Activity 3: Creating a Supply Chain Management platform.

Solidity basics

Events.

Events are used in Solidity to store logs. Each event has a name and arguments. Arguments are stored along with contract address in a special data structure, a log Patricia Merkle data structure. Up to three parameters can be indexed. These arguments are “topics”. A topic can store a single word (32 bytes) and allow searching for events. Web3 applications usually subscribe to events topics. Events can be also filtered by the address of the contract that emitted the events. Storing data in logs cost less gas than storing data in storage.

Reverts and errors.

Solidity has the following syntax to treat failure situations.

- **revert** statement: revert statement aborts and reverts all changes performed to the state before the transaction. Revert can indicate the name of the error encountered with additional parameters.
    - revert(string memory reason): abort execution and revert state changes, providing an explanatory string
    - revert CustomError: abort execution and revert state changes, throwing CustomError.

- **errors:** can be used in revert statements. They can be defined inside and outside of contracts. The error creates data that is then passed to the caller with the revert operation or catch it in a try/catch statement.
- **require:** The require function either creates an error without any data or an error of type Error(string). It should be used to ensure valid conditions that cannot be detected until execution time. This includes conditions on inputs or return values from calls to external contracts.
- **assert:** The assert function creates an error of type Panic(uint256). Assert should only be used to test for internal errors, and to check invariants.

Modifiers.

Modifiers can be used to change the behavior of functions in a declarative way. For example, you can use a modifier to automatically check a condition prior to executing the function. Multiple modifiers are applied to a function by specifying them in a whitespace-separated list and are evaluated in the order presented. Arbitrary expressions are allowed for modifier arguments. The definition of a modifier includes:

- **“_”**: indicates where the code of the function is to be executed.

- **require** statement: indicates the condition to be tested.

Supply Chain Management

Supply chain management blockchain solutions oversee participant interactions, initiate events recorded for participants' visibility and transaction tracking, securely facilitate payment dispersal, and actively govern the interaction dynamics between sellers and consumers by monitoring transaction status. Furthermore, smart contracts may monitor products state and secure transport by ensuring predefined parameters, for example temperature control for sensitive shipments.

EXERCISES

1. Work on file VotingStart.sol. Test on RemixIDE https://remix.ethereum.org/

- Add events *UpdateProposalState* with topics projectName, teamName and *NewVote* with topic voter.
- Log events (emit) *UpdateProposalState* and *NewVote* when an update a proposal state occurs or when a new vote is registered.
- Add modifiers *votingActive*, *votingEnded* (or add a single *votingState* modifier with an argument of type bool), and *onlyChairPerson.* Use modifiers for functions: *setProposalState*, *registerVoter*, *vote* and *winningProposal* to ensure that voting is in the required timeframe and only the chairperson may change the state of a proposal or find the *winningProposal* after the vote has ended.
- Add modifier *validVote*(uint[] memory votes, uint len) to ensure that each vote picks the required number (*len*) of distinct proposals.

2. **Supply chain management**: Create a contract *SupplyChain* to be integrated as part of a Supply Chain Management System architecture. The main actors are *sellers* and *consumers* identified by their Ethereum addresses. The products are to be shipped in a secure environment, designed for freight with strict temperature requirements. A seller creates a package. Each package is shipped in a container (IoT device). A secret code is generated for each package, the code is used by the consumer the unlock the container. The container monitors the temperature. If the critical temperature is detected the shipment is aborted. Also, if the shipment takes longer than a specific number of days the shipment is aborted. After the creation of the package the consumer will store in the contact the sum requested by the seller. After the consumer opens the container, the sum will be available to be sent to the seller.
    a. Define the ***enumeration*** needed to indicate package status.
    b. Define ***events***: CreatePackage, ConsumerDeposit, StartShipment, CancelShipment, OpenContainer, UnlockPaymentToSeller. Write the functions that will trigger these events.
    c. Add ***modifiers*** to test that some specific operations are performed only by the rightful actors. For instance, check that the container is opened only by the consumer.
    d. Ensure ***reverts or errors*** when needed. For instance, package opening should revert if the consumer does not provide the correct security code.

Bibliography

- https://docs.soliditylang.org/_/downloads/en/latest/pdf/
- https://docs.soliditylang.org/en/v0.8.0/contracts.html#modifiers
- Abdallah, Salam, and Nishara Nizamuddin. "Blockchain-based solution for pharma supply chain industry." *Computers & Industrial Engineering* 177 (2023): 108997.