String`类型，因此也需要将其还原为密钥类型的算法。

1. **构造函数**：此构造函数用于初始化交易对象，设定交易的基本属性如发送者、接收者、交易金额、输入案例、附加信息以及双方的名称。
2. **私钥字符串解析算法parsePrivateKeyFromString()**：这个静态方法用于从字符串形式的Base64编码中解析出私钥。使用`KeyFactory.getInstance("ECDSA")`获取加密方式，使用`keyFactory.generatePrivate( new PKCS8EncodedKeySpec(privateBytes))`进行字符串解析，得到私钥。主要用于将存储或传输的加密格式私钥转换为PrivateKey对象，以便于使用。
3. **公钥字符串解析算法parsePublickeyFromString()**：类似于`parsePrivateKeyFromString`，这个静态方法用于从Base64编码的字符串中解析出公钥，转换成PublicKey对象，区别在于使用`X509EncodedKeySpec(publicBytes)`进行公钥解析。
4. **ECDSA加密算法**：该方法用于使用ECDSA算法和指定的私钥对输入字符串进行签名，返回签名的字节数组。使用`Signature.getInstance("ECDSA", "BC")`语句获取加密方法。
5. **ECDSA公钥验证方法**：此方法使用公钥验证给定数据的ECDSA签名是否有效。使用`initVerify(publicKey)`语句进行公钥验证，用于确保交易的完整性和验证交易发起者的身份。
6. **交易处理算法**：这个函数用于处理交易。首先，它验证交易的数字签名是否有效，然后检查交易的输入是否与有效的未花费交易输出（UTXO）相匹配。接着，它计算交易的输入值和输出值，并确保输出值不大于输入值。最后，它创建交易的输出，并更新UTXO池以反映新的交易。
7. **UTXO池更新算法**：该方法负责更新UTXO池，包括将新的输出添加到UTXO池中以及从UTXO池中移除已经使用的输入。
8. **UTXO总值计算方法**：计算并返回交易输入案例中的所有UTXO的总值。这个方法用于在创建交易输出之前，确认所有输入的资金总额。
9. **数字签名生成算法**：此方法用于生成当前交易的数字签名。它首先解析私钥，然后对交易的核心数据（包括发送者、接收者和金额）应用ECDSA签名算法。生成的签名存储在交易对象的`signature`属性中。

1.6.交易输出类设计 - OutputCase

系统采用比特币的`UTXO`机制，需要封装交易输出相关信息。输出不仅仅定义了金额的接收方，也实质上定义了新的资金所有权。这些输出后继可以被用作新交易的输入，形成资金流动和交易链的基础。此外，每个输出作为UTXO（未花费交易输出）的一部分，构成了区块链网络中的主要账本结构。UTXO集合是判断账户余额和验证新交易合法性的关键数据源。

1.6.1.交易输出数据结构设计

1. **id**：输出的唯一标识符，通过哈希函数SHA-256生成，使用接收者的地址、输出值和父交易ID作为输入。这个ID帮助唯一地标识每一个输出，使其可以被后续的交易作为输入引用。
2. **接收者公钥**：接收者的公钥，指明了这些资金的新的合法所有者。
3. **输出币数**：输出中包含的虚拟币的数量，表示接收者所获得的资金量，用于定向资金的流向和分配。
4. **父交易id**：这个输出所属的父交易的ID，说明这个输出是在哪一笔交易中创建的，有助于追踪资金的来源。

1.7.交易来源类设计 - InputCase

系统为了在创建新交易时表示交易的来源，设计了交易来源类。根据`UTXO`机制规定，每笔新交易必须来源于先前的交易输出（即UTXOs）。通过列出一个或多个输入，交易指明了资金具体是从哪些先前的输出中来的。

- **资金链的追踪和验证**：每个输入必须清楚地指出其资金来源，即指定它所使用的UTXO。这允许网络节点验证交易的有效性，确保交易中的资金确实存在且未被之前的交易已使用。
- **防止双重支付**：通过引用特定的UTXO，并在交易完成后将其标记为已花费，系统防止了资金的双重支付。这是区块链安全性和完整性的关键组成部分。
- **简化交易验证**：因为每个输入都直接引用一个特定的UTXO，这简化了交易验证过程。节点可以快速检查引用的UTXO是否有效，以及交易中声明的资金是否与这些UTXO中的金额相匹配。

1.7.1.交易来源数据结构

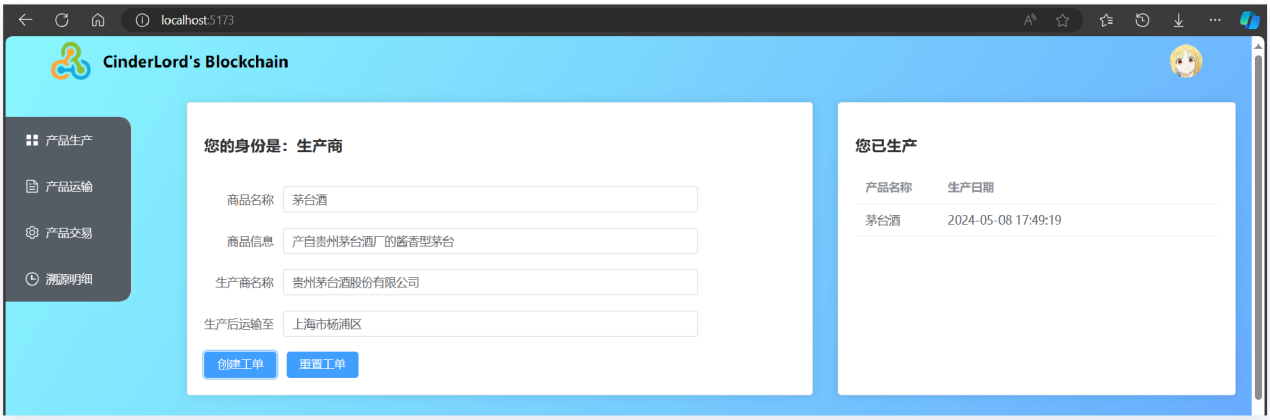
1. **引用输出标识id**：使用字符串数据，表示这个输入所引用的输出的唯一标识符（ID）。每个输入都对应于之前某个交易的输出，通过这个ID可以找到那个输出，形成资金流的连续性。
2. **未花费交易输出UTXO**：这是一个`OutputCase`对象，存储实际未花费的交易输出。`UTXO`代表了引用的输出的详细信息，包括接收者、金额和父交易ID。这个对象实质上是对`transactionOutputId`的具体化，提供了更多关于资金来源的信息。

2.程序界面说明

本次程序主要目的是进行区块链溯源系统的模拟实现，从生产商，运输商，交易者和买家溯源四个角度，分别设定产品生产，产品运输，产品交易，溯源明细四个页面进行程序实现。

2.1.产品生产页

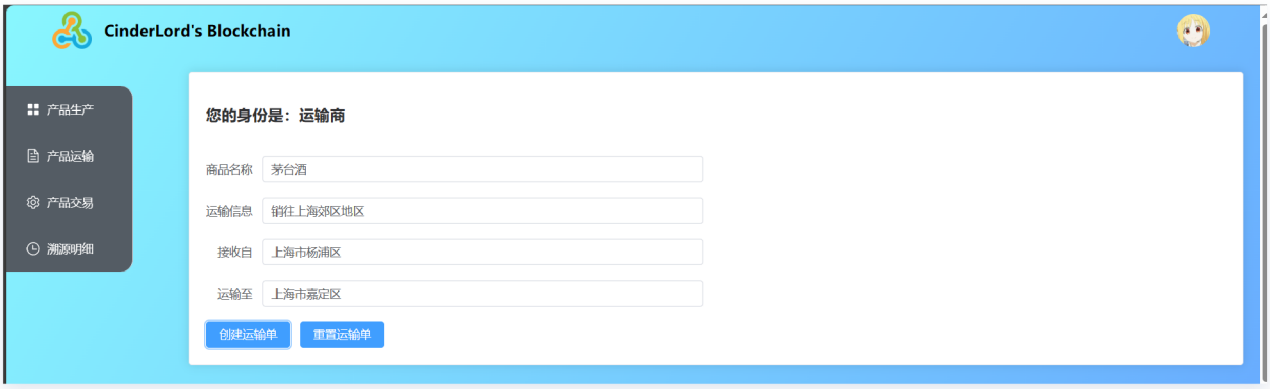
在产品生产页，生产商能够输入商品名称，商品信息，生产商名称和生产后第一运输商名称进行工单创建，此处也会显示生产商已经生产的产品列表。



单击创建工单后，处于实践教学目的，系统会自动返回新创建的区块和交易信息，并展示在网页下方：

2.2.产品运输页

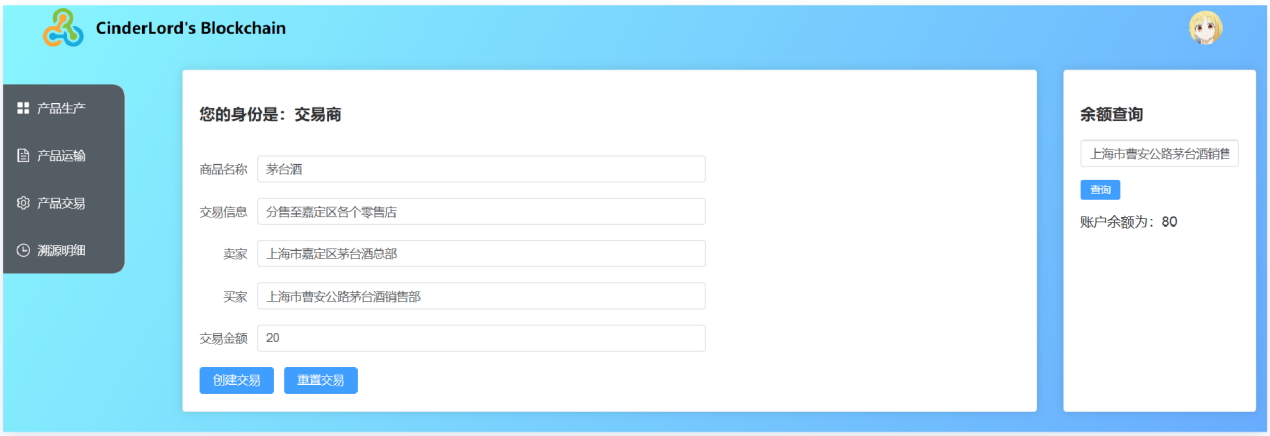
在产品运输页，用户能够输入商品名称，运输备注，当前运输商名称，下游运输商名称信息，系统后台会据此创建新的交易区块。



同上，处于教学考虑，页面下方会展示本次创建的区块信息：

2.3.产品交易页

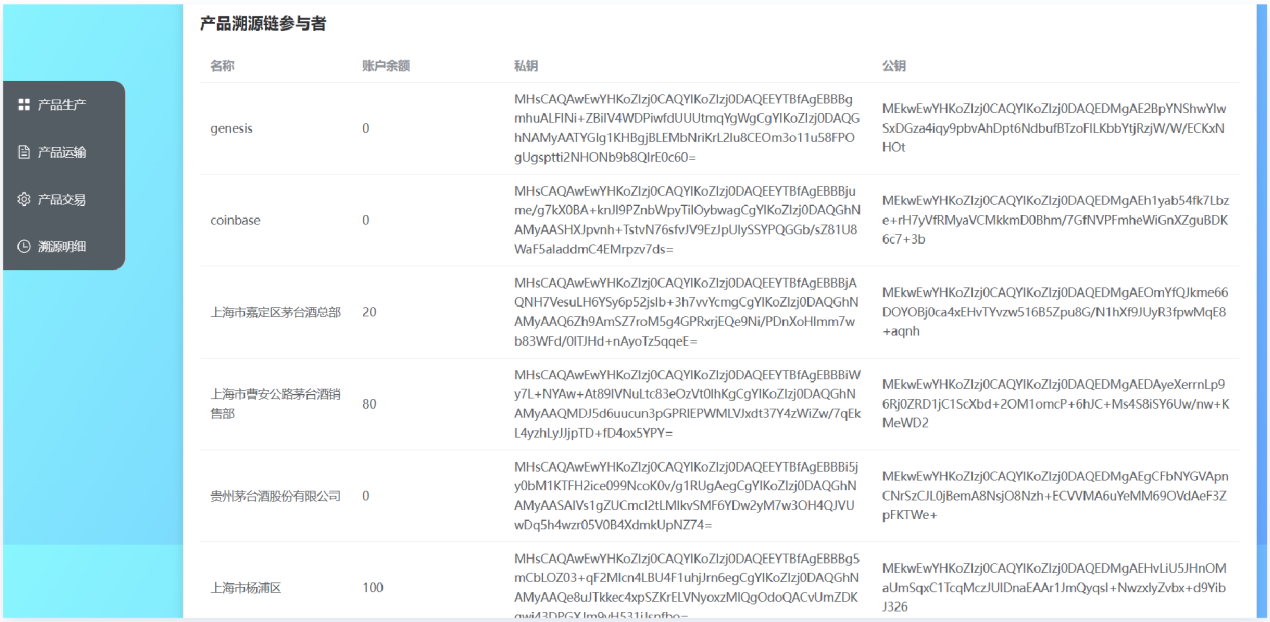
在产品交易页面中，用户能够输入商品名称，交易信息，卖家和买家，以及交易余额，点击创建交易后系统后台会自动根据 `UTXO` 规则进行交易。页面右侧提供余额查询入口，用户能够通过输入名称来进行余额查询。



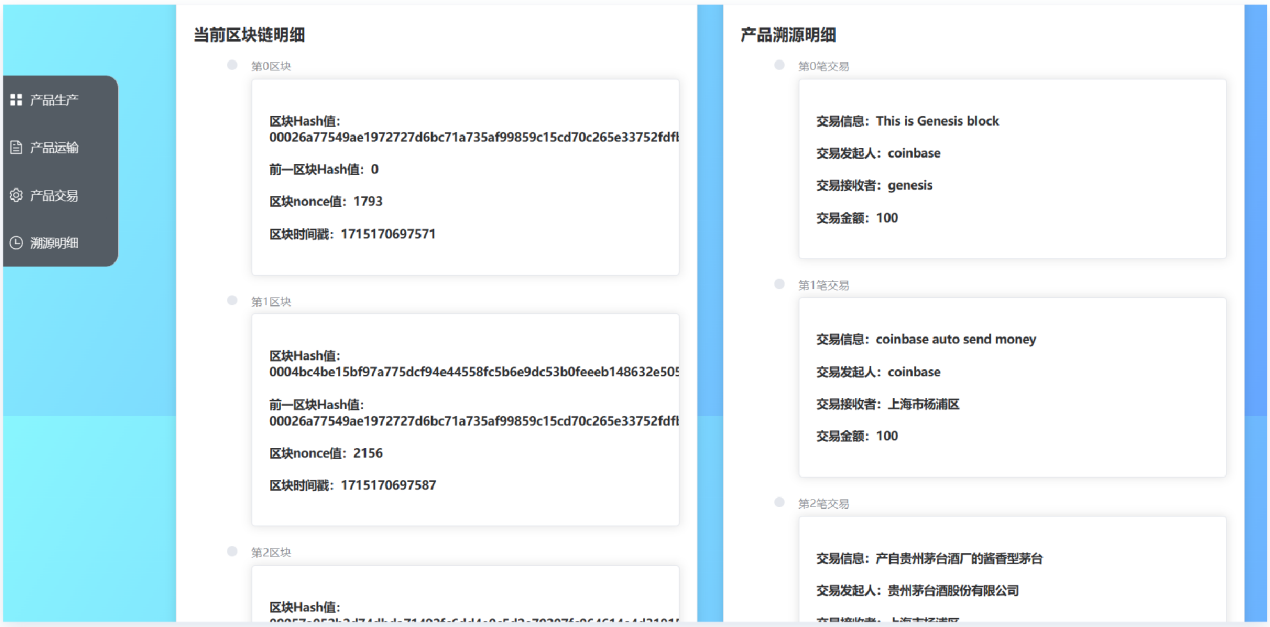
同上，为了明确展示新区块内容和交易信息，在页面下方进行可视化处理：

2.4.产品溯源页

产品溯源页首先展示本次产品链中的所有参与者，以及其余额。系统为了更加明确体现参与者标识数据，也会展示公私钥数据，但实际场景不可提供私钥数据。



在页面下方，会分为两列，分别展示区块链链信息和产品溯源信息。左侧区块链链信息从创世节点开始，到最后一个区块结束。而右侧产品溯源明细则是展示相对实际的溯源应用信息，展示了包括生产，运输和交易在内的多种信息。系统出于节省资源考虑，将虚拟币获取简化为创建参与者由 `coinbase` 自动分配100虚拟币，因此交易链中也会有 `coinbase` 的出现。



不难看到，最后由 `coinbase` 向上海市曹安公路茅台酒销售部提供100币，供其购买来自上海市嘉定区茅台酒总部的价值20币的茅台酒。

3.程序代码

由于区块链实现的具体内容上文已经进行详细介绍，故而此处重点展示为了实现系统而必要的 `SpringBoot` 框架下的区块链调用和接口程序。

3.1. Service 层代码

3.1.1.创世区块创建

```
public Block createGenesis(){
        Security.addProvider(new
org.bouncycastle.jce.provider.BouncyCastleProvider());
        Participants participantsA = new Participants("genesis");
        Participants coinbase = new Participants("coinbase");
        participantsHashMap.put("coinbase", coinbase);
        participantsHashMap.put("genesis", participantsA);
        Transaction genesisTransaction = new Transaction(coinbase.publicKey,
participantsA.publicKey,100f, new ArrayList<>(), "This is Genesis block",
"coinbase", "genesis");
        genesisTransaction.generateSignature(coinbase.privateKey);
        genesisTransaction.transactionId = "genesis";
        Block genesis = new Block("0");
        genesis.addTransaction(genesisTransaction);
        allTransactions.add(genesisTransaction);
        blockChain.addBlock(genesis);
        return genesis;
    }
```

3.1.2.添加区块/交易信息

```
public Block addBlock(String info, String fromName, String toName, float
value){
        Security.addProvider(new
org.bouncycastle.jce.provider.BouncyCastleProvider());
        Participants participantA = participantsHashMap.getOrDefault(fromName,
new Participants(fromName));
        Participants participantB = participantsHashMap.get(toName);
        if (participantB == null) {
            // 为了模拟交易场景，新建账户，系统自动发100币
            participantB = new Participants(toName);
            Participants coinbase = new Participants("coinbase");
            Transaction genesisTransaction = new
Transaction(coinbase.publicKey, participantB.publicKey,100f, new ArrayList<>
(), "coinbase auto send money", "coinbase", toName);
            genesisTransaction.generateSignature(coinbase.privateKey);
            genesisTransaction.transactionId = "coinbase auto send money";
```

```
            genesisTransaction.outputCases.add(new
OutputCase(genesisTransaction.receiver, genesisTransaction.value,
genesisTransaction.transactionId));
            BlockChain.UTXOs.put(genesisTransaction.outputCases.get(0).id,
genesisTransaction.outputCases.get(0)); //its important to store our first
transaction in the UTXOs list.
            Block newB = new Block(blockChain.getLastBlockHash());
            newB.addTransaction(genesisTransaction);
            allTransactions.add(genesisTransaction);
            blockChain.addBlock(newB);
            System.out.println(participantB.getBalance());
        }
        Block newBlock = new Block(blockChain.getLastBlockHash());
        System.out.println(value);
        if(value==0f){
            Transaction sendFunds =
participantB.sendFunds(participantA.publicKey, value, info, fromName, toName);
            newBlock.addTransaction(sendFunds);
            allTransactions.add(sendFunds);
        }else{
            Transaction sendFunds =
participantB.sendFunds(participantA.publicKey, value, info, toName, fromName);
            newBlock.addTransaction(sendFunds);
            allTransactions.add(sendFunds);
        }
        participantsHashMap.put(fromName, participantA);
        participantsHashMap.put(toName, participantB);
        blockChain.addBlock(newBlock);
        return newBlock;
    }
```

3.1.3.检查区块链合法性

```
public boolean checkChainValid(){
        return blockChain.isChainValid();
    }
```

3.1.4.获取所有参与者

```
public List<Map<String, Object>> queryParticipant(){
        Set<String> keys = participantsHashMap.keySet();
        List<Map<String, Object>> participantList = new ArrayList<>();
        // 遍历并输出所有键
```

```
        for (String key : keys) {
            Map<String, Object> participant = new HashMap<>();
            participant.put("name", key); // Add participant name
            participant.put("balance",
participantsHashMap.get(key).getBalance()); // Add participant balance
            participant.put("privateKey",
participantsHashMap.get(key).privateKey);
            participant.put("publicKey",
participantsHashMap.get(key).publicKey);
            participantList.add(participant); // Add participant object to the
list
        }
        return participantList;
    }
```

3.1.5.获取区块链

```
public ArrayList<Block> getBlockChain(){
        return BlockChain.blockchain;
    }
```

3.1.6.获取交易信息

```
public List<Transaction> queryTransactions(){
        return allTransactions;
    }
```

3.1.7.添加产品

```
public List<Map<String, Object>> addProducts(String name) {
        Map<String, Object> product = new HashMap<>();
        product.put("name", name);
        product.put("date", new Date());
        products.add(product);
        return products;
    }
```