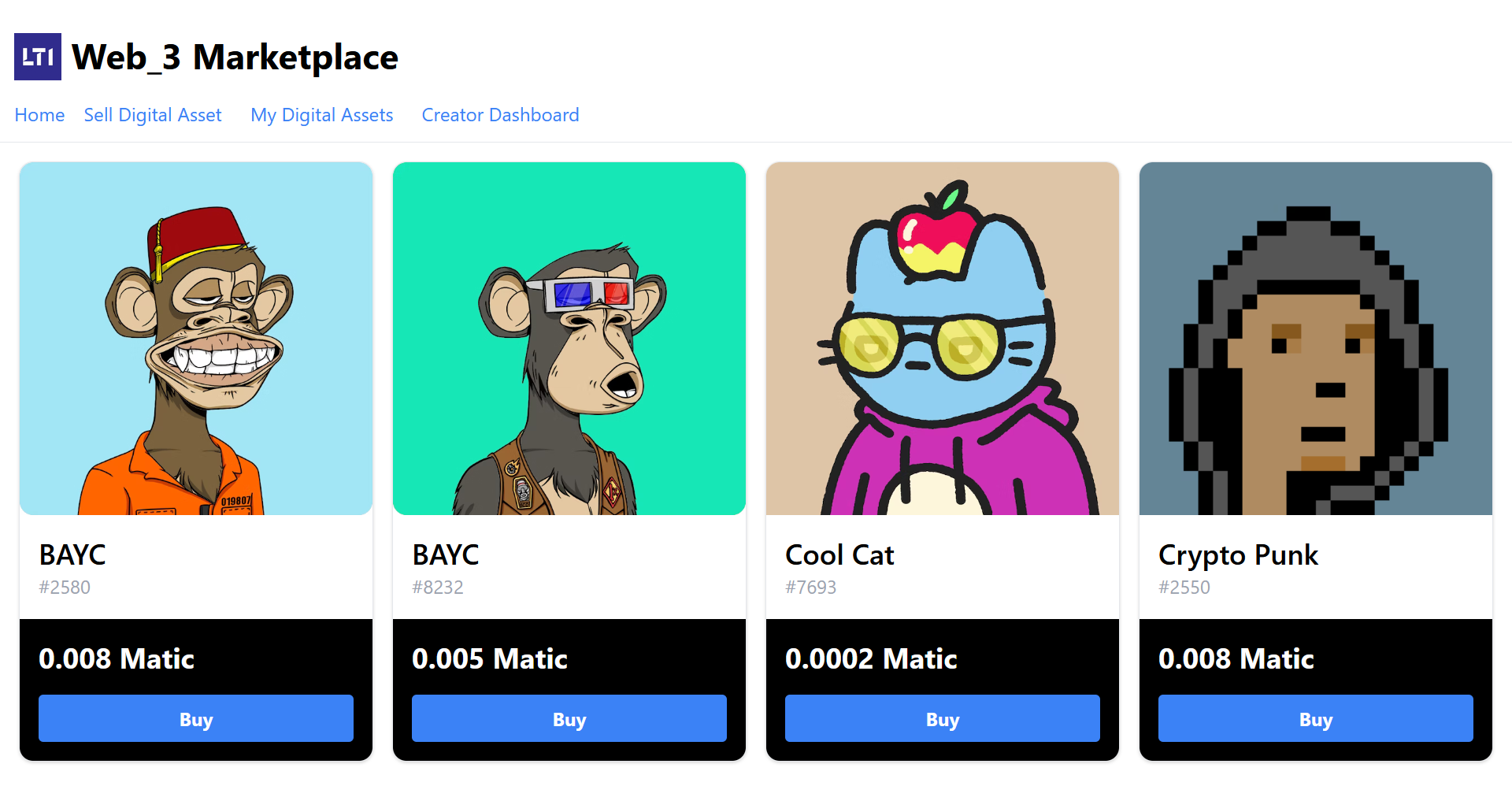
**How to Build a Full Stack NFT Marketplace**

In our project we have built a basic app on Ethereum using modern tooling like Hardhat and Ether.js.

The project that we have built is **Web3 Marketplace –** an NFT marketplace.



When a user puts an NFT for sale, the ownership of the item will be transferred from the creator to the marketplace contract.

When a user purchases an NFT, the purchase price will be transferred from the buyer to the seller and the item will be transferred from the marketplace to the buyer.

The marketplace owner will be able to set a listing fee. This fee will be taken from the seller and transferred to the contract owner upon completion of any sale, enabling the owner of the marketplace to earn recurring revenue from any sale transacted in the marketplace.

**Prerequisites**

1. We have installed Node.js version 16.14.0 in our machine.
2. Meta mask wallet extension installed as a browser extension

**The Stack**

Web application framework – Next.js

Solidity development environment – Hardhat

File Storage – IPFS

Ethereum Web Client Library – Ether.js

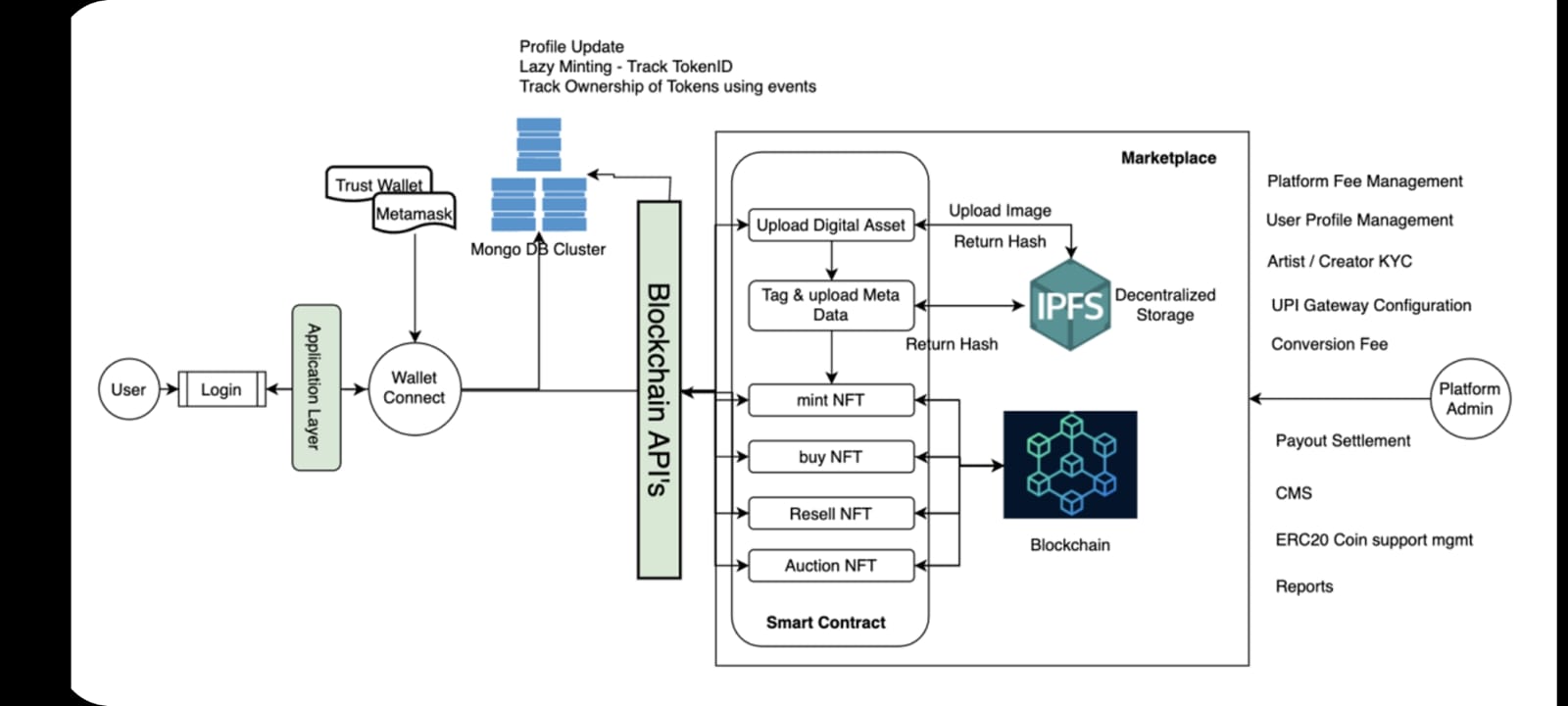
**Initializing the Project**

1. First, we create a new Next.js app using following command: **npx create-next-app nft-marketplace**
2. Then we have installed all the necessary libraries like ethers, hardhat, Ethereum-waffle, Tailwind CSS.
3. Then we have created a Smart Contract in the NFT marketplace.
4. Designed various pages for our project like creating dashboard, buy and selling of NFTs using tailwind CSS.
5. Imported accounts into Meta Mask.
6. Run and Deploying the NFT Marketplace to Mumbai Matic.

**NFT Architecture**

An NFT marketplace would have 3 basic components.

1. A user connects their wallet to the NFT marketplace.
2. A user creates an NFT marketplace token
3. A user creates a sell order or a buy order.



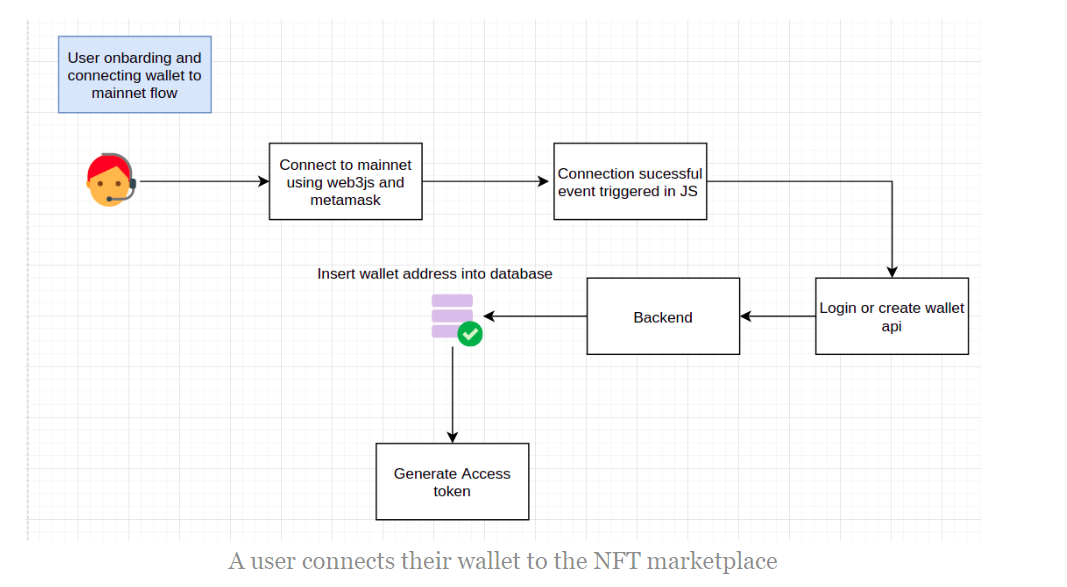
**Connecting wallet to the NFT marketplace**

A user needs to connect their wallet to the NFT marketplace to purchase an NFT. The connection is established between the user’s wallet and marketplace platform Via

**web3.js** and **meta mask**. Once the connection is established, an address is returned, and

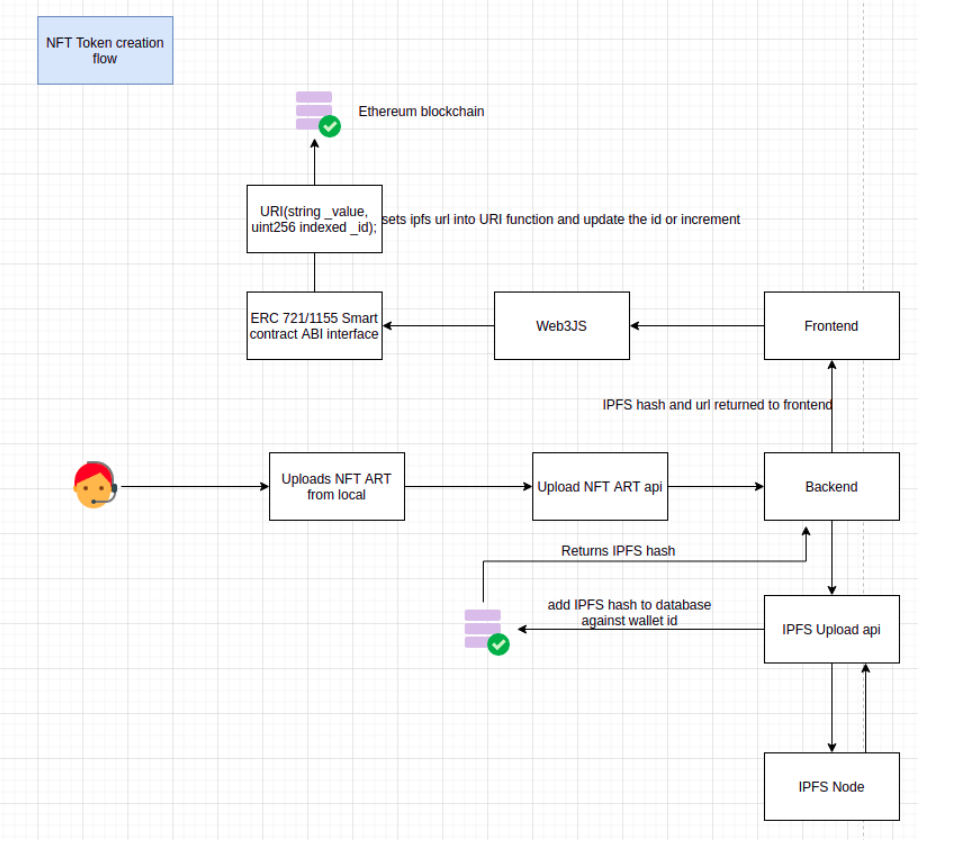
It can be sent to the server and store in the marketplace database via an API. At the same time, an access token can be generated that can be used to authenticate other API calls.

A pictorial representation of this process is given below.



**The user creates an NFT marketplace token**

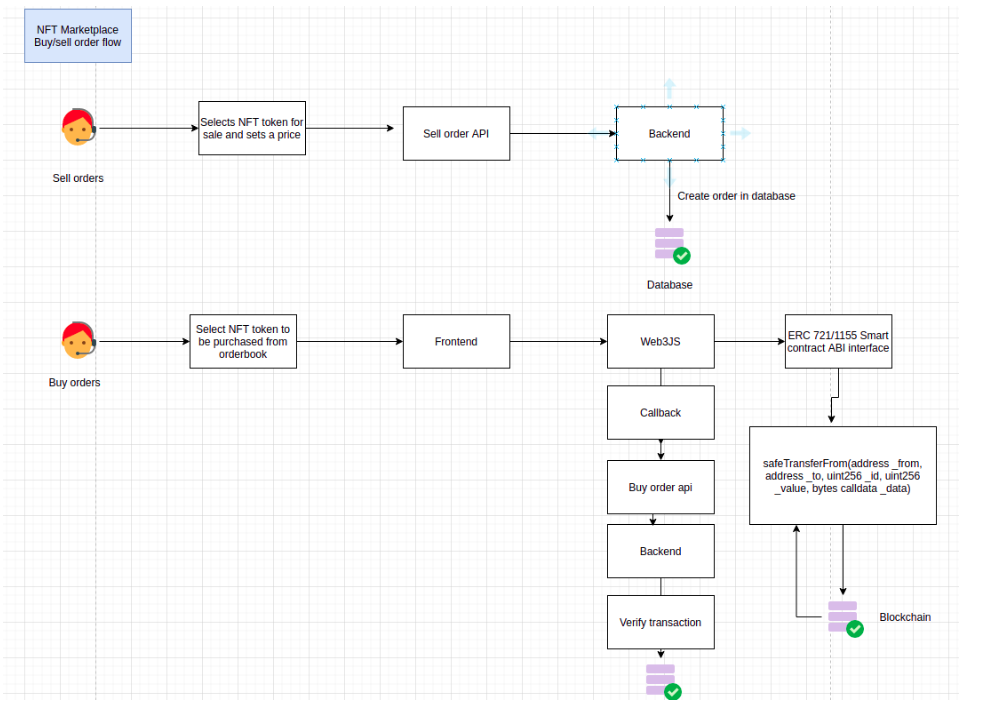
Now let’s see how the NFT token creation works. First, you need to create an API for uploading digital files to the server. This can be music, video, images etc. Once the file is uploaded into the back end it’s further saved into the IPFS node from where a hash is returned that can be then sent back to the front-end. Then, the URL can be stored into the blockchain using ERC 721/1155 smart contracts which are interfaced into the front-end using **ABI** and **Web3JS**. Any functions inside the contracts can be invoked from the front-end.



**A user creates a sell order or a buy order**

The logic for creating a sell order is simple. When the user selects an NFT token they created and places a sell order, the NFT will be listed in the order book. The logic for a Buy order is a bit more complex. Here, once a user clicks on an already listed sell order, they can send the Ethereum to the contract. The corresponding ID will be reassigned to the sender and finally,

The back end is notified that a buy order has been completed and verification is also done from the back end on the transaction to prove the validity.



**Deploying To Polygon**

We have deployed our NFT Marketplace to Mumbai test network. The first thing we will need to do is save one of our private keys from our wallet as an environment variable.

Next, we need to switch from the local test network to the Mumbai Testnet.

To do so, we need to create and set the network configuration.

Here, we will add the following configurations for the Mumbai test network as listed:

Network Name: **Mumbai Testnet**.

New RPC URL: <https://rpc-mumbai.maticvigil.com>

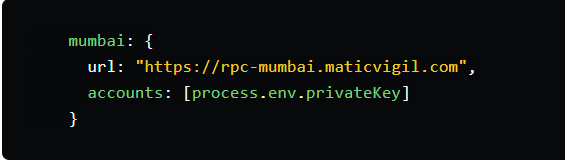
Chain ID: 80001

Currency Symbol: Matic

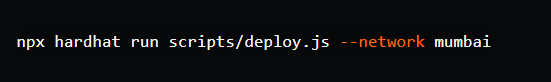
Then we need some testnet Matic tokens to interact with the applications. To get these, we have visited the **Matic Faucet,** inputting the address of the wallets that we would like to request the tokens.

**Deploying to the Matic/Polygon network**

Now that we have some Matic tokens, we can deploy to the Polygon network! To do so, be sure that the address associated with the private key we are deploying the contract with has received some Matic tokens to pay the gas fees for transaction.



To deploy to Matic, run the following command:

****

Once the contracts have been deployed, updates the loadNFTs function call in pages/index.js to include the new RPC endpoint. Then we are able to update the contract address in our project and test on the new network using command: **npm run dev**