```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.26;

contract DrugSupplyChain {
    enum DrugStatus {
        Manufactured,
        Shipped,
        Received,
        Dispatched
    }

    struct Drug {
        uint256 id;
        string name;
        address manufacturer;
        uint256 expirationDate;
        DrugStatus status;
        address currentHolder;
        uint256 lastUpdated;
    }

    struct DrugStatusUpdate {
        uint256 drugId;
        DrugStatus status;
        address sender;
        address receiver;
        uint256 timestamp;
    }

    // Set the fee for each transaction (in wei)
    uint256 public constant GAS_FEE = 5045;
```

```
    address public owner;

    Drug[] public drugs;
    DrugStatusUpdate[] public statusUpdates;

    event DrugRegistered(uint256 indexed drugId, string name, address manufacturer, uint256 expirationDate);
    event StatusUpdated(uint256 indexed drugId, DrugStatus indexed status, address indexed sender, address receiver);

    modifier onlyOwner() {
        require(msg.sender == owner, "Only the owner can call this");
        _;
    }

    modifier hasPaidFee() {
        require(msg.value >= GAS_FEE, "Insufficient fee to perform operation");
        _;
    }

    constructor() {
        owner = msg.sender;
    }

    // Register a new drug
    function registerDrug(
        string memory _name,
        address _manufacturer,
        uint256 _expirationDate
    ) public payable hasPaidFee returns (uint256) {
```

```
        drugs.push(Drug(drugs.length, _name, _manufacturer, _expirationDate,
DrugStatus.Manufactured, _manufacturer, block.timestamp));
        emit DrugRegistered(drugs.length - 1, _name, _manufacturer, _expirationDate);
        return drugs.length - 1;
    }

    // Update the status of a drug
    function updateStatus(
        uint256 _drugId,
        DrugStatus _status,
        address _receiver
    ) public payable hasPaidFee {
        require(_drugId < drugs.length, "Invalid drug ID");
        drugs[_drugId].status = _status;
        drugs[_drugId].currentHolder = _receiver;
        drugs[_drugId].lastUpdated = block.timestamp;

        statusUpdates.push(DrugStatusUpdate(_drugId, _status, msg.sender, _receiver,
block.timestamp));
        emit StatusUpdated(_drugId, _status, msg.sender, _receiver);
    }

    // Function to retrieve drug information by ID
    function getDrugById(uint256 _id) public view returns (Drug memory) {
        return drugs[_id];
    }

    // Function to retrieve status updates for a specific drug
    function getStatusUpdatesByDrugId(uint256 _drugId) public view returns (DrugStatusUpdate[]
memory) {
        DrugStatusUpdate[] memory updates = new DrugStatusUpdate[](statusUpdates.length);
        uint256 count = 0;
```

```
        for (uint256 i = 0; i < statusUpdates.length; i++) {
            if (statusUpdates[i].drugId == _drugId) {
                updates[count] = statusUpdates[i];
                count++;
            }
        }
        return updates;
    }

    // Withdraw contract balance (if needed)
    function withdraw() public onlyOwner {
        payable(owner).transfer(address(this).balance);
    }

    // Fallback function to accept any ETH sent to the contract
    receive() external payable {}
}
```