**pragma solidity ^0.6.0;** This is a Solidity pragma statement that specifies the version of the Solidity compiler that should be used to compile the code. In this case, the code is compatible with version 0.6.0 and above.

**contract SupplyChain {** This line starts the definition of a new Solidity smart contract named "SupplyChain". The code for the smart contract will be contained within the curly braces that follow.

csharpCopy code

event Added(uint256 index);

This is an event declaration that defines a new event named "Added". Events are a way for smart contracts to communicate with the outside world, and this event will be emitted whenever a new product is added to the supply chain. The event takes a single parameter of type uint256, which represents the index of the new product in the **allProducts** mapping.

goCopy code

struct State{ string description; address person; }

This line defines a new struct named "State" that contains two fields: a string field named "description" and an address field named "person". This struct will be used to represent the current state of a product in the supply chain.

goCopy code

struct Product{ address creator; string productName; uint256 productId; string date; uint256 totalStates; mapping (uint256 => State) positions; }

This line defines another struct named "Product" that contains several fields: an address field named "creator" that represents the creator of the product, a string field named "productName" that represents the name of the product, a uint256 field named "productId" that represents the ID of the product, a string field named "date" that represents the manufacture date of the product, a uint256 field named "totalStates" that represents the total number of states that the product has gone through, and a mapping field named "positions" that maps a uint256 key to a State value. This struct will be used to represent a product in the supply chain.

scssCopy code

mapping(uint => Product) allProducts;

This line declares a new mapping named "allProducts" that maps a uint key to a Product value. This mapping will be used to store all of the products in the supply chain.

Copy code

uint256 items=0;

This line declares a new uint256 variable named "items" and initializes it to 0. This variable will be used to keep track of the total number of products in the supply chain.

scssCopy code

function concat(string memory \_a, string memory \_b) public returns (string memory){ bytes memory bytes\_a = bytes(\_a); bytes memory bytes\_b = bytes(\_b); string memory length\_ab = new string(bytes\_a.length + bytes\_b.length); bytes memory bytes\_c = bytes(length\_ab); uint k = 0; for (uint i = 0; i < bytes\_a.length; i++) bytes\_c[k++] = bytes\_a[i]; for (uint i = 0; i < bytes\_b.length; i++) bytes\_c[k++] = bytes\_b[i]; return string(bytes\_c); }

This is a helper function that concatenates two strings together and returns the result. It takes two string parameters (\_a and \_b) and returns a new concatenated string.

phpCopy code

function newItem(string memory \_text, string memory \_date) public returns (bool) { Product memory newItem = Product({creator: msg.sender, totalStates: 0,productName: \_text, productId: items, date: \_date}); allProducts[items]=newItem; items = items+1; emit Added(items-1

go on

The **function newItem(string memory \_text, string memory \_date) public returns (bool)** starts the definition of a new function named **newItem**. It takes two input parameters of type **string** and returns a **bool** value.

solidityCopy code

Product memory newItem = Product({creator: msg.sender, totalStates: 0,productName: \_text, productId: items, date: \_date});

This line creates a new **Product** struct named **newItem** and assigns values to its attributes. **msg.sender** is the Ethereum address of the caller of this function, which will be the creator of the product. **totalStates** is initialized to 0 as the product has not gone through any state changes yet. **\_text** and **\_date** are the two parameters passed to the function that define the product's name and creation date respectively. **items** is a global counter that keeps track of the total number of products created so far, and is used to assign a unique **productId** to each new product.

solidityCopy code

allProducts[items]=newItem;

This line adds the newly created **Product** to a **mapping** named **allProducts**. The key is the **productId** of the **Product**, and the value is the entire **Product** struct.

solidityCopy code

items = items+1;

This line increments the **items** counter by 1, to ensure that the next new product has a unique **productId**.

solidityCopy code

emit Added(items-1);

This line emits an **event** named **Added** with the parameter **items-1**. This event notifies any subscribed clients of the newly added product's **productId**.

solidityCopy code

return true;

This line returns a **bool** value of **true** to indicate that the function was executed successfully.

go o

solidityCopy code

function addState(uint \_productId, string memory info) public returns (string memory) { require(\_productId<=items); State memory newState = State({person: msg.sender, description: info}); allProducts[\_productId].positions[ allProducts[\_productId].totalStates ]=newState; allProducts[\_productId].totalStates = allProducts[\_productId].totalStates +1; return info; }

This code defines a function called **addState**. It takes two arguments: **\_productId**, an unsigned integer representing the ID of the product being updated; and **info**, a string describing the new state of the product.

solidityCopy code

require(\_productId<=items);

This line verifies that **\_productId** is less than or equal to **items**. This ensures that the specified product ID is within the range of products that have been created, to prevent any errors from occurring.

solidityCopy code

State memory newState = State({person: msg.sender, description: info});

This line creates a new **State** struct named **newState** and assigns values to its attributes. **msg.sender** is the Ethereum address of the caller of this function, which represents the current owner of the product. **info** is the string passed as an argument to this function, which describes the new state of the product.

solidityCopy code

allProducts[\_productId].positions[ allProducts[\_productId].totalStates ]=newState;

This line updates the **positions** mapping of the specified product with the new **State** struct. The key is the current value of **totalStates**, which represents the number of previous states the product has gone through, and the value is the **newState** struct created in the previous line.

solidityCopy code

allProducts[\_productId].totalStates = allProducts[\_productId].totalStates +1;

This line increments the **totalStates** attribute of the specified product to reflect the fact that a new state has been added.

solidityCopy code

return info;

This line returns the **info** string that was passed as an argument to this function. This is not strictly necessary, as the caller of the function already knows the new state of the product, but it can be helpful for debugging purposes.

go on

solidityCopy code

function searchProduct(uint \_productId) public returns (string memory) { require(\_productId<=items); string memory output="Product Name: "; output=concat(output, allProducts[\_productId].productName); output=concat(output, "<br>Manufacture Date: "); output=concat(output, allProducts[\_productId].date); for (uint256 j=0; j<allProducts[\_productId].totalStates; j++){ output=concat(output, allProducts[\_productId].positions[j].description); } return output; }

This code defines a function called **searchProduct**, which takes a single argument **\_productId** of type **uint** (unsigned integer). The function returns a string that contains information about the specified product, including its name, manufacture date, and all the states it has gone through.

solidityCopy code

require(\_productId<=items);

This line verifies that the specified **\_productId** is less than or equal to **items**, which represents the number of products that have been created so far. This ensures that the product ID is within the range of valid IDs and prevents any errors from occurring.

solidityCopy code

string memory output="Product Name: "; output=concat(output, allProducts[\_productId].productName); output=concat(output, "<br>Manufacture Date: "); output=concat(output, allProducts[\_productId].date);

These lines create a new string variable called **output**, which will be used to store the information about the specified product. The initial value of **output** is the string **"Product Name: "**. The **concat** function is then used to concatenate this string with the name of the product, as well as the manufacture date, which is separated by a **<br>** HTML tag.

solidityCopy code

for (uint256 j=0; j<allProducts[\_productId].totalStates; j++){ output=concat(output, allProducts[\_productId].positions[j].description); }

This **for** loop iterates over all the states that the specified product has gone through, starting from the first state and continuing until the last state. For each state, the description of that state is added to the **output** string using the **concat** function. This ensures that the information about all the states is included in the final output.

solidityCopy code

return output;

This line returns the final **output** string, which contains all the information about the specified product. This string can then be used by the caller of the function to display information about the product to users or to perform other operations on the product.