# Ideas

Quizá tenga más sentido usar un contrato sin más que un NFT. Cuando se hace la escritura con el viaje se modifican ciertas variables asociadas a la dirección que envía la transacción y punto. Esto genera eventos que se leen y se sacan en la interfaz.

Para calcular la huella de carbono en viajes nacionales el usuario introduce ciudad de origen y ciudad de destino (esto si el proyecto se desarrollase más se haría de manera sencilla desde la propia red wifi del tren introduciendo el localizador del viaje o algo así).

¿¿Metatransactions??

Conectar mediante API con Chainlink functions para los cálculos de huella de carbono. Hemos visto la [API de GreenPerk](https://developers.travelperk.com/docs/greenperk-api?_gl=1*a1e8it*_ga*MTAyNDU4NTU1OS4xNzAyMzkyMjQ1*_ga_NJT5XL96L4*MTcwMjM5MjI0NS4xLjEuMTcwMjM5MjI2MS40NC4wLjA.), que pinta bien.

Se podría usar Chainlink automation para actualizar el año.

# Contrato

// SPDX-License-Identifier: GPL-3.0

pragma solidity ^0.8.20;

/\*\*

 \* @author Antonio Rodríguez-Ynyesto

 \* @title CarbonFootprint

 \* @dev Store the carbon footprint avoided by train travelers compared to car or plane

 \*/

contract CarbonFootprintAvoided {

    /////////////////////

    // State variables //

    /////////////////////

    address immutable i\_owner;

    mapping(uint16 year => mapping(address user => uint256 carbonFootprintAvoided)) private

        s\_userToCarbonFootprintAvoided;

    address[] s\_users;

    uint16 private \_year = 2023;

    ////////////

    // Events //

    ////////////

    event CarbonFootprintUpdated(address indexed user, uint256 indexed carbonFootprintAvoided);

    ///////////////

    // Modifiers //

    ///////////////

    modifier onlyOwner() {

        require(msg.sender == i\_owner);

        \_;

    }

    constructor() {

        i\_owner = msg.sender;

    }

    /\*\*

     \* @dev Store value in variable

     \* @param \_carbonAvoided value to store

     \*/

    function updateCarbonFootprintAvoided(uint256 \_carbonAvoided) external {

        s\_userToCarbonFootprintAvoided[\_year][msg.sender] += \_carbonAvoided;

        emit CarbonFootprintUpdated(msg.sender, s\_userToCarbonFootprintAvoided[\_year][msg.sender]);

    }

    /\*\*

     \* @dev Return value

     \* @return value of 'number'

     \*/

    function getCarbonFootPrintAvoided() external view returns (uint256) {

        return s\_userToCarbonFootprintAvoided[\_year][msg.sender];

    }

    /\*\*

     \* @dev Update year

     \* @param \_newYear the new year

     \*/

    function updateToNewYear(uint16 \_newYear) external onlyOwner {

        \_year = \_newYear;

    }

}

# Pruebas

// SPDX-License-Identifier: UNLICENSED

pragma solidity ^0.8.13;

import {Test, console2} from "forge-std/Test.sol";

import {CarbonFootprintAvoided} from "../src/CarbonFootprintAvoided.sol";

contract CarbonFootprintAvoidedTest is Test {

    /\*\*

     \* Events

     \*/

    event CarbonFootprintUpdated(address indexed user, uint256 indexed carbonFootprintAvoided);

    CarbonFootprintAvoided public carbonFootprintAvoided;

    address USER = makeAddr("user");

    uint256 CARBON\_AVOIDED = 123456;

    function setUp() external {

        carbonFootprintAvoided = new CarbonFootprintAvoided();

    }

    function testUpdateCarbonFootprintAvoided() external {

        carbonFootprintAvoided.updateCarbonFootprintAvoided(CARBON\_AVOIDED);

        assertEq(carbonFootprintAvoided.getCarbonFootPrintAvoided(), CARBON\_AVOIDED);

    }

    function testFuzzUpdateCarbonFootprintAvoided(uint256 x) external {

        carbonFootprintAvoided.updateCarbonFootprintAvoided(x);

        assertEq(carbonFootprintAvoided.getCarbonFootPrintAvoided(), x);

    }

    function testEmitsEventOnUpdateCarbonFootprintAvoided() external {

        vm.prank(USER);

        vm.expectEmit(true, false, false, false, address(carbonFootprintAvoided));

        emit CarbonFootprintUpdated(USER, CARBON\_AVOIDED);

        carbonFootprintAvoided.updateCarbonFootprintAvoided(CARBON\_AVOIDED);

    }

    function testUpdateYear() external {

        vm.prank(address(this));

        carbonFootprintAvoided.updateToNewYear(2024);

        vm.expectRevert();

        vm.prank(USER);

        carbonFootprintAvoided.updateToNewYear(2024);

    }

}

Texto

Descripción generada automáticamente

# Despliegue

// SPDX-License-Identifier: UNLICENSED

pragma solidity ^0.8.13;

import {Script, console2} from "forge-std/Script.sol";

import {CarbonFootprintAvoided} from "../src/CarbonFootprintAvoided.sol";

contract DeployCarbonFootprintAvoided is Script {

    function setUp() public {}

    function run() external returns (CarbonFootprintAvoided) {

        vm.broadcast();

        CarbonFootprintAvoided carbonFootprintAvoided = new CarbonFootprintAvoided();

        return (carbonFootprintAvoided);

    }

}

**source .env**

**forge script script/DeployCarbonFootprintAvoided.s.sol --rpc-url $SEPOLIA\_RPC\_URL --private-key $PRIVATE\_KEY --broadcast –verify**

[Contrato](https://sepolia.etherscan.io/address/0x2732bAbD3F4DA9d253B7c03fc2A8d1C1DbADE5a2) desplegado en la siguiente dirección: 0x2732bAbD3F4DA9d253B7c03fc2A8d1C1DbADE5a2

# Interfaz

**yarn create vite carbon-footprint-saved-dapp --template react**

**cd carbon-footprint-saved-dapp**

**yarn**

**yarn dev**

**yarn add @mui/material @emotion/react @emotion/styled**