TRAIN BLOCK: NEXT GENERATION TICKET SYSTEM USAGE OF NFTs

Introduction

Non-fungible tokens (NFTs) have gained popularity in recent years as a secure and verifiable way to represent digital assets. In this project, we explore the use of NFTs in IRCTC Railways to provide secure and verifiable ticketing for passengers. By using NFTs, passengers can purchase and redeem their tickets digitally, eliminating the need for paper tickets and reducing the risk of fraud. We will develop a DAPP that allows passengers to search for train routes, purchase NFT tickets, and store them securely on their mobile device. The application will feature faster online booking, resoldable tickets, and less need for ground surveillance. Additionally, we will associate some travel benefits with NFT tickets, such as priority boarding or access to lounges, and offer discounts for regular travelers on a specific route, incentivizing them to use NFT tickets for their journeys. Users can also take up loans on their NFTs, which they have to clear after the journey ends. This is beneficial in situations where there are network issues with payments. The application will also provide ticket inspectors with a way to verify the authenticity of NFT tickets. Through this project, we aim to provide a more secure, convenient, and rewarding ticketing system for IRCTC Railways passengers.

TECHNOLOGIES USED

Smart Contracts: In this system, smart contracts are used as the backend, which is nothing but a simple programme that runs upon the meeting of certain conditions and is stored on Blockchain [13,14,15]. • Solidity: To write our smart contract we used solidity language

Ethereum: To run and deploy the proposed dApp, Ethereum blockchain is used. The smart contract was built and executed on the Ethereum platform, which provides an Ethereum virtual machine as the smart contract's runtime environment.

• **Hardhat**: Hardhat has been used as the development environment to compile, test, and debug the code on various test cases. It provides one with a blockchain server that runs on the developer's machines locally. It creates several false accounts with fake ETH tokens in order to perform transactions, and test the code without the need to perform these transactions on any public blockchain with real ETH tokens.

• **Meta Mask wallet:** It is a widely used cryptocurrency wallet that allows one to access their Ethereum wallet via a mobile app or an extension. It has been used to connect securely to this decentralized application, manage and store account keys, send and receive ETH-based tokens and currencies, and perform and broadcast transactions.

• **ReactJS** : To build a user side interface, ReactJs is used. It is a NodeJS framework that has been used as the development environment, because JavaScript provides extensive support for blockchain-based applications.

Contracts

TICKET NFT(Curently Included In Project)

Solidity smart contract that inherits from the ERC721 standard token contract and allows for the creation and minting of unique tokens (in this case, tickets) on the Ethereum blockchain.

The contract defines a struct called Ticket, which has four fields: name, wallet, source, and destination. name is a string that represents the name of the ticket, wallet is the Ethereum address that owns the ticket, source and destination are strings that represent the source and destination of the ticket.

The contract also has a ticketCount variable, which keeps track of the total number of tickets created, and a tickets mapping, which maps each ticket ID to its corresponding Ticket struct.

The constructor function sets the name and symbol of the ERC721 token to "TicketNFT" and "TKT", respectively.

The mintTicket function allows users to create and mint new tickets. It takes four arguments: _name, _wallet, _source, and _destination, which correspond to the name, wallet, source, and destination fields of the Ticket struct. The function creates a new Ticket struct with these values, assigns it a unique ID (based on the current ticketCount), and stores it in the tickets mapping. The function then mints a new ERC721 token with the owner set to _wallet and the token ID set to the new ticket ID, and increments the ticketCount variable.

Overall, this contract provides a simple implementation for minting tickets as NFTs on the Ethereum blockchain.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "../node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol";

contract Mint is ERC721 {
    struct Ticket {
        string name;
        address wallet;
        string source;
        string destination;
    }

    uint256 public ticketCount;
    mapping(uint256 => Ticket) public tickets;

    constructor() ERC721("TicketNFT", "TKT") {}

    function mintTicket(string memory _name, address _wallet, string memory _source, string memory
_destination) public {
        Ticket memory newTicket = Ticket({
            name: _name,
            wallet: _wallet,
            source: _source,
            destination: _destination
        });

        uint256 newTicketId = ticketCount;
        tickets[newTicketId] = newTicket;

        _safeMint(_wallet, newTicketId);
        ticketCount++;
    }
}
```

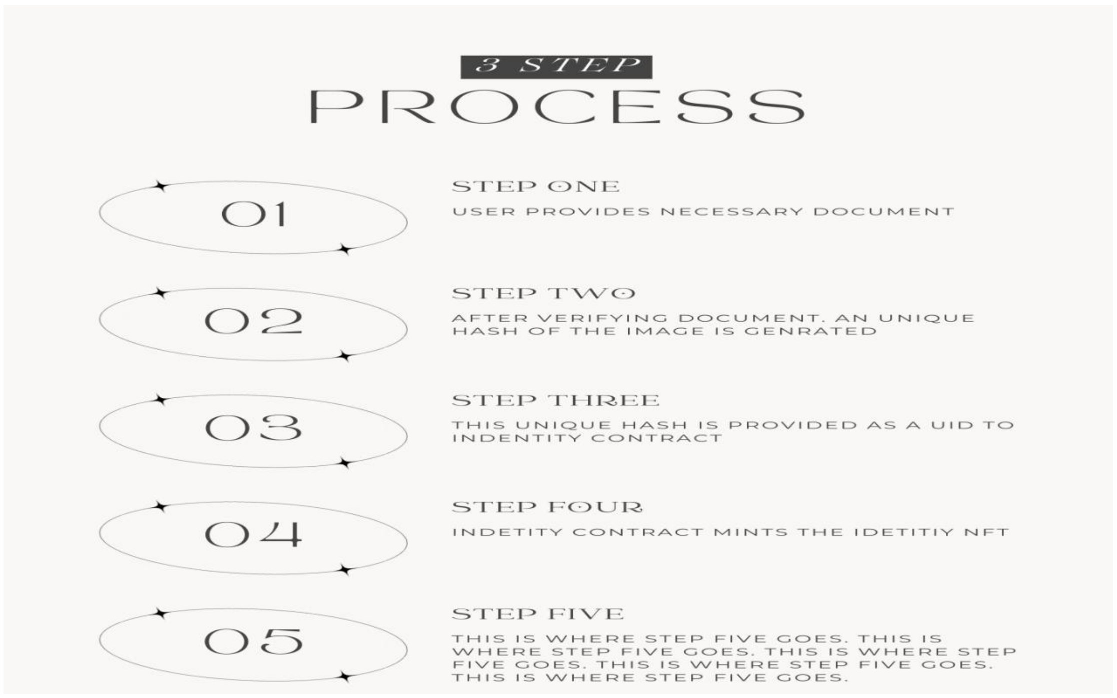
IDENTITY CONTRACT (Future Prospect)

The main purpose of this contact is to Mint an Unique Identity NFT for an Individual which in future could be linked to their Government Document. In future intead of a UID we use an government ID.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";

contract IdentityNFT is ERC721 {
    uint256 public tokenId = 0;
    mapping (uint256 => bool) private uidExists;
    mapping (uint256 => address) private uidToOwner;

    constructor(string memory name, string memory symbol) ERC721(name, symbol) {}

    function mintIdentityNFT(uint256 uid) public {
        require(!uidExists[uid], "UID already exists");
        tokenId++;
        _safeMint(msg.sender, tokenId);
        uidExists[uid] = true;
        uidToOwner[uid] = msg.sender;
    }

    function getOwner(uint256 uid) public view returns (address) {
        require(uidExists[uid], "UID does not exist");
        return uidToOwner[uid];
    }
}
```

In this smart contract, the IdentityNFT contract extends the ERC721 contract provided by the OpenZeppelin library. The ERC721 contract is a standard for creating non-fungible tokens (NFTs) on the Ethereum blockchain.
The tokenId variable is used to keep track of the unique identifier for each NFT. The uidExists mapping is used to keep track of whether a UID has already been used to mint an NFT. The uidToOwner mapping is used to keep track of the owner of an NFT associated with a particular UID.
The mintIdentityNFT function is used to mint an NFT for a particular UID. It first checks if the UID has already been used to mint an NFT. If it has, then it throws an error. If the UID has not been used before, then it increments the tokenId variable, mints an NFT with the new tokenId, and associates the UID with the owner of the new NFT.
The getOwner function can be used to get the owner of an NFT associated with a particular UID.
To use this smart contract, a user would call the mintIdentityNFT function, passing in their UID as a parameter. This would create a new NFT associated with their UID and transfer ownership of the NFT to the user's address. The user could then call the getOwner function to confirm that they are the owner of the NFT associated with their UID.

Working with Government IDs (Future Prospect)


3 STEP
PROCESS
01
STEP ONE
USER PROVIDES NECESSARY DOCUMENT
02
STEP TWO
AFTER VERIFYING DOCUMENT. AN UNIQUE HASH OF THE IMAGE IS GENRATED
03
STEP THREE
THIS UNIQUE HASH IS PROVIDED AS A UID TO INDENTITY CONTRACT
04
STEP FOUR
INDETITY CONTRACT MINTS THE IDETITIY NFT
05
STEP FIVE
THIS IS WHERE STEP FIVE GOES. THIS IS WHERE STEP FIVE GOES. THIS IS WHERE STEP FIVE GOES. THIS IS WHERE STEP FIVE GOES. THIS IS WHERE STEP FIVE GOES.

Sell Ticket Contract(Future Prospect)-

In this trasnferNFT we basically sell our ticket if we do not need it.
This function takes in two arguments - to and tokenId. The to argument is the address of the user to whom the TicketNFT ownership will be transferred. The tokenId argument is the unique identifier of the TicketNFT.

The require statement checks if the sender of the transaction is the current owner of the TicketNFT. If the sender is not the current owner, the function will not execute and will throw an error message.

If the sender is the current owner, the safeTransferFrom function is called to transfer the ownership of the TicketNFT to the user specified in the to argument. This function is provided by the OpenZeppelin ERC721 implementation and ensures that the transfer is done safely and securely.

Once the transfer is successful, the ownership of the TicketNFT will be updated in the blockchain, and the new owner can now control the NFT

```
//CODE STARTS

function transferTicketNFT(address to, uint256 tokenId) public {
    require(ownerOf(tokenId) == msg.sender, "You are not the owner of this Ticket NFT");
    safeTransferFrom(msg.sender, to, tokenId);
}

//CODE ENDS
```

A question arises doest it would increase the fraud of peoplebuying more tickets. The answer is a simple NO. In future the contracts would be further designed to Give back 30% back to IRCTC and rest to the last owner if the ticket is sold and all of this could only be done via on-Chain Transactions.

Flow Of Project

Developing Front-End

The front-end of the SolveForIndia project is built using React, a popular JavaScript library for building user interfaces. The components in this project are located in the src/components directory. Each component has its own folder containing a JavaScript file and a CSS file. The JavaScript file defines the component, while the CSS file defines its styles.

One of the main components in the project is the App component. This component is the root component of the application and is responsible for rendering all other components. It contains the application state and handles user interactions. The Header component is another key component in the project. This component renders the header of the application.he front-end of the SolveForIndia project is built using React and follows a component-based architecture. It uses several third-party libraries and packages to enhance its functionality and user experience.

Devloping Smart Contract -

Solidity smart contract that inherits from the ERC721 standard token contract and allows for the creation and minting of unique tokens (in this case, tickets) on the Ethereum blockchain.

The contract defines a struct called Ticket, which has four fields: name, wallet, source, and destination. name is a string that represents the name of the ticket, wallet is the Ethereum address that owns the ticket, source and destination are strings that represent the source and destination of the ticket.

The contract also has a ticketCount variable, which keeps track of the total number of tickets created, and a tickets mapping, which maps each ticket ID to its corresponding Ticket struct.

The constructor function sets the name and symbol of the ERC721 token to "TicketNFT" and "TKT", respectively.

The mintTicket function allows users to create and mint new tickets. It takes four arguments: _name, _wallet, _source, and _destination, which correspond to the name, wallet, source, and destination fields of the Ticket struct. The function creates a new Ticket struct with these values, assigns it a unique ID (based on the current ticketCount), and stores it in the tickets mapping. The function then mints a new ERC721 token with the owner set to _wallet and the token ID set to the new ticket ID, and increments the ticketCount variable.

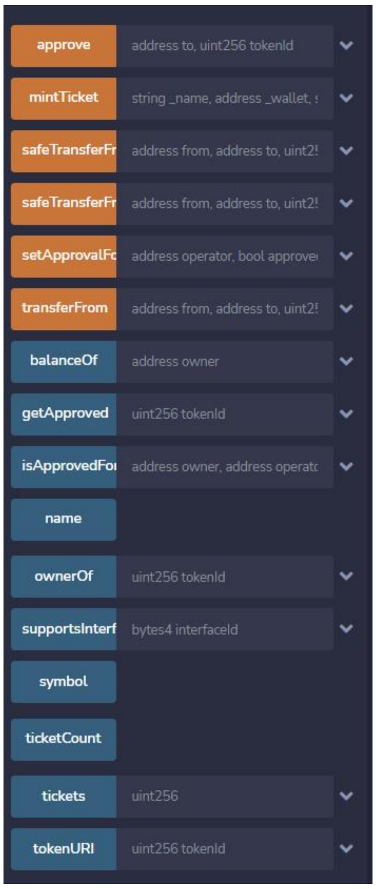
Overall, this contract provides a simple implementation for minting tickets as NFTs on the Ethereum blockchain.

FUNCTIONS

We have to work with **mintTicket** function which takes in Name , Destination , Source and WalletAddress as parameters.

To verify our NFT we use **mapping of tickets** to with Tickets starting from 0.

Other Function comes preload-ed with ERC721 Contract.



Deploying On Local Hardhat -

Install Hardhat: If you haven't already done so, you need to install Hardhat on your machine. You can do this by running npm install --save-dev hardhat in your project directory.

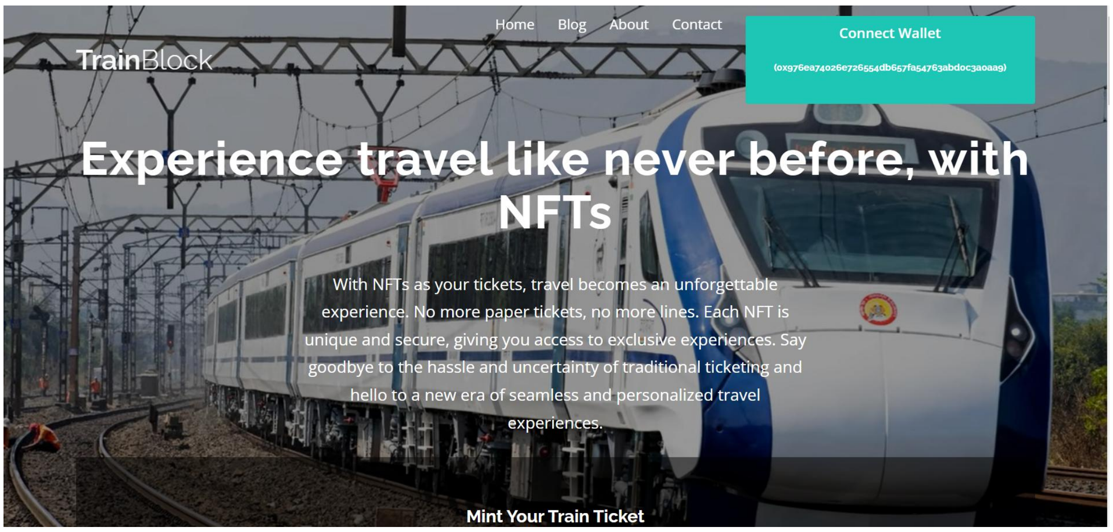
Configure Hardhat: You need to create a Hardhat configuration file hardhat.config.js in your project directory. This file should contain information about the network you want to deploy to.

Create a Deployment Script: You need to create a deployment script in a deploy.js file.

Run the Deployment Script: You can run the deployment script by executing the following command in your terminal: npx hardhat run --network development deploy.js. This will deploy your contract to the specified network.

Results-

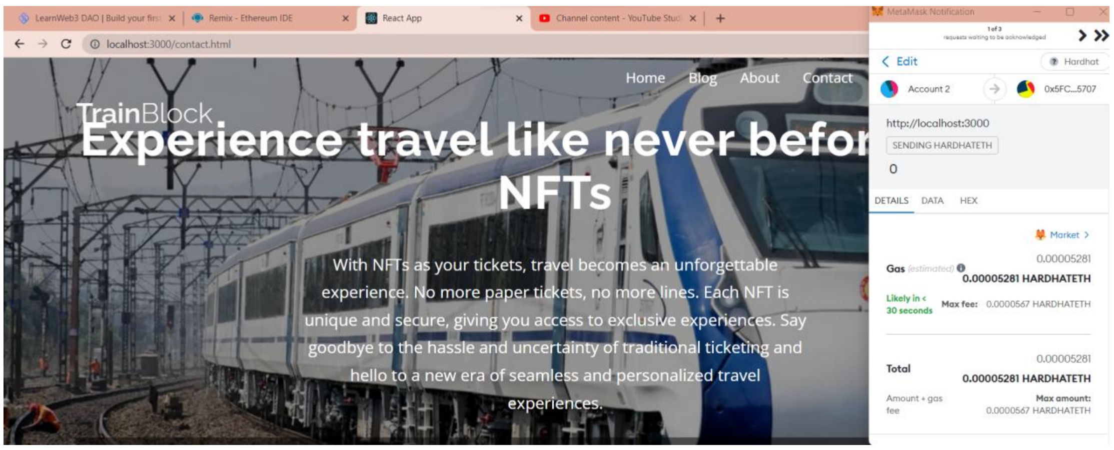
Current FrontEnd (with wallet connected) -



Experience travel like never before, with NFTs

With NFTs as your tickets, travel becomes an unforgettable experience. No more paper tickets, no more lines. Each NFT is unique and secure, giving you access to exclusive experiences. Say goodbye to the hassle and uncertainty of traditional ticketing and hello to a new era of seamless and personalized travel experiences.

Signing With MetaMask-



Experience travel like never befor
NFTs

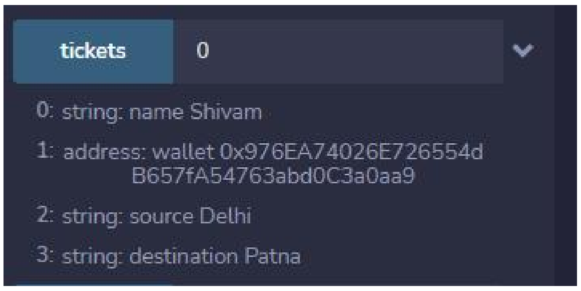
With NFTs as your tickets, travel becomes an unforgettable experience. No more paper tickets, no more lines. Each NFT is unique and secure, giving you access to exclusive experiences. Say goodbye to the hassle and uncertainty of traditional ticketing and hello to a new era of seamless and personalized travel experiences.

http://localhost:3000
SENDING HARDHATETH
0

Gas (estimated) 0.00005281
0.00005281 HARDHATETH
Likely in < 30 seconds Max fee: 0.0000567 HARDHATETH

Total 0.00005281
0.00005281 HARDHATETH
Amount + gas fee Max amount: 0.0000567 HARDHATETH

Verifying our minted NFT using Remix which is connected to our Local Hardhat Environment using Ticket Mapping -


```
tickets   0

0: string: name Shivam
1: address: wallet 0x976EA74026E726554d
          B657fA54763abd0C3a0aa9
2: string: source Delhi
3: string: destination Patna
```

Conclusion -

In conclusion, the NFT Ticket System developed for IRCTC has the potential to revolutionize the way tickets are issued and tracked. By using NFTs to represent tickets, the system provides a secure and immutable record of ticket ownership that can be easily verified by both customers and railway staff.

The system is built using Solidity, a smart contract language that is well-suited for developing decentralized applications on the Ethereum blockchain. The front-end is developed using React, a popular JavaScript framework that allows for the creation of user-friendly and responsive interfaces.

One of the key advantages of the system is that it eliminates the need for physical tickets, reducing the risk of lost or stolen tickets. Customers can easily purchase and manage their tickets through a simple and intuitive interface, while railway staff can use the system to quickly and accurately verify the ownership of tickets.

Overall, the NFT Ticket System is a promising application of blockchain technology in the railway industry, and has the potential to improve the efficiency, security, and transparency of ticketing processes.

FUTURE PROSPECTS OF THE PROJECTS

This part contains the ideas that were not included due to time constrains.

-- Adding other feature such as selling tickets as NFTs this would include a constrain that the ticket could not be sold above the price it was bought. And the reselling could be only done using using **on-chain transactions.**

--Adding feature of QR code generator to print an NFT with an scan-able QR which could be shown at stations.

--Improving UI with more data Fields

-- Uploading only metadata on-chain and storing the rest of information in file storage solutions such as IPFS. This would reduce the Gas cost of every transaction. This could be achieved by using Pinata as service.

-- Others improvements are also encouraged to be added.