

Vehicle History

LuckyDay Vehicle History Report

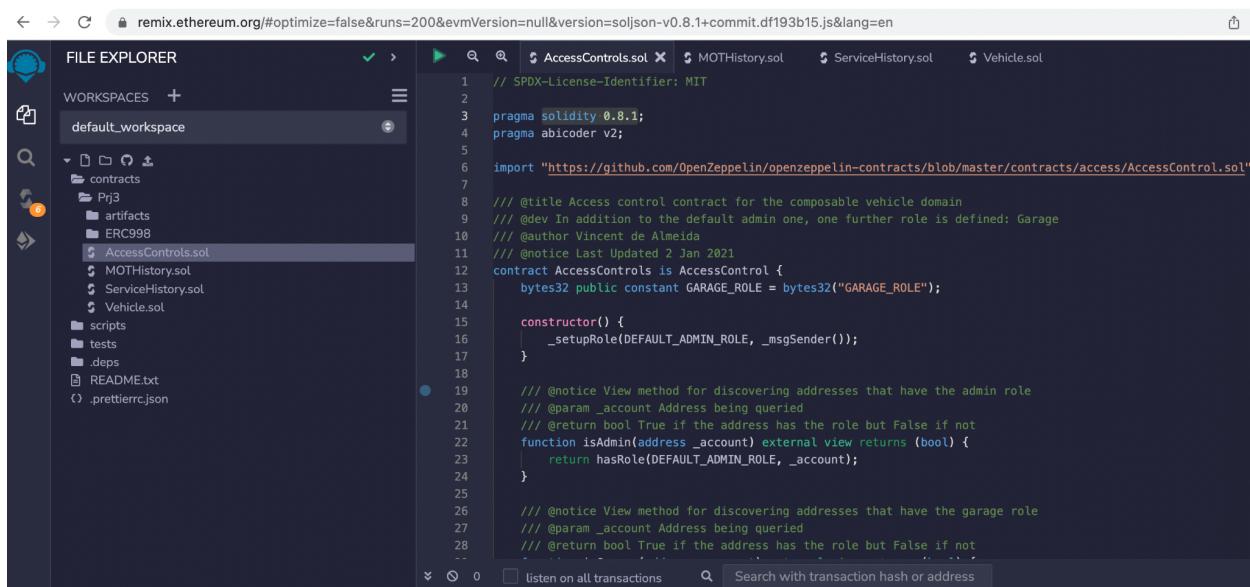
On Blockchain - Smart Contracts Setup

Introduction

LuckyDay Vehicle History Report is a part of the LuckyDay web app , for storing vehicle history records on the blockchain. The github repository for LuckyDay app is https://github.com/KausarHina/lucky_day_multipage. This document describes smart contracts setting up steps needed to access Vehicle History Report.

Compile Smart contracts

Clone github repository https://github.com/KausarHina/lucky_day_multipage in your local machine path. Then copy ERC998 , AccessControls.sol, MOTHistory.sol, ServiceHistory.sol and Vehicle.sol in Remix IDE from your cloned path and compile AccessControls.sol, MOTHistory.sol, ServiceHistory.sol and Vehicle.sol in the Remix.



The screenshot shows the Remix IDE interface. The left sidebar displays the file structure of the workspace, including contracts (AccessControls.sol, MOTHistory.sol, ServiceHistory.sol, Vehicle.sol), artifacts, and scripts. The right pane shows the Solidity code for the AccessControls contract. The code includes imports, license information, and a constructor function that sets up roles. It also contains functions for checking if an account has a specific role (isAdmin) and for discovering addresses with a given role (getRoleMembers).

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.1;
pragma abicoder v2;

import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/AccessControl.sol";

/// @title Access control contract for the composable vehicle domain
/// @dev In addition to the default admin one, one further role is defined: Garage
/// @author Vincent de Almeida
/// @notice Last Updated 2 Jan 2021
contract AccessControls is AccessControl {
    bytes32 public constant GARAGE_ROLE = bytes32("GARAGE_ROLE");

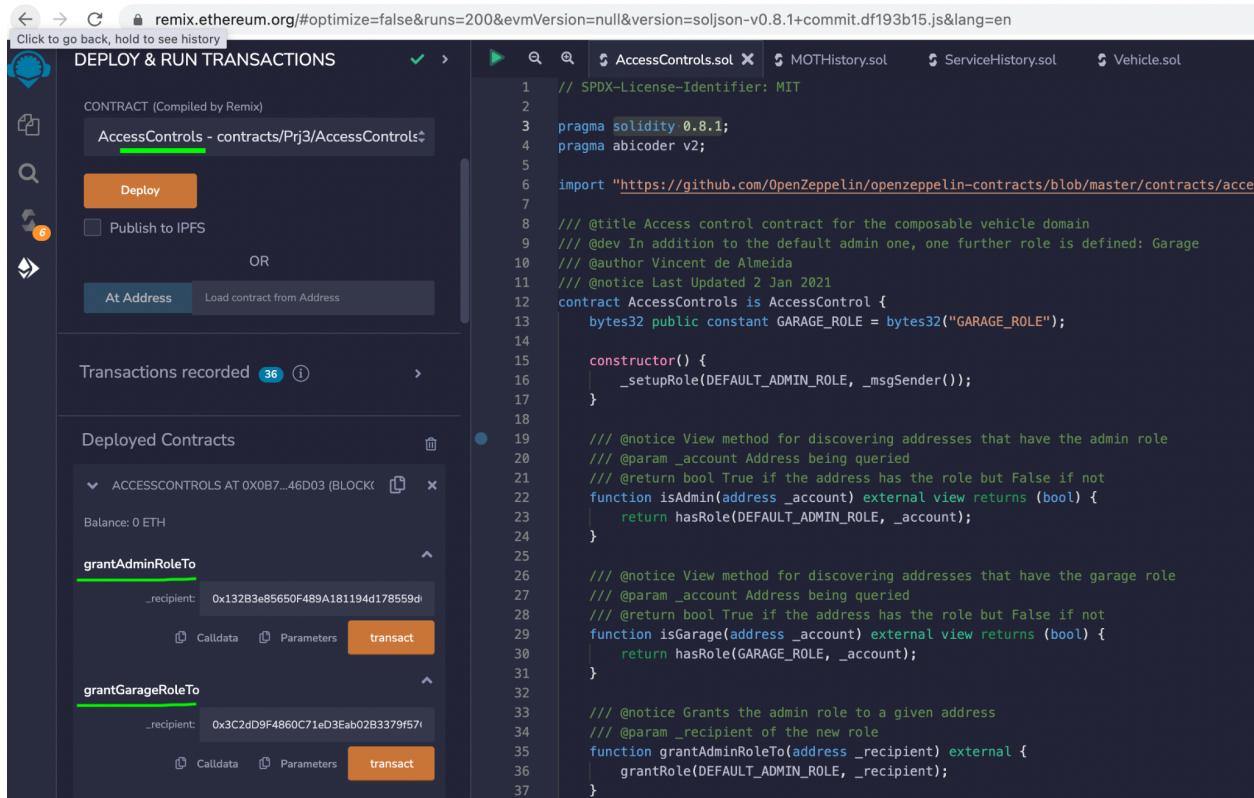
    constructor() {
        _setupRole(DEFAULT_ADMIN_ROLE, _msgSender());
    }

    /// @notice View method for discovering addresses that have the admin role
    /// @param _account Address being queried
    /// @return bool True if the address has the role but False if not
    function isAdmin(address _account) external view returns (bool) {
        return hasRole(DEFAULT_ADMIN_ROLE, _account);
    }

    /// @notice View method for discovering addresses that have the garage role
    /// @param _account Address being queried
    /// @return bool True if the address has the role but False if not
}
```

Compile and Deploy Smart Contract AccessControls

Compile smart contract AccessControls in Remix and deploy . Then grant admin role and garage role to recipient accounts.



The screenshot shows the Remix IDE interface. On the left, there's a sidebar with various icons. The main area has tabs for "DEPLOY & RUN TRANSACTIONS" and "AccessControls - contracts/Prj3/AccessControls". Under "DEPLOY & RUN TRANSACTIONS", there are buttons for "Deploy", "Publish to IPFS", and "At Address". Below these are sections for "Transactions recorded" (with 36 items) and "Deployed Contracts" (listing "ACCESSCONTROLS AT 0XB7...46D03 (BLOCK)". Under "Deployed Contracts", there are two functions: "grantAdminRoleTo" and "grantGarageRoleTo", each with a recipient address input field and a "transact" button. On the right side of the interface, the "AccessControls.sol" file is displayed with its Solidity code:

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.1;
pragma abicoder v2;

import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/acce
...
// @title Access control contract for the composable vehicle domain
// @dev In addition to the default admin one, one further role is defined: Garage
// @author Vincent de Almeida
// @notice Last Updated 2 Jan 2021
contract AccessControls is AccessControl {
    bytes32 public constant GARAGE_ROLE = bytes32("GARAGE_ROLE");

    constructor() {
        _setupRole(DEFAULT_ADMIN_ROLE, _msgSender());
    }

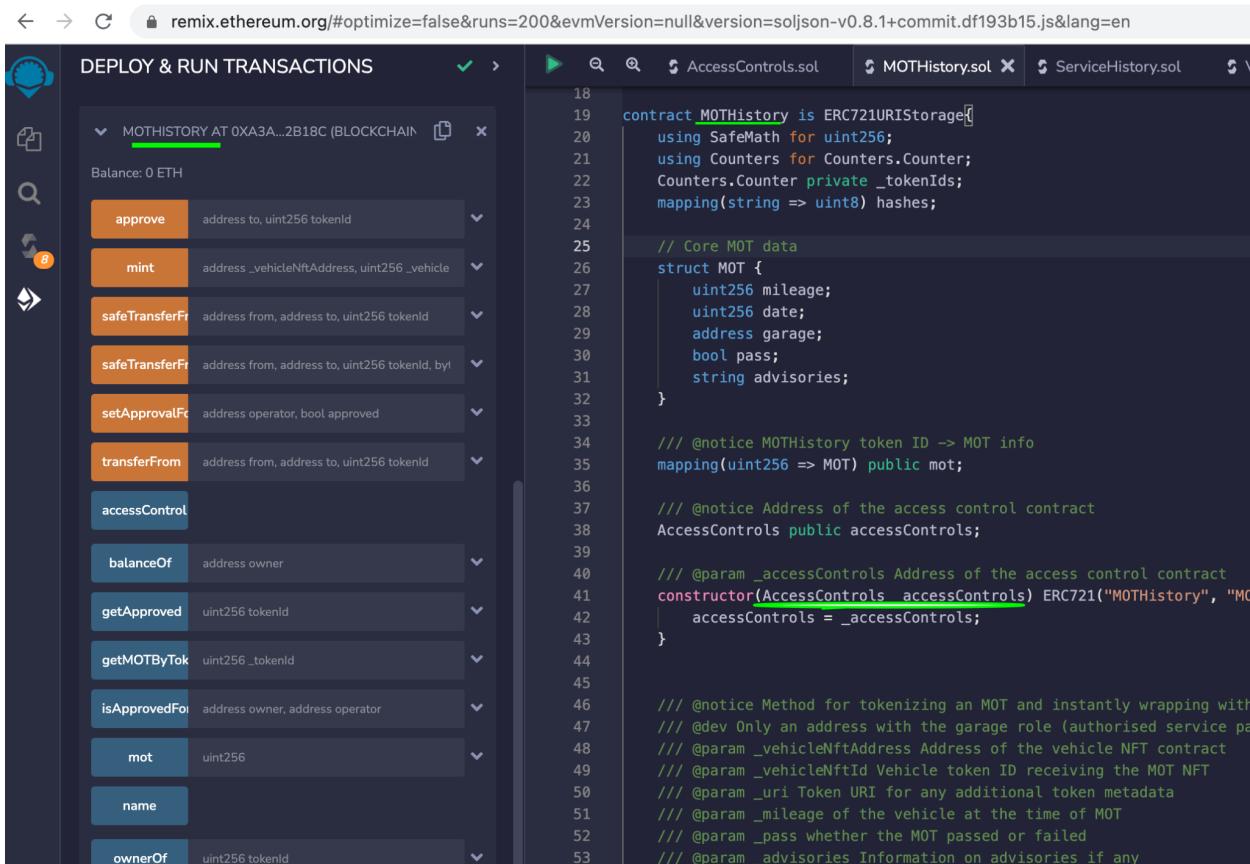
    // @notice View method for discovering addresses that have the admin role
    // @param _account Address being queried
    // @return bool True if the address has the role but False if not
    function isAdmin(address _account) external view returns (bool) {
        return hasRole(DEFAULT_ADMIN_ROLE, _account);
    }

    // @notice View method for discovering addresses that have the garage role
    // @param _account Address being queried
    // @return bool True if the address has the role but False if not
    function isGarage(address _account) external view returns (bool) {
        return hasRole(GARAGE_ROLE, _account);
    }

    // @notice Grants the admin role to a given address
    // @param _recipient of the new role
    function grantAdminRoleTo(address _recipient) external {
        grantRole(DEFAULT_ADMIN_ROLE, _recipient);
    }
}
```

Compile and Deploy Smart Contract MotHistory

Compile MotHistory smart contract in Remix and deploy it with the deployed address of AccessControls.



The screenshot shows the Remix IDE interface. On the left, there's a sidebar with various icons. The main area has tabs for "AccessControls.sol", "MOTHistory.sol" (which is currently selected), and "ServiceHistory.sol". The "MOTHistory.sol" tab shows the Solidity code for the MotHistory contract. The code defines a struct MOT with fields for mileage, date, garage, pass, and advisories. It also includes a mapping from token ID to MOT info and a constructor that initializes the access control contract. The "DEPLOY & RUN TRANSACTIONS" section on the right shows the deployed contract at address 0xA3A...2B18C, with a balance of 0 ETH. It lists several functions: approve, mint, safeTransferFrom, safeTransferFrom, setApprovalFor, transferFrom, accessControl, balanceOf, getApproved, getMOTByTok, isApprovedFor, mot, name, and ownerOf. Each function has its parameters listed below it.

```
contract MOTHistory is ERC721URIStorage{
    using SafeMath for uint256;
    using Counters for Counters.Counter;
    Counters.Counter private _tokenIds;
    mapping(string => uint8) hashes;

    // Core MOT data
    struct MOT {
        uint256 mileage;
        uint256 date;
        address garage;
        bool pass;
        string advisories;
    }

    /// @notice MOTHistory token ID -> MOT info
    mapping(uint256 => MOT) public mot;

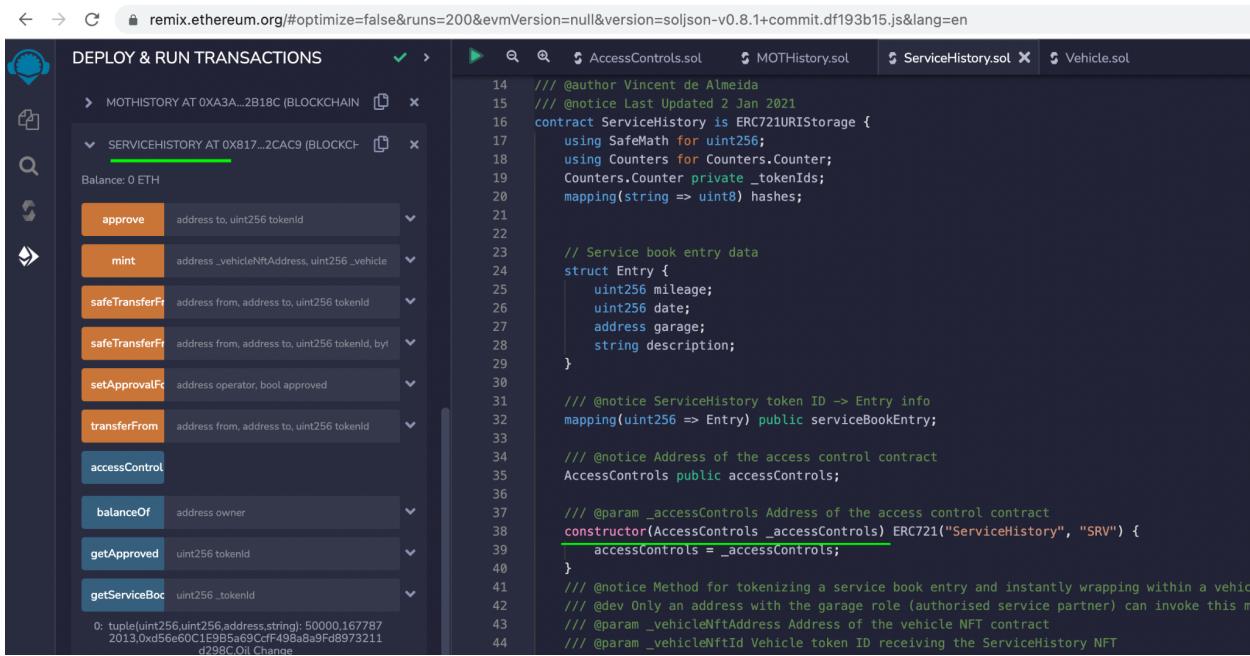
    /// @notice Address of the access control contract
    AccessControls public accessControls;

    /// @param _accessControls Address of the access control contract
    constructor(AccessControls _accessControls) ERC721("MOTHistory", "MO")
        accessControls = _accessControls;
    }

    /// @notice Method for tokenizing an MOT and instantly wrapping with
    /// @dev Only an address with the garage role (authorised service pa
    /// @param _vehicleNftAddress Address of the vehicle NFT contract
    /// @param _vehicleNftId Vehicle token ID receiving the MOT NFT
    /// @param _uri Token URI for any additional token metadata
    /// @param _mileage of the vehicle at the time of MOT
    /// @param _pass whether the MOT passed or failed
    /// @param _advisories Information on advisories if any
}
```

Compile and Deploy Smart Contract ServiceHistory

Compile ServiceHistory smart contract in Remix and deploy it with the deployed address of AccessControls.



The screenshot shows the Remix IDE interface. On the left, there's a sidebar with icons for deploying, running, and interacting with contracts. The main area is divided into two sections: 'DEPLOY & RUN TRANSACTIONS' on the left and the 'ServiceHistory.sol' code editor on the right.

DEPLOY & RUN TRANSACTIONS:

- MOTHISTORY AT 0XA3A...2B18C (BLOCKCHAIN)
- SERVICEHISTORY AT 0XB17...2CAC9 (BLOCKCHAIN) - This section is expanded, showing:
 - Balance: 0 ETH
 - Approve: address to, uint256 tokenId
 - Mint: address _vehicleNftAddress, uint256 _vehicle
 - SafeTransferFrom: address from, address to, uint256 tokenId
 - SafeTransferFrom: address from, address to, uint256 tokenId, bytes memory _data
 - SetApprovalForAll: address operator, bool approved
 - TransferFrom: address from, address to, uint256 tokenId
 - AccessControl
 - BalanceOf: address owner
 - GetApproved: uint256 tokenId
 - GetServiceBookEntry: uint256 tokenId

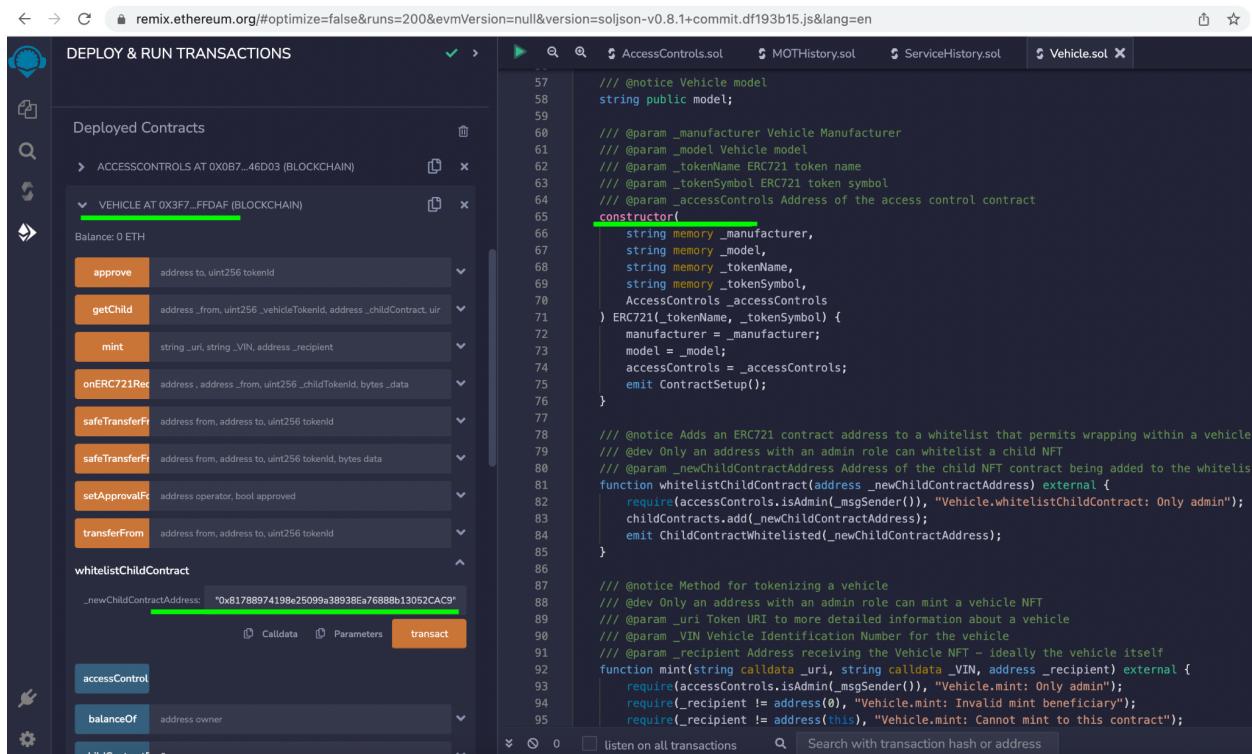
ServiceHistory.sol Code:

```
14 /// @author Vincent de Almeida
15 /// @notice Last Updated 2 Jan 2021
16 contract ServiceHistory is ERC721URIStorage {
17     using SafeMath for uint256;
18     using Counters for Counters.Counter;
19     Counters.Counter private _tokenIds;
20     mapping(string => uint8) hashes;
21
22
23     // Service book entry data
24     struct Entry {
25         uint256 mileage;
26         uint256 date;
27         address garage;
28         string description;
29     }
30
31     /// @notice ServiceHistory token ID => Entry info
32     mapping(uint256 => Entry) public serviceBookEntry;
33
34     /// @notice Address of the access control contract
35     AccessControls public accessControls;
36
37     /// @param _accessControls Address of the access control contract
38     constructor(AccessControls _accessControls) ERC721("ServiceHistory", "SRV") {
39         accessControls = _accessControls;
40     }
41
42     /// @notice Method for tokenizing a service book entry and instantly wrapping within a vehicle NFT
43     /// @dev Only an address with the garage role (authorised service partner) can invoke this method
44     /// @param _vehicleNftAddress Address of the vehicle NFT contract
45     /// @param _vehicleNftId Vehicle token ID receiving the ServiceHistory NFT
46 }
```

At the bottom of the code editor, there is a note: 0: tuple(uint256,uint256,address,string): 50000,167787 2013,0xd56e60C1E9B5a69Ccf498a8a9Fd8973211 d298C_Oil Change

Compile and Deploy Smart Contract Vehicle

Compile Vehicle smart contract in Remix and deploy it with the deployed address of AccessControls. Then add deployed address of MotHistory and ServiceHistory contracts in whitelistChildContract of Vehicle contract.



The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' sidebar lists 'Deployed Contracts' under 'VEHICLE AT 0X3F7...FFDAF (BLOCKCHAIN)'. It shows a balance of 0 ETH and lists several functions: approve, getChild, mint, onERC721Rec, safeTransferFrom, safeTransferFrom, setApprovalFor, and transferFrom. Below these is a 'whitelistChildContract' section with a field '_newChildContractAddress' containing the value '0x81788974198e25099a38938e76888b13052CAC9'. On the right, the main panel displays the Solidity code for the Vehicle contract. The code includes imports for AccessControls.sol, MOTHistory.sol, ServiceHistory.sol, and Vehicle.sol. It defines a constructor that takes manufacturer, model, tokenName, tokenSymbol, and accessControls as parameters. It also includes a function to add a child contract to a whitelist and another to mint a vehicle NFT. The code is annotated with comments explaining its purpose.

```
// SPDX-License-Identifier: MIT
// @notice Vehicle model
string public model;

// @param _manufacturer Vehicle Manufacturer
// @param _model Vehicle model
// @param _tokenName ERC721 token name
// @param _tokenSymbol ERC721 token symbol
// @param _accessControls Address of the access control contract
constructor(
    string memory _manufacturer,
    string memory _model,
    string memory _tokenName,
    string memory _tokenSymbol,
    AccessControls _accessControls
) ERC721(_tokenName, _tokenSymbol) {
    manufacturer = _manufacturer;
    model = _model;
    accessControls = _accessControls;
    emit ContractSetup();
}

/// @notice Adds an ERC721 contract address to a whitelist that permits wrapping within a vehicle
/// @dev Only an address with an admin role can whitelist a child NFT
/// @param _newChildContractAddress Address of the child NFT contract being added to the whitelist
function whitelistChildContract(address _newChildContractAddress) external {
    require(accessControls.isAdmin(_msgSender()), "Vehicle.whitelistChildContract: Only admin");
    childContracts.add(_newChildContractAddress);
    emit ChildContractWhitelisted(_newChildContractAddress);
}

/// @notice Method for tokenizing a vehicle
/// @dev Only an address with an admin role can mint a vehicle NFT
/// @param _uri Token URI to more detailed information about a vehicle
/// @param _VIN Vehicle Identification Number for the vehicle
/// @param _recipient Address receiving the Vehicle NFT - ideally the vehicle itself
function mint(string calldata _uri, string calldata _VIN, address _recipient) external {
    require(accessControls.isAdmin(_msgSender()), "Vehicle.mint: Only admin");
    require(_recipient != address(0), "Vehicle.mint: Invalid mint beneficiary");
    require(_recipient != address(this), "Vehicle.mint: Cannot mint to this contract");
}
```

Add Smart Contracts Deployed Addresses in .env file

Add smart contracts deployed addresses in the .env file of LuckyDay web app.

MNEMONIC='advance era food settle shrug safe taxi sight sun core kite clean'

VEHICLE_SMART_CONTRACT_ADDRESS='0x3f7A23B9AD479DC243f7bB325F66DF28d85FFdAf'

ACCESSCONTROLS_SMART_CONTRACT_ADDRESS='0x0b791a231fd946D322abdf2DBBa54ABd97D46D03'

MOTHISTORY_SMART_CONTRACT_ADDRESS='0xa3A05560e8cE9e5784BCb0D1b04F76Cd04C2B18C'

SERVICE_HISTORY_SMART_CONTRACT_ADDRESS='0x81788974198e25099a38938Ea76888b13052CAC9'

WEB3_PROVIDER_URI='HTTP://127.0.0.1:7545'

```
.env — lucky_day_multipage
service_history_abi.json .env accesscontrols_abi.json sample.env

app > .env
1 MNEMONIC='advance era food settle shrug safe taxi sight sun core kite clean'
2 VEHICLE_SMART_CONTRACT_ADDRESS='0x3f7A23B9AD479DC243f7bB325F66DF28d85FFdAf'
3 ACCESSCONTROLS_SMART_CONTRACT_ADDRESS='0x0b791a231fd946D322abdf2DBBa54ABd97D46D03'
4 MOTHISTORY_SMART_CONTRACT_ADDRESS='0xa3A05560e8cE9e5784BCb0D1b04F76Cd04C2B18C'
5 SERVICE_HISTORY_SMART_CONTRACT_ADDRESS='0x81788974198e25099a38938Ea76888b13052CAC9'
6 WEB3_PROVIDER_URI='HTTP://127.0.0.1:7545'
7
```