4

Auditable Accounting

4.1 Objectives

The auditable accounting component is responsible of registering auditable logs. As such, this component is one of the main tools to enhance the trust in the ecosystem of data marketplaces. Our solution must enforce the data sharing agreement terms, agreed upon all involved parties, by recording them in an auditable, transparent, and immutable way. Smart contracts are the key part of the proposed solution for auditable accounting. Figure 4.1 shows that the auditable accounting component is an abstraction layer to access the smart contracts and to allow the integration with the rest of the platform. The auditable accounting component is a service that includes an API to automate the process of logging and auditing interactions between components and record the registries in the blockchain.

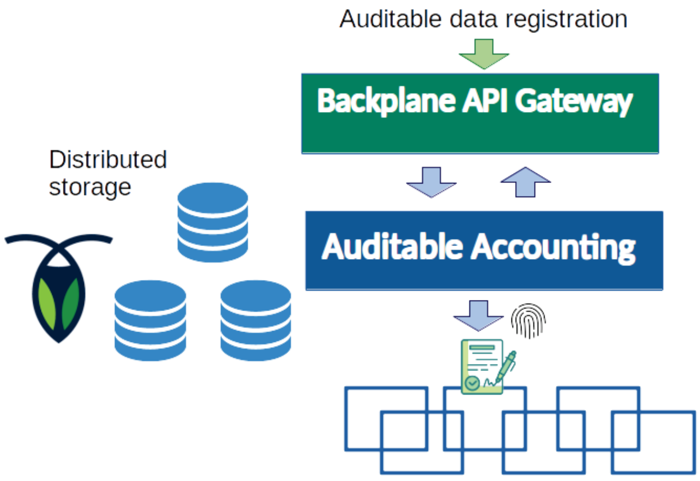


Figure 4.1 Auditable accounting architecture.

The auditable accounting development has been made publicly available in the i3-MARKET GitHub and Gitlab repositories (e.g., https://github.com/i3-Market-V3-Public-Repository/SP3-SCGBSSW-AA-Auditable Accounting). The Table 4.1 summarises the technical contributions used to design and implement the i3-MARKET Auditable accounting component.

4.2 Technical Requirements

Table 4.1 Main technical contributions.

Name	Description	Labels
Auditable Log	i3-MARKET needs to be able to log data and events in blockchain. It is a key component for accounting, billing, and conflict resolution. It is also important to control access to sensitive information and to detect potential data breaches. A public distributed ledger will be used to store non-repudiable and reliable proofs of the required actions. **Children:** 1. Auditable accounting: marketplace billing 2. Auditable accounting: conflict resolution 3. Auditable accounting: providing sensitive data 4. Providing sensitive data 5. Conflict resolution 6. Marketplace billing 7. Consumer billing 8. Provider billing **Parents:** 1. REQ-B-005 – i3-MARKET will ensure *trust* 2. REQ-B-008 – i3-MARKET will provide a payment solution 3. Semantic description of the SLS and the subscription	**V1** **Epic** **Data marketplace** **Data consumer** **Data provider** **Data owner**

4.3 Solution Design/Blocks

The solution must be scalable and cost-efficient. In this regard, transaction costs can be a considerable problem if, as it is expected, the number of auditable registries that need to be stored in the blockchain is high. To overcome this problem, it is a requirement to implement a transaction optimizer to efficiently register substantial amounts of data in the blockchain without incurring in excessive costs. To achieve this, we first store registries in an internal database of the component and then aggregate the registries with a Merkle tree to minimize the number of blockchain transactions and provide the appropriate data for proving each individual registry.

The smart contract managed by the auditable accounting component is used to store the necessary evidence of the aggregated registries from the DSA. Once the registration process is complete, the auditable accounting component will save a copy of all the information needed to verify that the registration was successful in the blockchain. This information can be consulted and obtained later by the marketplace users. The auditable accounting component provides the functionality to trace registries and obtain "certificates" of them that can be publicly or privately used to prove that a certain registry was performed. Users must be able to download these certificates and validate the registry without further interaction with the auditable accounting component having a proof that can be universally validated without the intervention of any other entity or software component. The certificate of a DSA will provide: the blockchain that has been used to create the auditable data registration, the address of the smart contract used, and the "proof of registry" of the associated data.

The auditable accounting component is a service that includes an API to automate the process of logging and auditing interactions between components and record the registries in the blockchain. As shown in Figure 4.1, in general, the API of the AA module is accessed through the Backplane API gateway. Additionally, the auditable accounting component can be accessed directly from any internal component of the platform.

On the other hand, to allow external parties to check that logs have been properly registered in the blockchain, interested parties need to obtain certain data from the distributed ledger as well as some off-chain data provided by the auditable accounting module via an API. This off-chain data are essentially Merkle proofs for each individual record. It is important that the off-chain data is provided with high availability. For this reason, the auditable accounting module uses the distributed storage component. In this way, high availability and data replication is provided to the relevant off-chain data required to store the registries and verify auditable logs.

Database model:

The database model proposed for this component is based on two SQL tables. The first one is the related one with the blockchain. It contains the transactions prepared or sent to the blockchain. Figure 4.2 shows the deployed columns as follows:

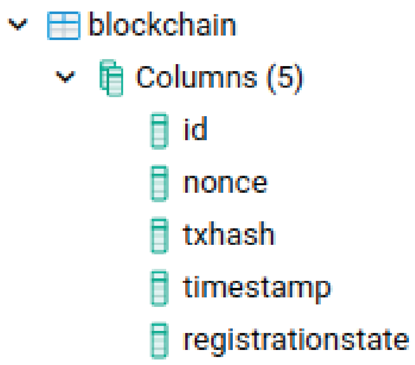
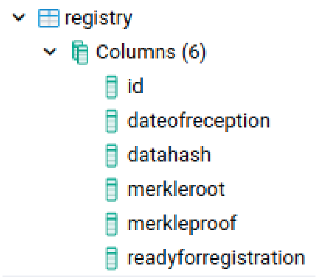


Figure 4.2 Auditable accounting library distribution.

- **Id:** Primary key to link with the other table.
- **Nonce:** Nonce from the account to build the transaction.
- **Txhash:** Hash of the transaction.
- **Timestamp:** Exact date of the creation of the transaction.
- **Registrationstate:** Status of the transaction.
    - **Unregistered:** Transaction not created.
    - **Pending:** Transaction created but not sent to the blockchain.
    - **Mined:** Transaction sent with less than 12 block confirmations.
    - **Confirmed:** Transaction sent with more that 12 block confirmations.

On the other hand, the registry table is responsible to store the proofs of the data hashes that want to be validated against the blockchain.

It contains the following columns:


registry
Columns (6)
id
dateofreception
datahash
merkleroot
merkleproof
readyforregistration

- **Id:** Primary key to link with the other table.
- **Dateofreception:** Date when the data is received.
- **Datahash:** Cryptographic hash function of the data. It is one of the leaves of the Merkle tree.
- **Merkleroot:** Root of the Merkle tree.
- **Merkleproof:** Concatenated hashes that allow to validate the datahash to the root of the tree.
- **Readyforregistration:** Boolean to indicate if the tree is built and ready to be deployed in the blockchain.

Smart contract:

The smart contract deployed for this component just stores the root of the Merkle tree that summarizes all the data hashes stored in the database. It only allows to modify that value by the owner of the smart contract, which shares the same account with the auditable accounting. Also, it includes the capability to subscribe to an event that notifies you about a new root released. The Solidity code is the following:

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.22 <0.8.0;

contract AuditableAccounting {
  uint256 public currentRoot;
  address public owner;


  event newRegistry(uint256 prevRootHash, uint256 currentRootHash);

  modifier onlyOwner(){ require(msg.sender == owner, "sender must be the
contract owner"); _; }

  constructor() { owner = msg.sender; }

  function setNewRegistry(uint256 _newRoot) public onlyOwner {
    emit newRegistry(currentRoot, _newRoot);
    currentRoot = _newRoot;
  }
}
```

4.4 Diagrams

The workflow to register auditable data is shown in Figure 4.3.

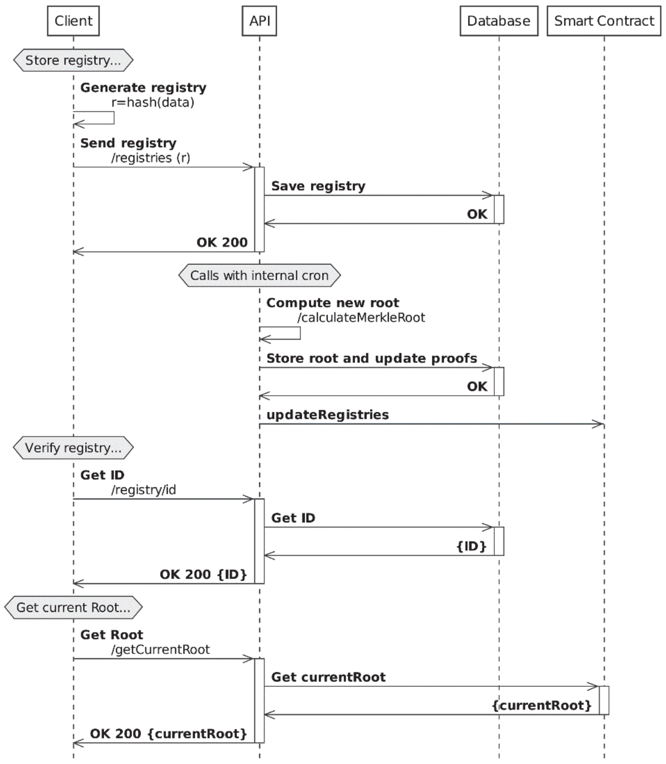
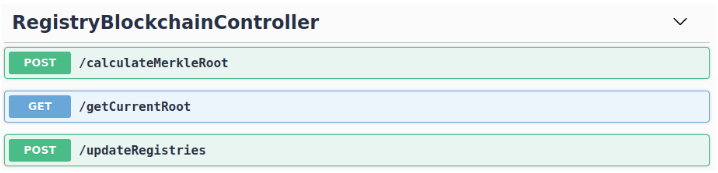


Figure 4.3 Auditable accounting flow.

The hash of the data to be registered is sent to the API using the end-point/registries. Each hash to be registered is stored by the auditable accounting module in distributed storage. Then, the endpoint */calculateMerkleRoot* has to be called. When called, this endpoint creates the structure that is going to be registered in the blockchain. In more detail, this structure is a Merkle hash tree. The controller of the endpoint computes the Merkle root with all the pending registries, computes an individual proof for each registry, and stores these proofs in the distributed storage. Additionally, a transaction to be sent to the blockchain is created and stored in the blockchain SQL table in the distributed storage. Next, the endpoint */updateRegistries* can be called to store the Merkle root of the registries in the blockchain via the smart contract. We would like to stress that the endpoints */calculateMerkleRoot* and */updateRegistries* can be called with a "cron job" or similar to schedule registrations in the blockchain at the desired frequency. Finally, if a party wants to verify a certain registry, it can call the endpoint */registry/:id* to obtain the corresponding Merkle proof, compute the Merkle root from this proof, and compare it to the root registered in the smart contract. If both are the same, this means that the registry is valid.

4.5 Interfaces

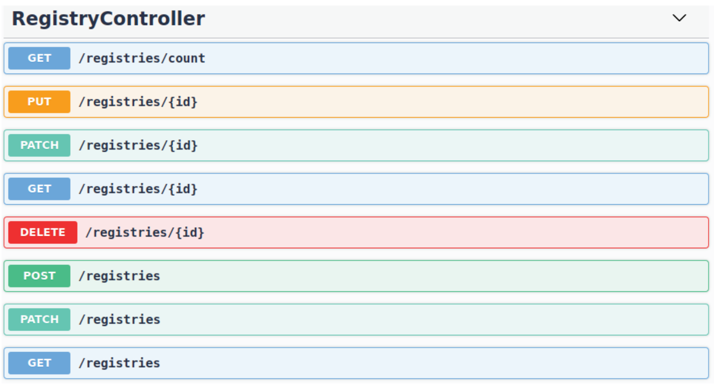
The component is built from a Loopback 4 framework, which facilitates the management of the smart contract and the database generating an API that allows to integrate the procedures with the Backplane. But, as a high-level definition, the endpoints are divided into two controllers.
Firstly, the RegistryBlockchain controller manages the smart contract interactions and has the following endpoints:


RegistryBlockchainController

```
POST   /calculateMerkleRoot
GET    /getCurrentRoot
POST   /updateRegistries
```

- **/calculateMerkleRoot:** Gets the pending registries from distributed storage that are not included in the current root and computes the new one.
- **/getCurrentRoot:** Gets the current root from the smart contract.
- **/updateRegistries:** Updates the status of the stored transactions and computes a new transaction.

On the other hand, there is the registry controller, which is responsible to manage the data hashes that are included in the auditable accounting system.


RegistryController

```
GET     /registries/count
PUT     /registries/{id}
PATCH   /registries/{id}
GET     /registries/{id}
DELETE  /registries/{id}
POST    /registries
PATCH   /registries
GET     /registries
```

- GET **/registries/count:** Returns the number of stored registries.
- PUT **/registries/{id}:** Forces the creation of a specific registry.
- PATCH **/registries/{id}:** Updates a specific registry.
- GET **/registries/{id}:** Returns the value of a specific registry.
- DELETE **/registries/{id}:** Removes a specific registry.
- POST **/registries:** Generates a new registry.
- GET **/registries:** Returns the value of the registries.

4.6 Background Technologies

- **Solidity:**

Solidity is an object-oriented, high-level language for implementing smart contracts. Smart contracts are programs that govern the behaviour of accounts within the Ethereum state. It is a curly-bracket language. It is influenced by C++, Python, and JavaScript and is designed to target the Ethereum virtual machine (EVM).

Solidity is used to develop the smart contract deployed on the blockchain, which is responsible to store the root of the Merkle hash tree.

- **PostgreSQL:**

PostgreSQL is a powerful, open-source object-relational database system with over 30 years of active development that has earned it a strong reputation for reliability, feature robustness, and performance.

PostgreSQL is used to store the registries and the Merkle proofs of each registry.

- **Loopback** 4**:**

LoopBack 4 is an award-winning, highly extensible, open-source Node.js and TypeScript framework based on Express. It enables you to quickly create APIs and microservices composed from backend systems such as databases and SOAP or REST services. Also, it allows to manage custom data sources like a smart contract.

Loopback is used to generate the API that manages the registration of the data, the computation of the Merkle hash trees, and the smart contract executions.