NFT MARKETPLACE DOCUMENT

SAFFI KHAN

Contents

CHAPTER 1

Project Introduction:

Brief Description:

In our SuperSwap the users can login using their crypto wallet. They can select the tokens they want to buy Smartcontract will calculate the transaction fees based upon pricing algorithm and then show the gas fees and transaction fees. Similarly, if

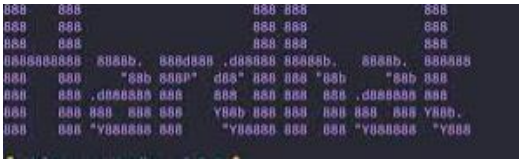
the user wants to create a non-fungible asset, then the listing price will be transferred to charity. User can deploy separate smart contracts for custom collections,

Project Beneficiary:

The normal users can trade their non-fungible tokens without any third-party. The charities like **Edhi foundation** and others get money at charged from users at each transaction.

Useful Tools and Technologies:

In following section, we will be giving brief description of the tools and technologies that we will be using for our project.



Hardhat

Hardhat provides development environment and testing tools for smart contract.

React JS

React and Nextjs will be used for client side and server-side rendering.

Ethereum

Ethereum virtual machine will be used as block chain.

Project Work Break Down

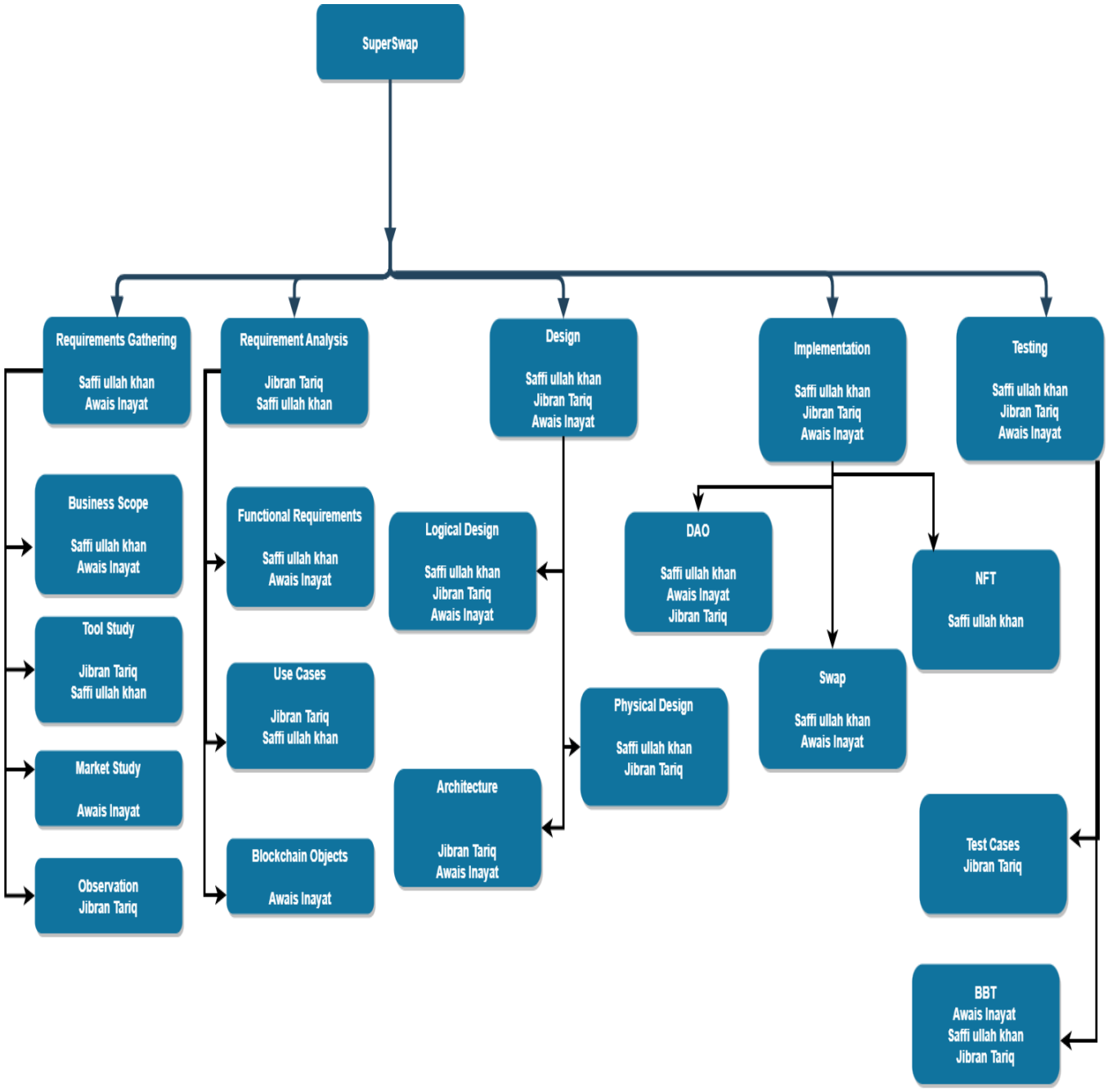


Figure1.1

Project Lifecycle:

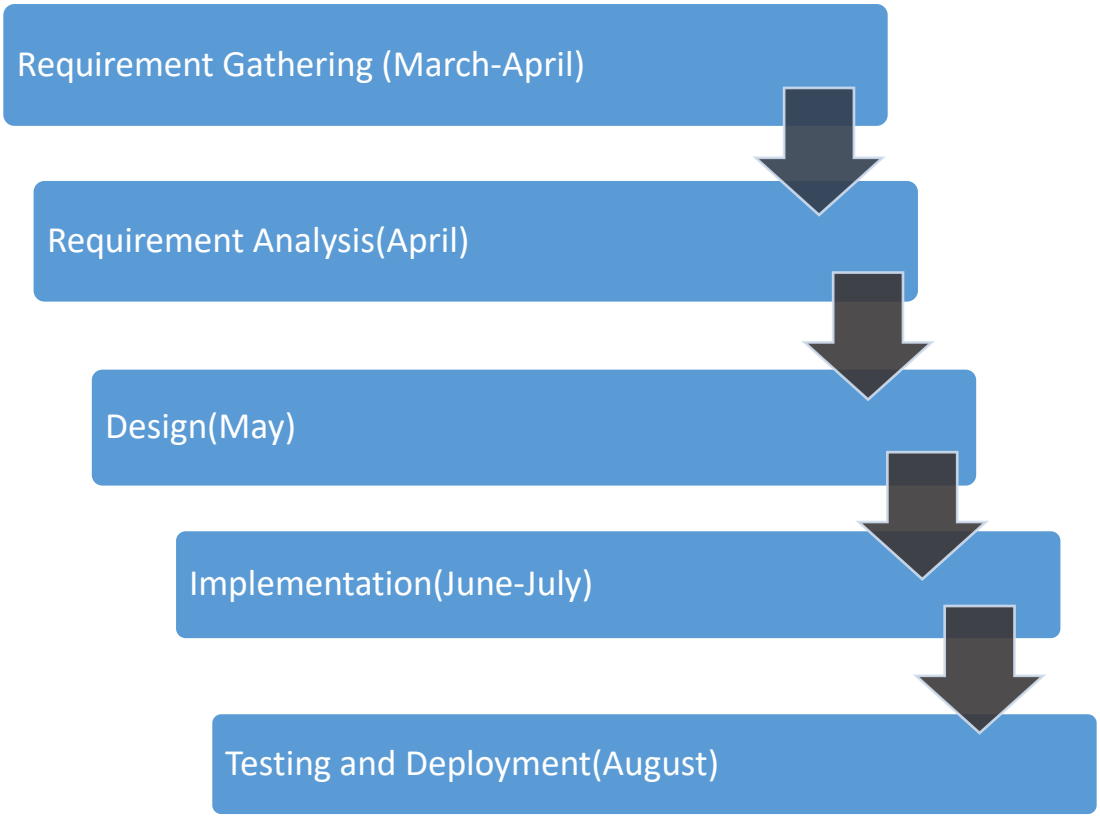


Figure 1.2

Chapter 2

Requirement Specification and Analysis

Non-Functional Requirements

S. No.	Non-Functional Requirements	Category
1	System should give quick response.	Performance
2	The system should keep and retrieve NFT correctly.	Reliability
3	System should give quick response.	Performance
4	The system should verify the information correctly.	Security
5	Performance analysis should be fast.	Performance
6	The up time of the System is 99%.	Availability

table2.2

Selected Functional Requirements

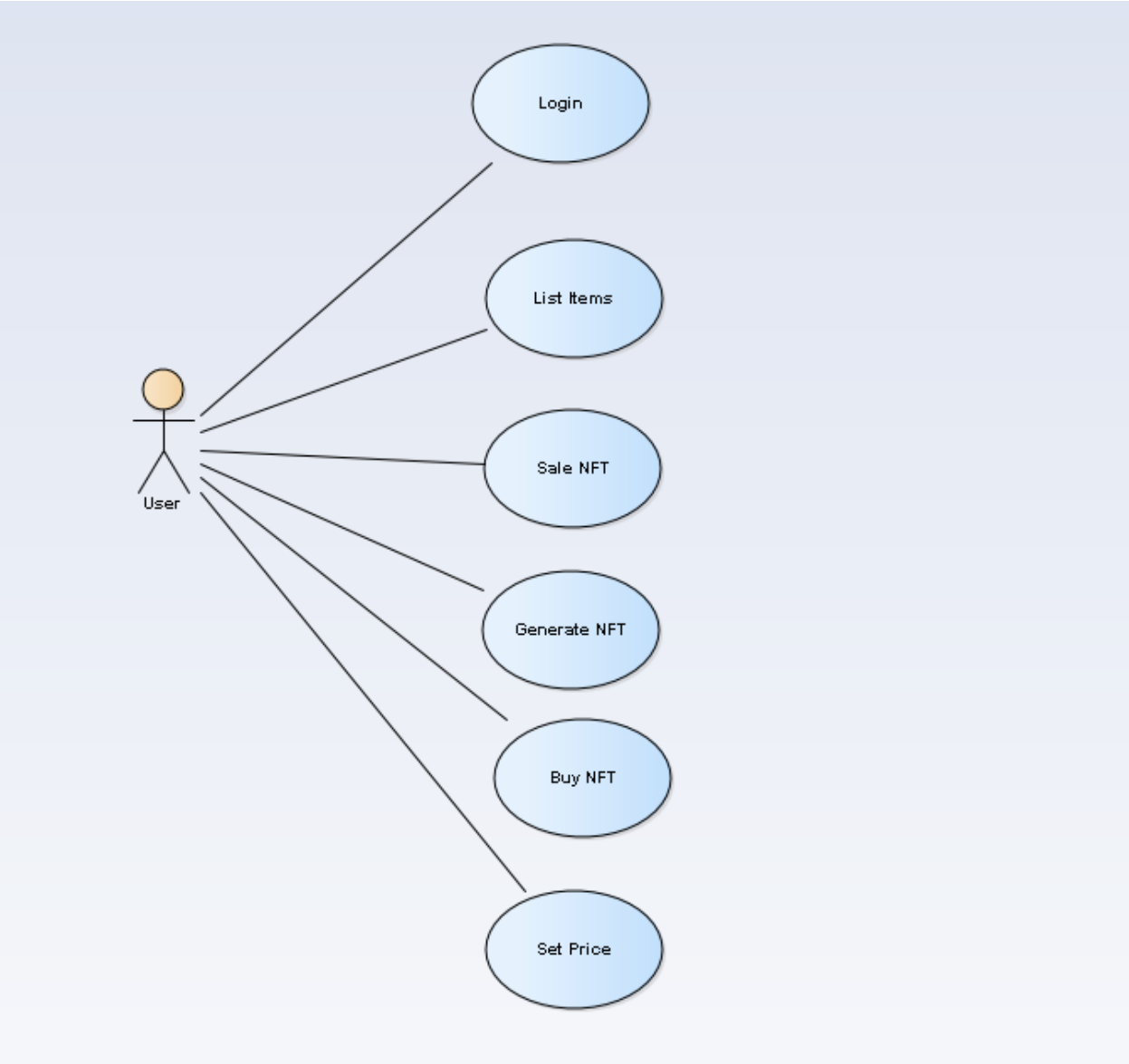
S. No.	Functional Requirement	Type	Status
1	User will be able to connect the wallet	Core	Completed
2	User will be able to see list items.	Core	Completed
3	User will be able to sell NFT.	Core	Completed
4	User will be able to generate NFT.	Core	Completed

5	User will be able to buy NFT	Core	Completed
6	User will be able to set prices.	Core	Completed
7	User will be able to launch collection	Core	Completed

table2.3

Usecase

SuperSwap



Use Cases

Use Case ID:	01		
Use Case Name:	List items		
Created By:	Saffi Ullah	Last Updated By:	Saffi khan
Date Created:	18/04/2022	Last Revision Date:	28/04/2022
Actors:	User		
Description:	User will select the items option.		
Trigger:	Accept offer.		
Preconditions:	User must login into the system		
Post conditions:	User will see the lists.		

Normal Flow:	User	System
	1. User must login into the system. 2. User select the list option.	1. The system shows the different options. 2. The system display the NFTs.
Alternative Flows:	System creates the error.	
Exceptions:	User enters the incorrect information. The system is not responding.	

Use Case ID:	02		
Use Case Name:	Sale NFT		
Created By:	Saffi Ullah	**Last Updated By:**	Jibran Tariq
Date Created:	18/04/2022	**Last Revision Date:**	22/04/2022
Actors:	User		
Description:	User will select the price for sale.		
Trigger:	User selected price.		
Preconditions:	User must login into the system		
Post conditions:	User will see the sold option.		

Normal Flow:	**User**	**System**
	1. User must login into the system. 2. User select the sale NFT button.	1. The system shows the different options. 2. The system display the NFT options. 3. Deal done successfully.
Alternative Flows:	System creates the error.	
Exceptions:	User enters the incorrect information. The system is not responding.	

Use Case ID:	03		
Use Case Name:	Generate NFTs		
Created By:	Saffi Ullah	**Last Updated By:**	Saffi Ullah
Date Created:	18/04/2022	**Last Revision Date:**	22/04/2022
Actors:	User		
Description:	User will login and click the generate button.		
Trigger:	button		
Preconditions:	User must login		

Post conditions:	User will see the NFTs generate system	
Normal Flow:	**User**	**System**
	User click generate option.	The system display the generate system.
Alternative Flows:	System creates the error.	
Exceptions:	The system is not responding.	

Use Case ID:	04		
Use Case Name:	Buy NFT		
Created By:	Saffi Ullah	**Last Updated By:**	Saffi Ullah
Date Created:	17/04/2022	**Last Revision Date:**	22/04/2022
Actors:	User		
Description:	User will buy NFT for future use		
Trigger:	Buy NFT button		
Preconditions:	User must login into the system		
Post conditions:	System will save data		
Normal Flow:	**User**	**System**	

	1. User enters the Text in the text fields and search the NFTs 2. User select the button save. 3. User select buy and give offer	1. The system displays the NFTs 2. System saves data successfully 3. Offer accepted
Alternative Flows:	System creates the error.	

Use Case ID:	5		
Use Case Name:	Set price for NFT		
Created By:	Saffi Ullah	**Last Updated By:**	Saffi Ullah
Date Created:	18/04/2022	**Last Revision Date:**	22/04/2022
Actors:	User		
Description:	User will select the set price option.		
Trigger:	Set		
Preconditions:	User must login into the system		
Post conditions:	User will see the set price		
Normal Flow:	**User**	**System**	

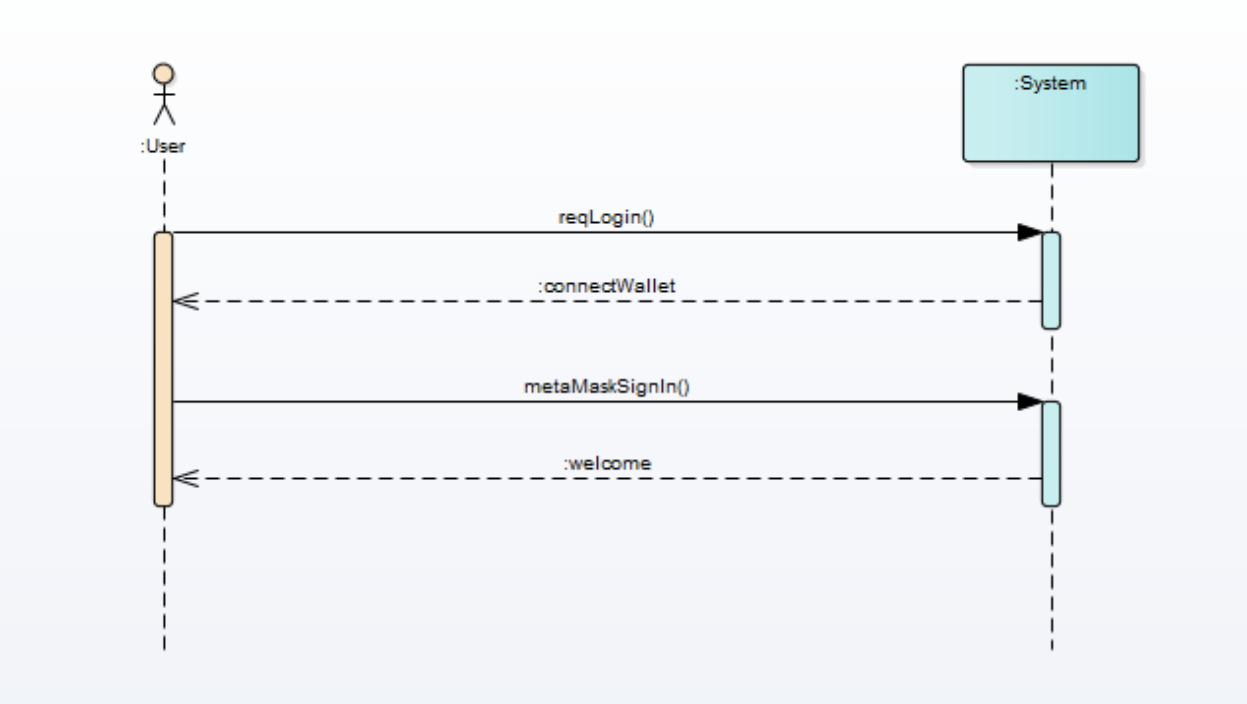
	1. User must login into the system. 2. User select the set price of NFT button. 3. User will select the price.	1. The system shows the different options. 2. The system display the NFT options. 3. Deal done successfully.
Alternative Flows:	System creates the error.	
Exceptions:	User enters the incorrect information. The system is not responding.	

Chapter 3

System Design

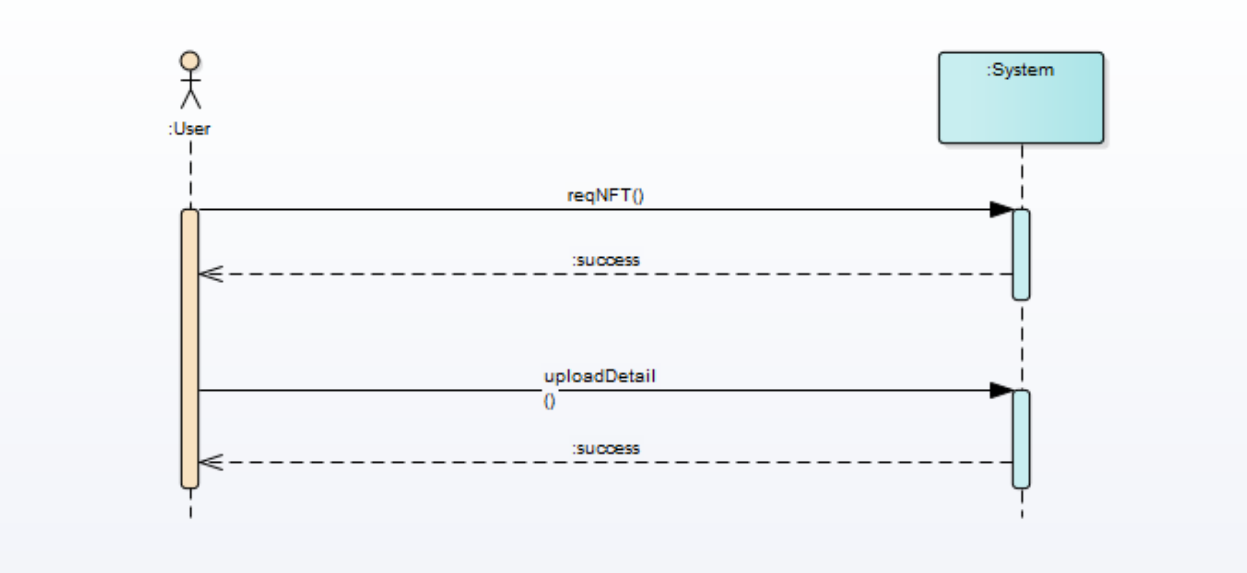
System sequence diagram

Login



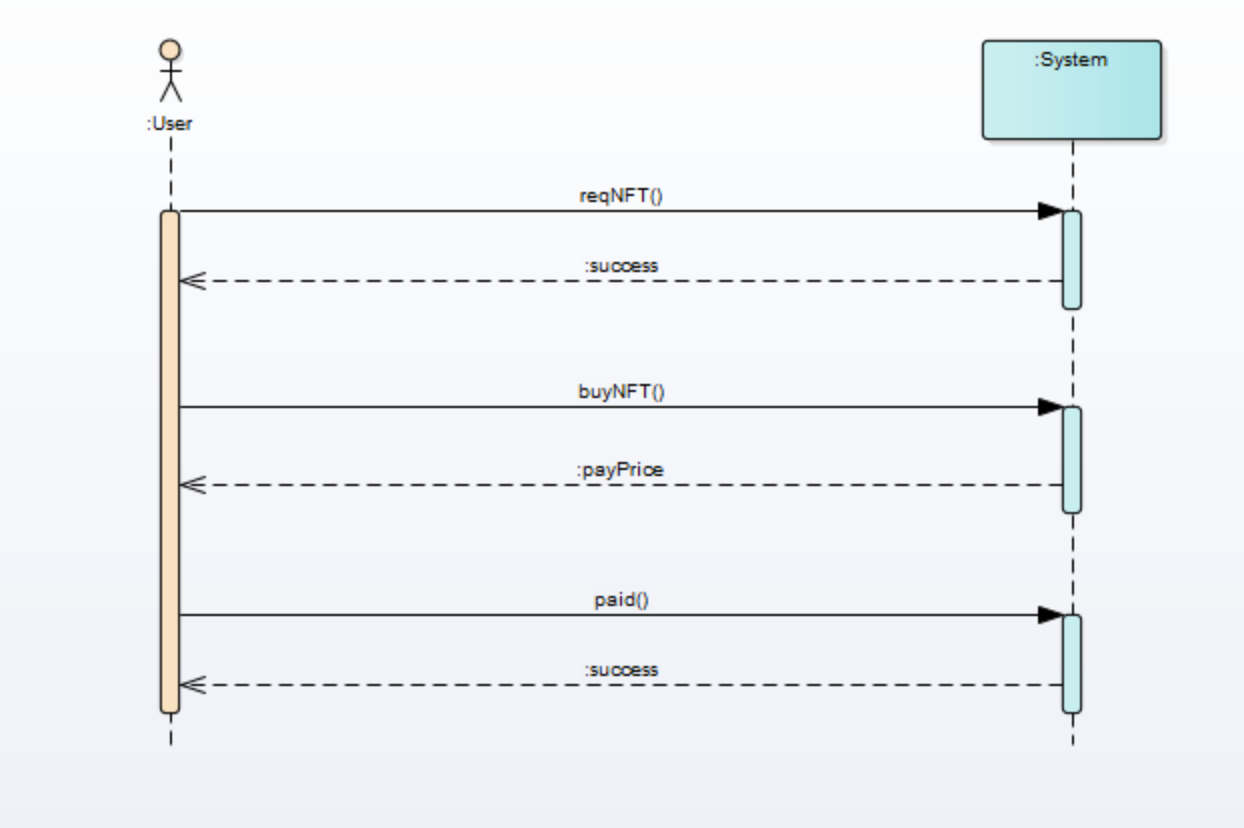
Fig#3.1

NFT generation



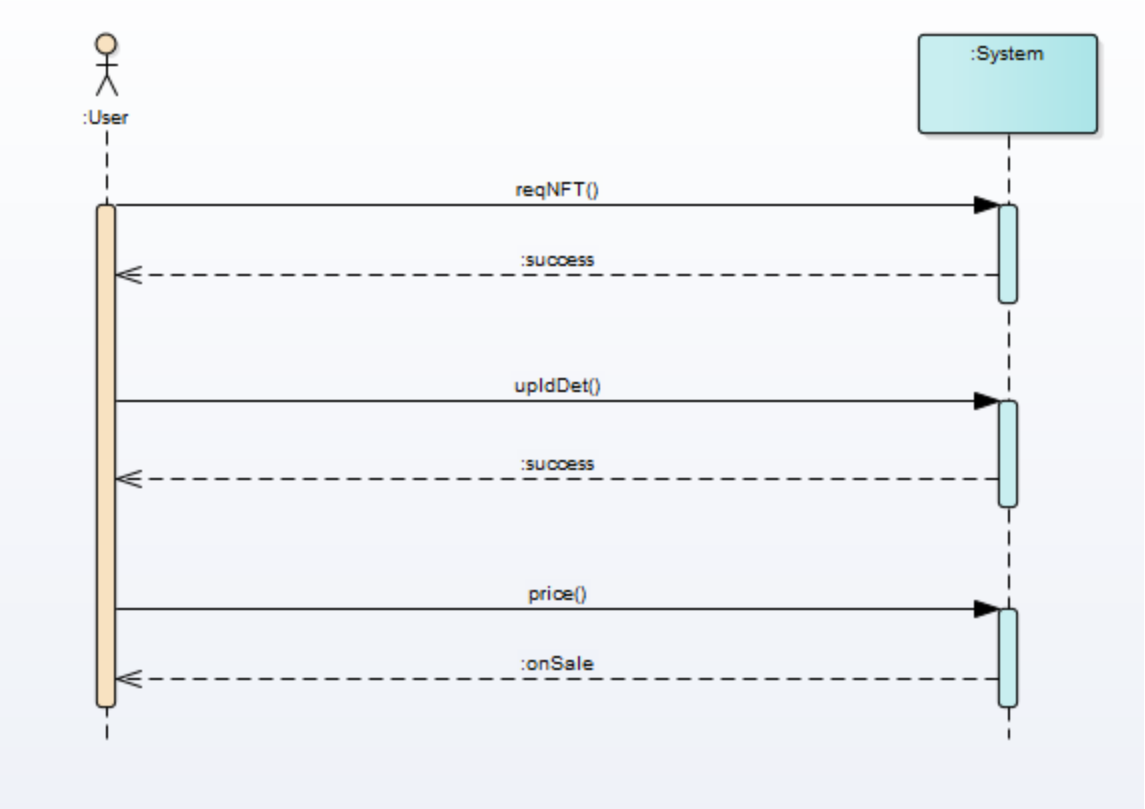
Fig#3.6

Buy NFT



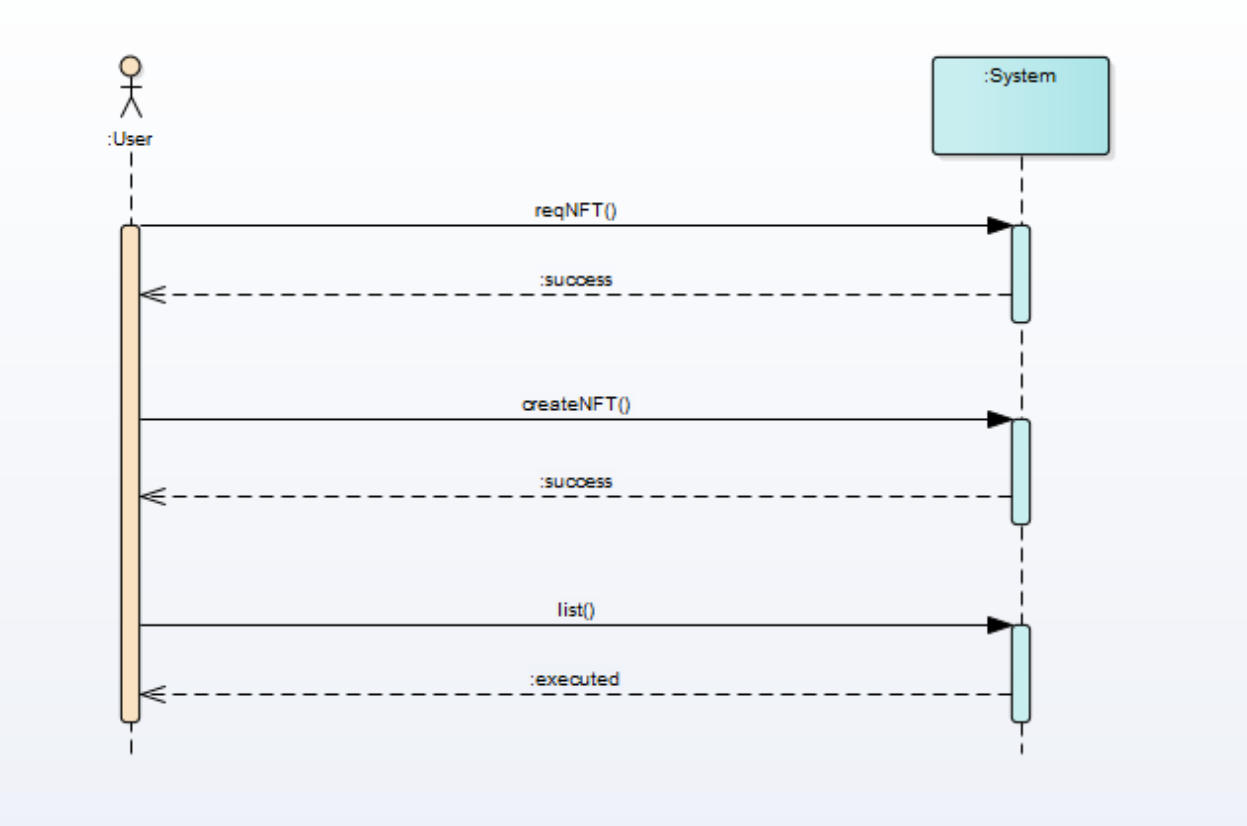
Fig#3.7

Sale NFT



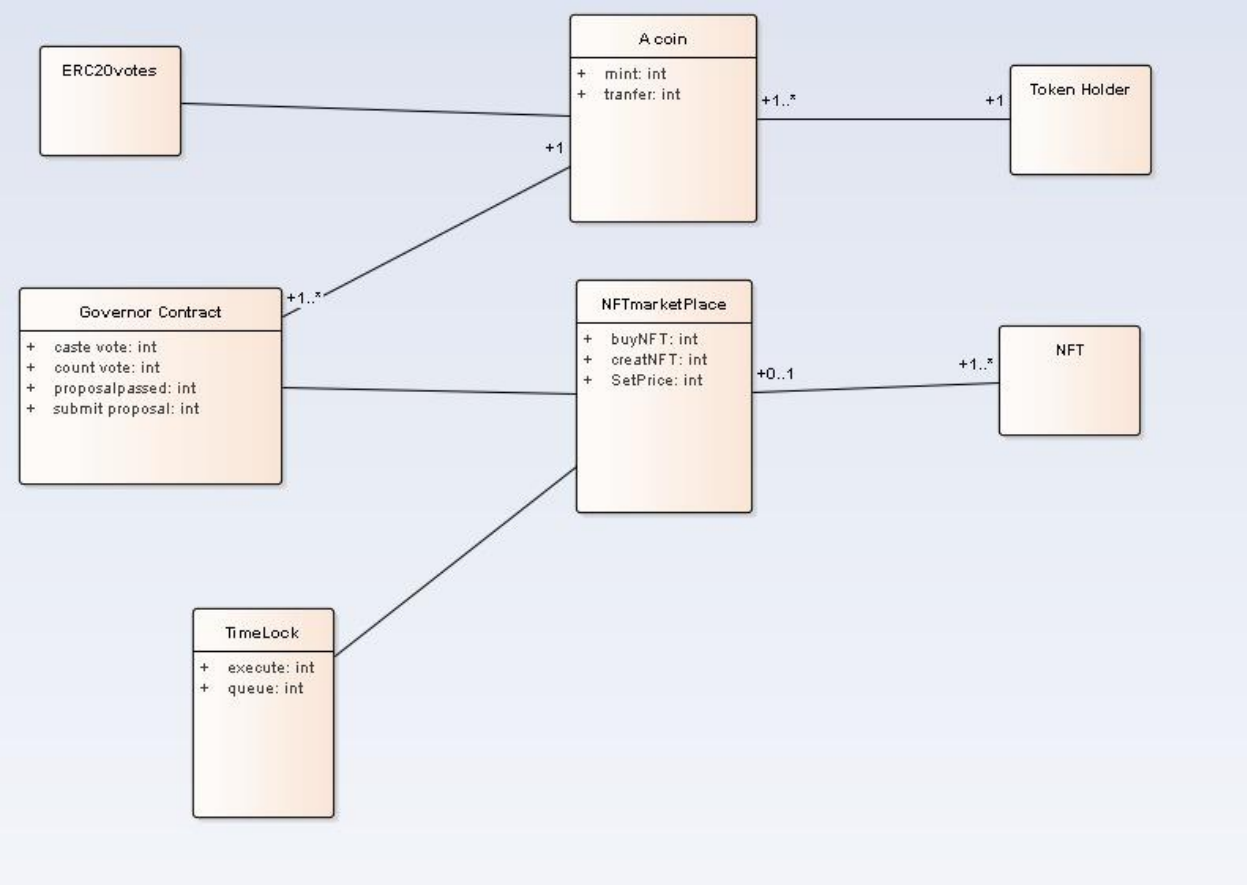
Fig#3.8

List NFT



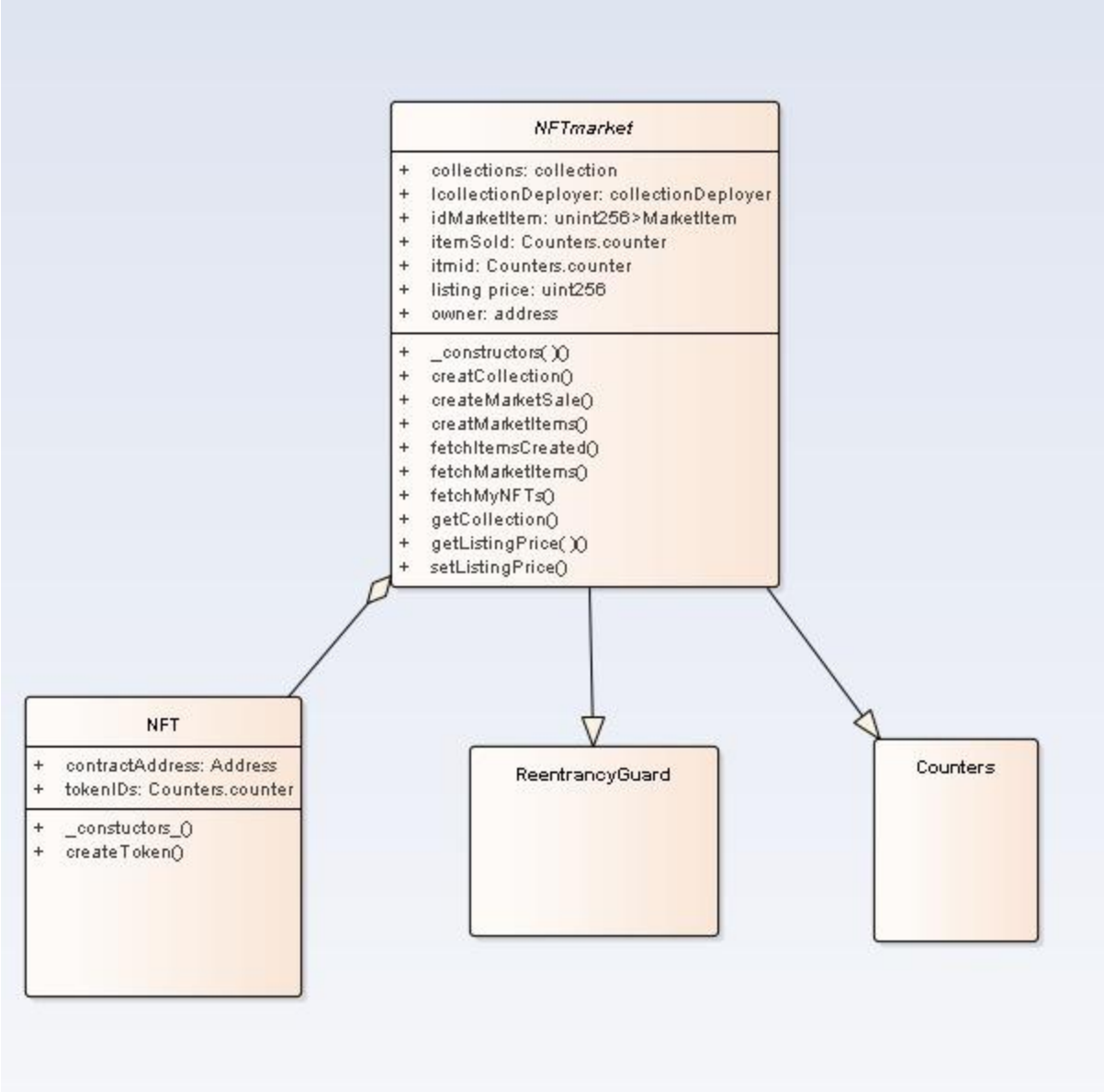
Fig#3.9

Domain model



Fig#3.9

Class Diagram



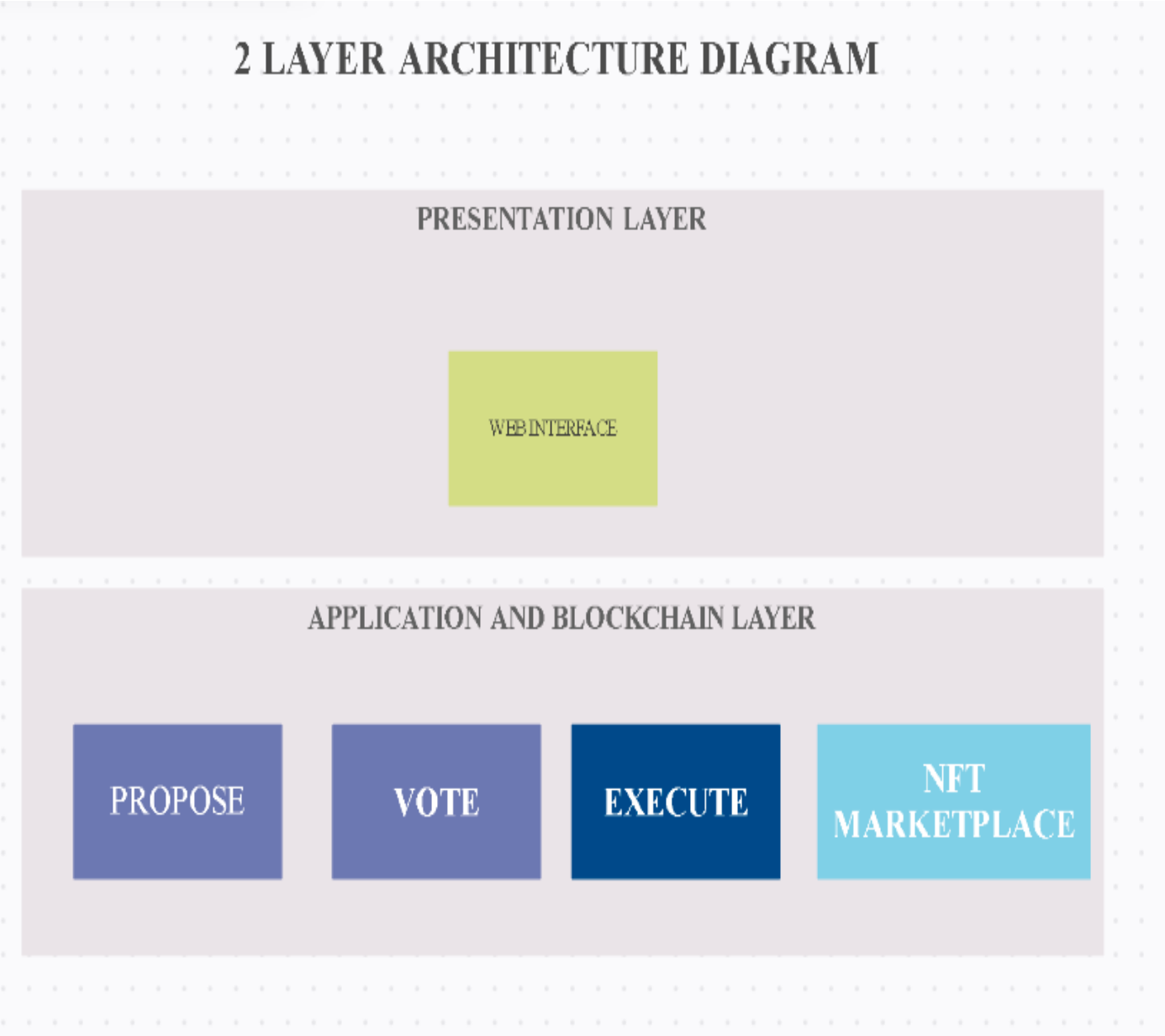
Fig#3.10

Software Architecture

Application architecture that organizes applications into two logical and physical computing tiers: the presentation tier, or user interface; the application with blockchain tier, where data is processed; and the, where the data associated with the application is stored and managed.

Presentation Tier

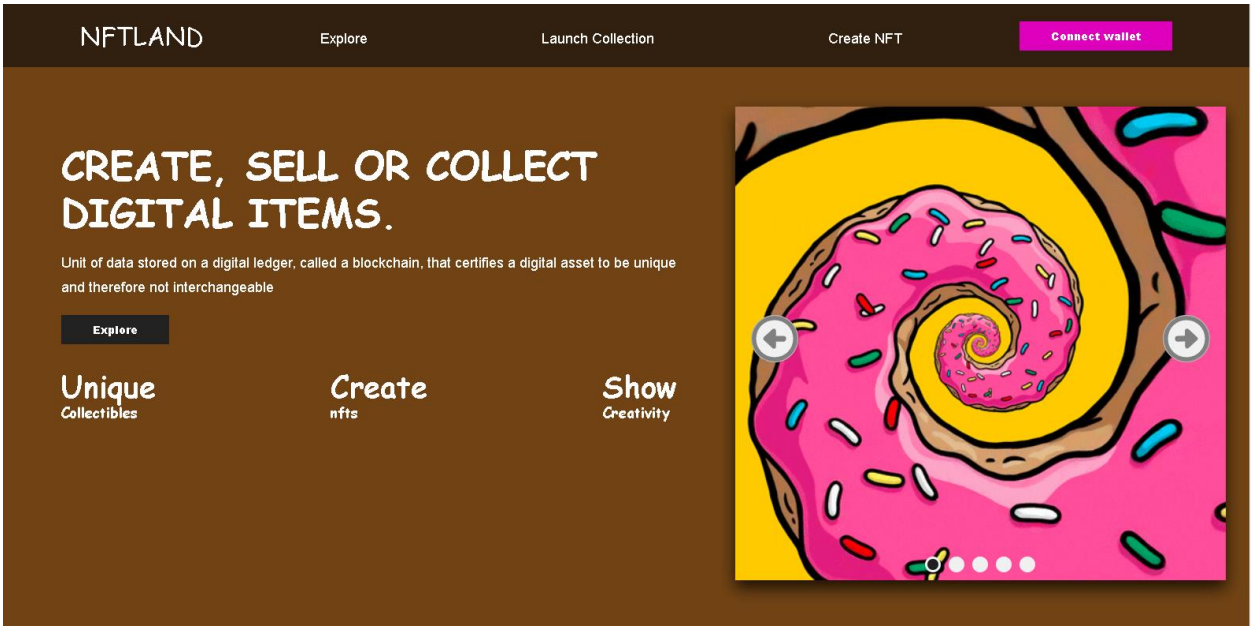
The presentation tier is the user interface and communication layer of the application, where the end user interacts with the application. Its main purpose is to display information to and collect information from the user. This top-level tier can run on a web browser, as desktop application, or a graphical user interface (GUI), for example. Web presentation tiers are usually developed using HTML, CSS and JavaScript. Desktop applications can be written in a variety of languages depending on the platform. Application tier and blockchain layer. The application tier, also known as the logic tier or middle tier, is the heart of the application. In this tier, information collected in the presentation tier is processed sometimes against other information in the data tier - using business logic, a specific set of business rules. The application tier can also add, delete or modify data in the data tier. The application tier is typically developed using Python, Java, Perl, PHP or Ruby, and communicates with the data tier using API calls. data access tier or back-end, is where the information processed by the application is stored and managed. This can be a blockchain such as Ethereum, Solana, BSC.


2 LAYER ARCHITECTURE DIAGRAM
PRESENTATION LAYER
WEB INTERFACE
APPLICATION AND BLOCKCHAIN LAYER
PROPOSE
VOTE
EXECUTE
NFT MARKETPLACE

Fig#3.11

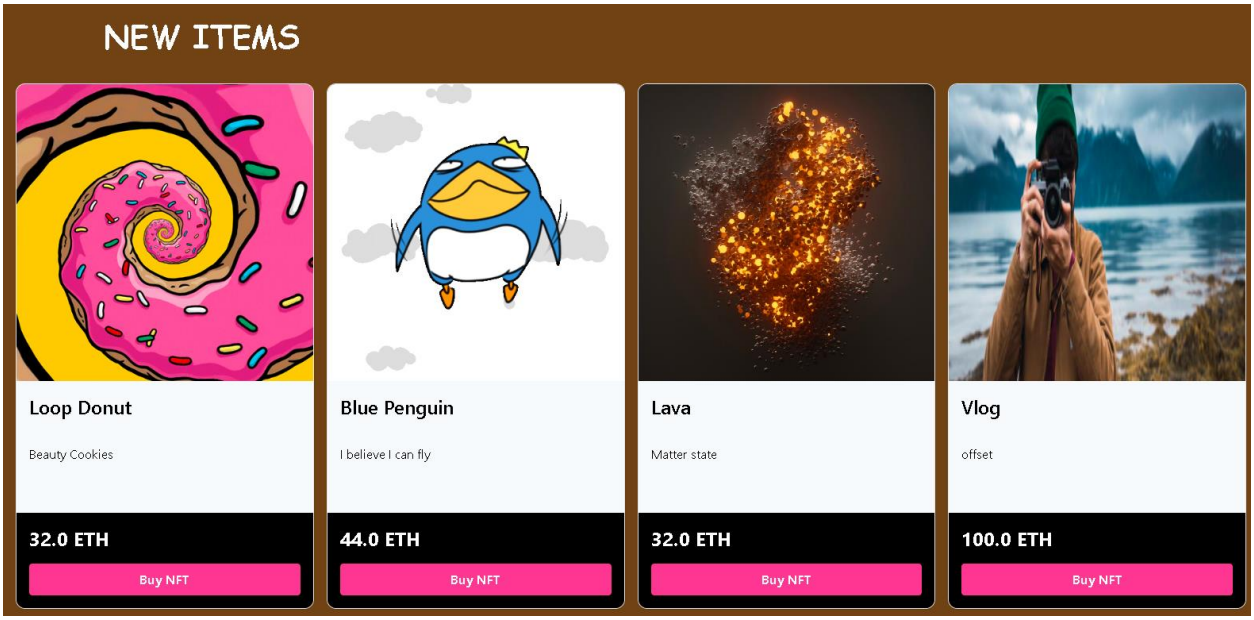
GUI

Dashboard



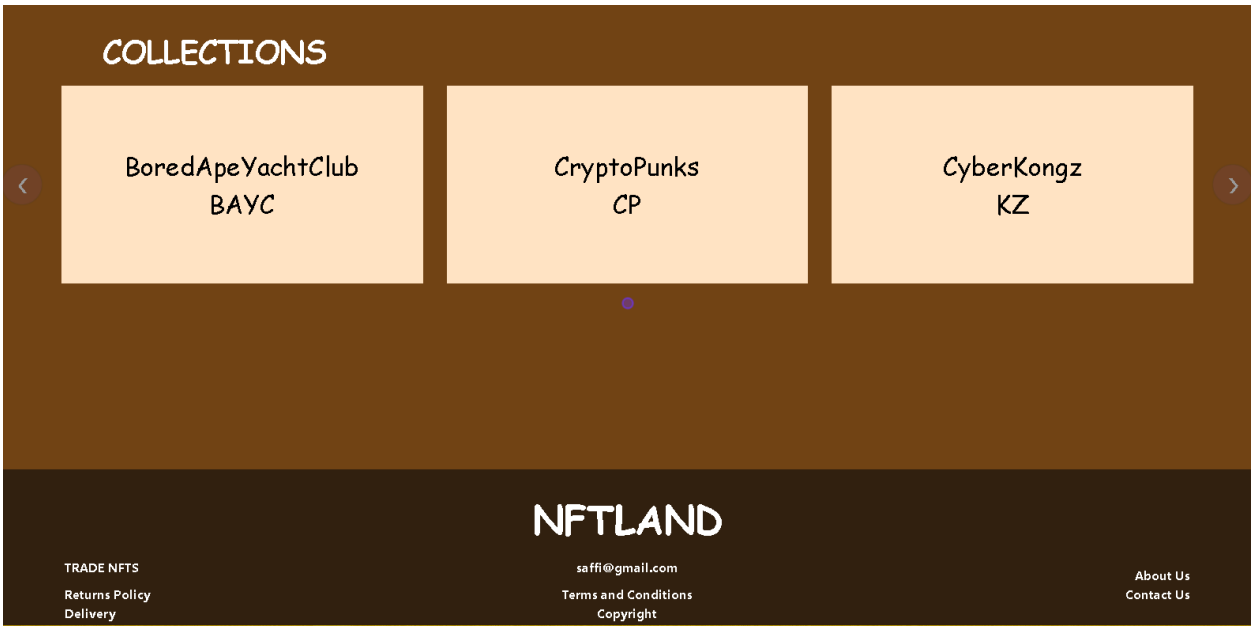
Fig#3.12

NFT New Items:



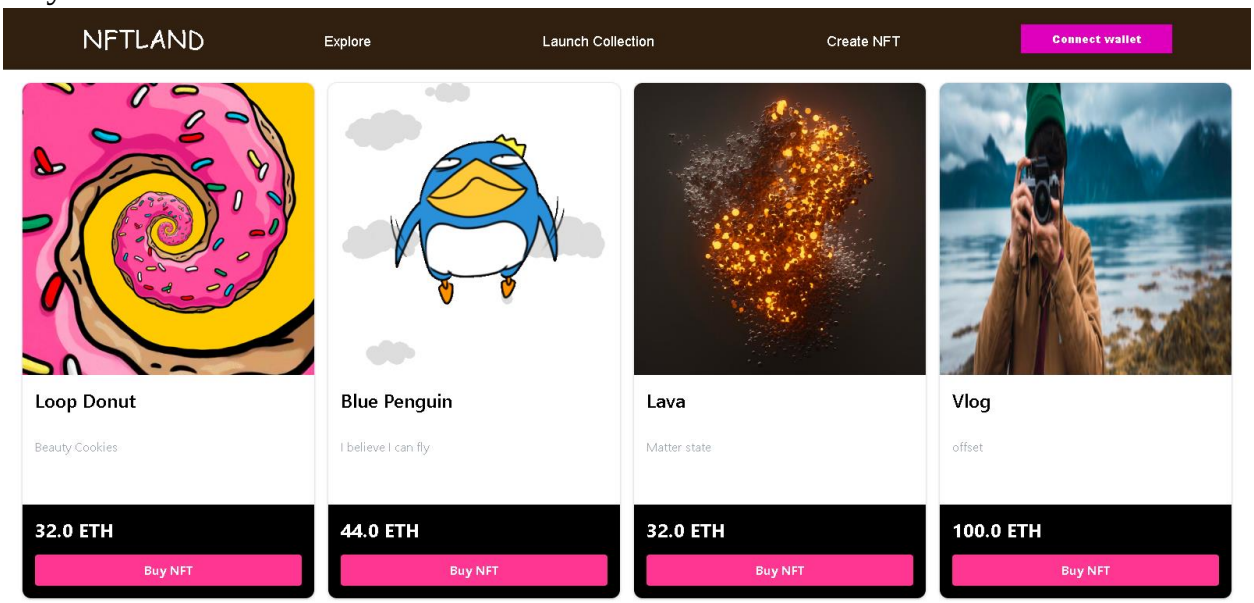
Fig#3.13

NFT Collection:

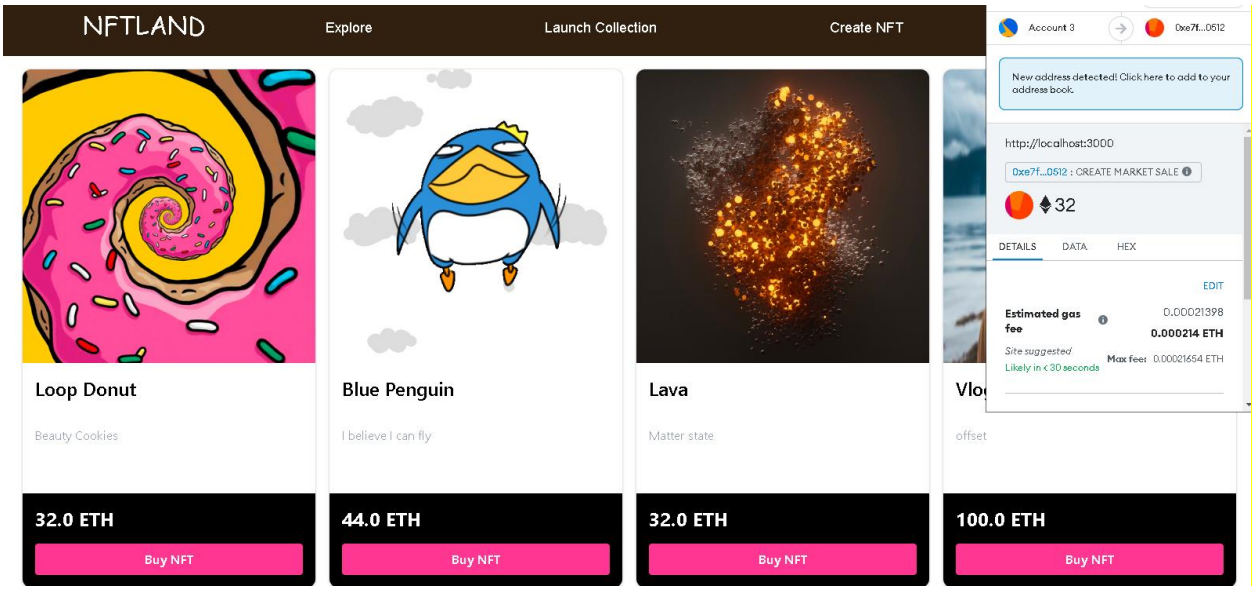


Fig#3.14

Buy NFT:

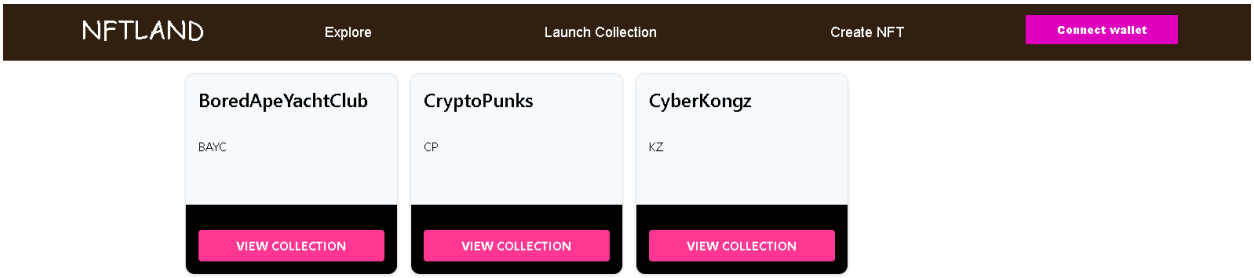


Fig#3.15

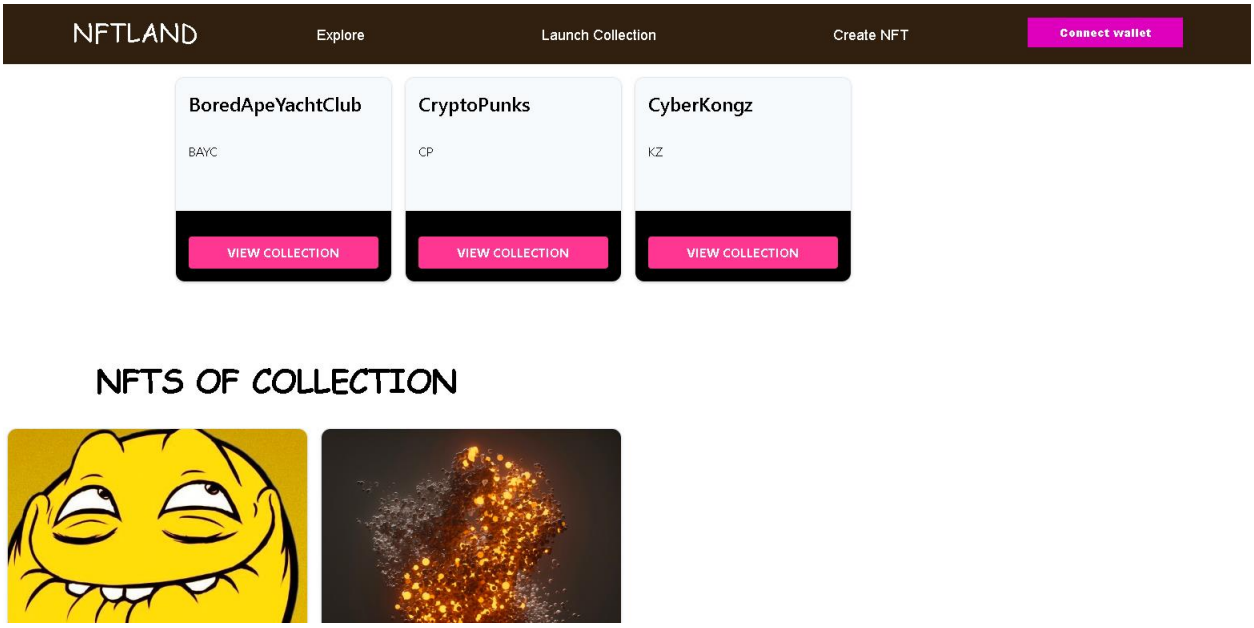


Fig#3.16

Collection:

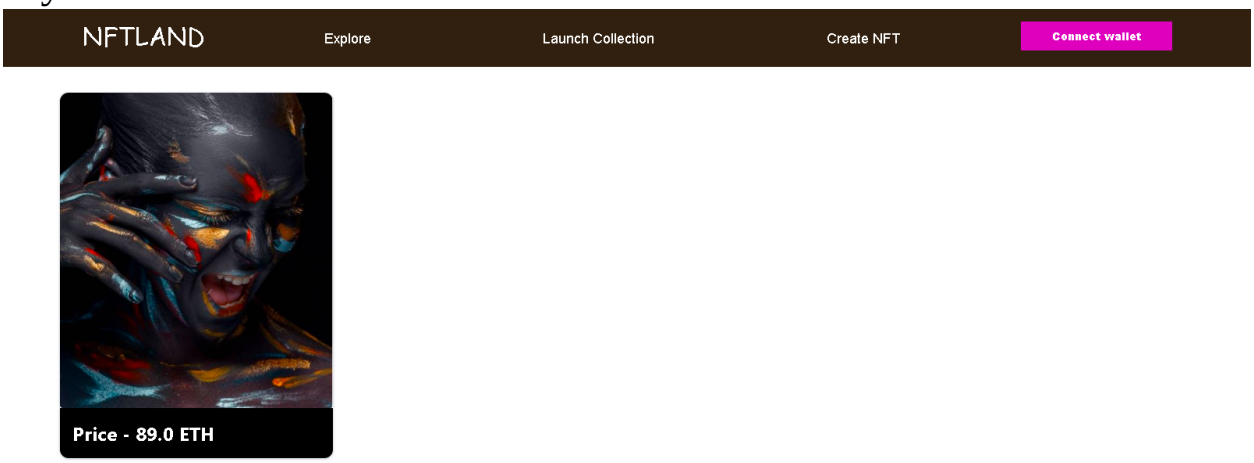


Fig#3.17



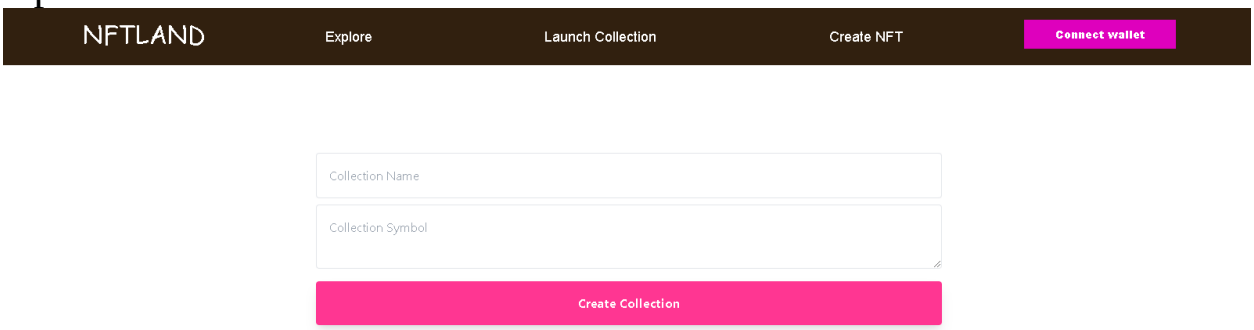
Fig#3.18

My NFT:

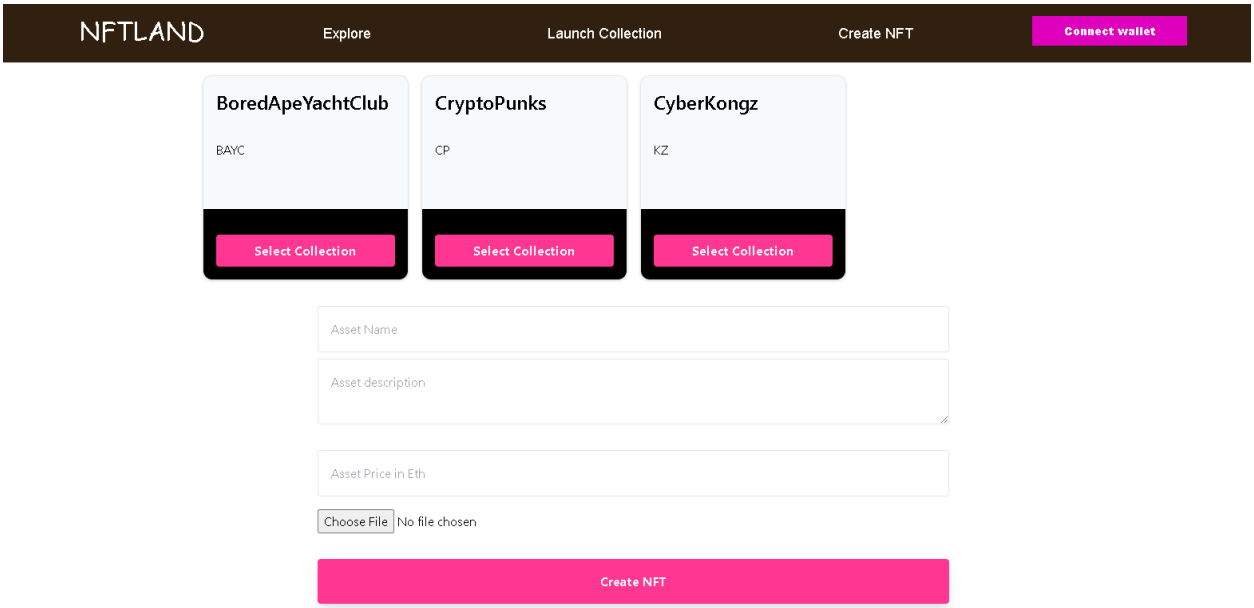


Fig#3.19

Update Collection:



Fig#3.20


NFTLAND
Explore
Launch Collection
Create NFT
Connect wallet
BoredApeYachtClub
BAYC
Select Collection
CryptoPunks
CP
Select Collection
CyberKongz
KZ
Select Collection
Asset Name
Asset description
Asset Price in Eth
Choose File
No file chosen
Create NFT

Fig#3.21

Chapter 4

Software Development

This chapter will provide the details about the coding standard, we adopted during implementation phase.

4.1. Coding Standards

We used the camel notation naming standards during the development of our system. For Example, to create the variable for admin name the syntax should be nft. Similarly for the method / functions the syntax is getNFT(). We mentioned the proper comments in code to make it understandable.

4.2. Development Environment

We implement our system in the following programming languages.

a. JSX

b. Javascript

c. Solidity

The tools we used for the development purpose are following.

a. VS code

b. Hardhat

c. npm

d. Nextjs

4.1.2. Declaration

One declaration per line is used to enhances the clarity of code. The order and position

of declaration is as follows:

• The static/class variables is placed in the sequence: First public class variables, protected.

• Package/default level i.e. with no access modifier and then the private. As far as possible static or class fields are explicitly instantiated.

• Instance variables are placed in the sequence: First public instance variables, protected.

• Package level with no access modifier and then private.

• Next the class constructors are declared.

• Class methods are grouped by functionality rather than by scope or accessibility to make reading and understanding the code easier.

• Declarations for local variables are only at the beginning of blocks e.g. at the beginning of a try/catch construct

4.1.3. Statement Standards

Each line contains at most one statement. While compound statements are statements that contain lists of statements enclosed in braces. The enclosed statements are indented one more level than the compound statement. The opening brace at the end of the line that begins the compound statement. The closing brace to begin a line and be indented to the beginning of the compound statement. Braces are used around all statements, even single statements, when they are part of a control structure, such as a if-else or for statement. A Boolean expression / function is compared to a Boolean constants.

4.1.4. Naming Convention

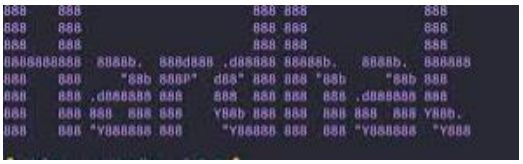
Naming conventions make programs more understandable by making them easier to read.

Following conventions are followed while naming a class or a member:

We used full English descriptors that accurately describe the variable, method or class. For example, use of names like tokenId, contractAddress instead of names like item Sold or itemIds. Terminology applicable to the domain is used. Implying that if user refers to clients as customers, then the term Customer is used for the class, not Client. Mixed case is used to make names readable with lower case letters in general capitalizing the first letter of class names and interface names.

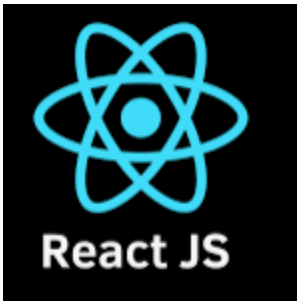
4.2. Development Environment

In following section, we will be giving brief description of the tools and technologies that we will be using for our project.



Hardhat

Hardhat provides development environment and testing tools for smart contract.



React JS

Nextjs will be used for client side and server-side rendering.

Ethereum

Ethereum virtual machine will be used as block chain

VS code

Visual Studio Code, also commonly referred to as VS Code, is a source-code editor made by Microsoft for Windows, Linux and macOS. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git.

4.3. Software Description

Main modules of our project are

- Create NFT
- Create custom Collection
- Buy NFT
- Sell NFT
- Create NFT inside custom collection

Create NFT

In this a new NFT would be created by the user

```
//1. create item (image/video) and upload to ipfs
async function createItem(){
  const {name, description, price} = formInput; //get the value from the form input

  //form validation
  if(!name || !description || !price || !fileUrl) {
    return
  }

  const data = JSON.stringify({
    name, description, image: fileUrl
```

```
    });

    try{
        const added = await client.add(data)
        //pass the url to sav eit on Polygon adter it has been uploaded to IPFS
        createSale(url)
    }catch(error){
        console.log(`Error uploading file: `, error)
    }
}

//2. List item for sale
async function createSale(url){
    const web3Modal = new Web3Modal();

    //get the tokenId from the transaction that occured above
    //there events array that is returned, the first item from that event
    //is the event, third item is the token id.
    console.log('Transaction: ',tx)
    console.log('Transaction events: ',tx.events[0])
    let event = tx.events[0]
    let value = event.args[2]
    let tokenId = value.toNumber() //we need to convert it a number

    //get a reference to the price entered in the form

    //get the listing price
    let listingPrice = await contract.getListingPrice()
    listingPrice = listingPrice.toString()

    transaction = await contract.createMarketItem(
        nftaddress, tokenId, price, {value: listingPrice }
    )

    await transaction.wait()
```

```
    router.push('/')
```

Create Custom Collection

In this a new custom Collection will be created

```
        return (
        <div className="flex justify-center" style={{marginTop:"5%"}} >
            <div className="w-1/2 flex flex-col pb-12">
                <input
                    placeholder="Collection Name"
                    className="mt-8 border rounded p-4"
                    onChange={e => updateFormInput({...formInput, name:
e.target.value})}
                    />
                <textarea
                     placeholder="Collection Symbol"
                     className="mt-2 border rounded p-4"
                     onChange={e => updateFormInput({...formInput, symbol:
e.target.value})}
                     />

                <button onClick={createItem}
                     className="font-bold mt-4 bg-pink-500 text-white rounded p-4
shadow-lg"
                     >Create Collection</button>
            </div>
        </div>
    )

}
```

• **Buy NFT**

In this we will buy NFT

```
 async function buyNFT(nft){
    const web3Modal = new Web3Modal();
    const connection = await web3Modal.connect();
    const provider = new
ethers.providers.Web3Provider(connection);

    //sign the transaction
    const signer = provider.getSigner();
    const contract = new ethers.Contract(nftmarketaddress,
Market.abi, signer);

    //set the price
    const price = ethers.utils.parseUnits(nft.price.toString(),
'ether');

    //make the sale
    const transaction = await
contract.createMarketSale(nftaddress, nft.tokenId, {
      value: price
    });
    await transaction.wait();

    loadNFTs()
  }

  if(loadingState === 'loaded' && nfts.length===0) return (
    <h1 className="px-20 py-10 text-3xl">No items in market
place</h1>
  )
```

```
  return (
   <div className="flex justify-center">
     <div className="px-4" style={{ maxWidth: '1600px'}}>
      <div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-
cols-4 gap-4 pt-4">
        {
          nfts.map((nft, i) =>(
            <div key={i} className="border shadow rounded-xl
overflow-hidden">

              <img
                  src={nft.image}
                  alt="Picture of the author"
style={{height:"374px",width:"374px"}}
                  width="500"
                   height="500"
                  // blurDataURL="data:..." automatically
provided
                  // placeholder="blur" // Optional blur-up
while loading
                />
                       <div className="p-4">
                <p style={{ height: '64px'}} className="text-
2xl font-semibold">
                  {nft.name}
                </p>
                <div style={{ height: '70px', overflow:
'hidden'}}>
                  <p className="text-gray-
400">{nft.description}</p>
                </div>
              </div>
              <div className="p-4 bg-black">
                <p className="text-2xl mb-4 font-bold text-
white">
                  {nft.price} ETH
```

```
                </p>
                <button className="w-full bg-pink-500 text-
white font-bold py-2 px-12 rounded"
                onClick={() => buyNFT(nft)}>Buy NFT</button>
            </div>
            </div>
          ))
        }
      </div>
    </div>
  </div>
  )
}
```

- **Sell NFT**

In this part of code we will be able to sell NFT

```
return (
  <div className="flex justify-center" style={{marginTop:"5%"}}>
    <div className="w-1/2 flex flex-col pb-12">
      <input
        placeholder="Asset Name"
        className="mt-8 border rounded p-4"
        onChange={e => updateFormInput({...formInput, name: e.target.value})}
        />
      <textarea
        placeholder="Asset description"
        className="mt-2 border rounded p-4"
        onChange={e => updateFormInput({...formInput, description: e.target.value})}
```

```
            />
          <input
            placeholder="Asset Price in Eth"
            className="mt-8 border rounded p-4"
            type="number"
            onChange={e => updateFormInput({...formInput, price: e.target.value})}
            />
            <input
              type="file"
              name="Asset"
              className="my-4"
              onChange={onChange}
            />
            {
              fileUrl && (
                <img
                src={fileUrl}
                alt="Picture of the author"
                className="rounded mt-4"
                width="350"
                />
              )
            }
            <button onClick={createItem}
             className="font-bold mt-4 bg-pink-500 text-white rounded p-4 shadow-lg"
             >Create NFT</button>
        </div>
      </div>
  )
}
```

- **Create NFT inside custom collection**

  In this we will Create NFT inside custom collection.

```
         <div className="flex justify-center">
     <div className="px-4" style={{ maxWidth: '1600px'}}>
      <div className="grid grid-cols-1 sm:grid-cols-2 lg:grid-
cols-4 gap-4 pt-4">
        {
          collection.map((nft, i) =>(
            <div key={i} className="border shadow rounded-xl
overflow-hidden">


                        <div className="p-4 bg-slate-50 text-
black">
                <p style={{ height: '64px'}} className="text-
2xl font-semibold">
                  {nft.name}
                </p>
                <div style={{ height: '70px', overflow:
'hidden'}}>
                  <p className="text-white-
400">{nft.symbol}</p>
                </div>
              </div>
              <div className="p-4 bg-black">
                <p className="text-2xl mb-4 font-bold text-
white">

                </p>
                <button className="w-full bg-pink-500 text-
white font-bold py-2 px-12 rounded"
```

```
                onClick={()=>updateSpecificCollection(nft.addr)
}
                >Select Collection</button>
            </div>
            </div>
          ))
        }
      </div>
    </div>
  </div>
```

Chapter 5

Software Testing

This chapter provides a description about the adopted testing procedure. This includes the selected testing methodology, test suite and the test results of the developed software.

Create NFT

Date: 18 July 2022	
System: NFT Land	
Objective: Create NFT	Test ID: 1
Version: 1	Test Type: Unit testing
Input: Image Name Description Price	
Expected Result: User create NFT successfully.	
Actual Result: Created Successfully.	

Table#5.1

Create Custom Collection

Date: 19 July 2022	
System: NFT Land	
Objective: Create Custom Collection	Test ID: 2
Version: 2	Test Type: Unit testing
Input: Name Symbol	
Expected Result: User create custom collection Successfully.	
Actual Result: Custom Collection created	

Table#5.2

Buy NFT

Date: 20 July 2022	
System: NFT Land	
Objective: Buy NFT successfully	Test ID: 3
Version: 3	Test Type: Unit testing
Input: Click buy button	
Expected Result: Buy NFT Successfully.	

Actual Result: NFT bought created

Table#5.3

Sell NFT

Date: 21 July 2022	
System: NFT Land	
Objective: Sell NFT successfully	Test ID: 4
Version: 4	Test Type: Unit testing
Input: Buy NFT clicked	
Expected Result: NFT sold Successfully.	
Actual Result: NFT sold	

Table#5.13

Create NFT inside Custom Collection

Date: 23 July 2022	
System: NFT Land	
Objective: Create NFT inside Custom Collection	Test ID: 5
Version: 5	Test Type: Unit testing
Input: Image Name	

Description
Expected Result: User NFT created in custom collection Successfully.
Actual Result: NFT created in Custom Collection.

CHAPTER 6

SOFTWARE DEPLOYMENT

Deployed Smart Contract Addressess on BSC testnet

CollectionDeployer = "0xbD7b620cA4cA8acE9D63092022C787d3Cb56a894"

nftaddress = "0x7353f711B296188386Ffa08a83e800417D9636b0"

nftmarketaddress = "0xe8ACd60bc58ff97B7202b89090c739DD714f99Fc"

Installation Process Description

Frontend React App Installation Process:

1. Extract the zip in the selected folder.
2. Open the folder in Vs Code.
3. Run "npm install" in the terminal to install all the necessary dependencies for the project.

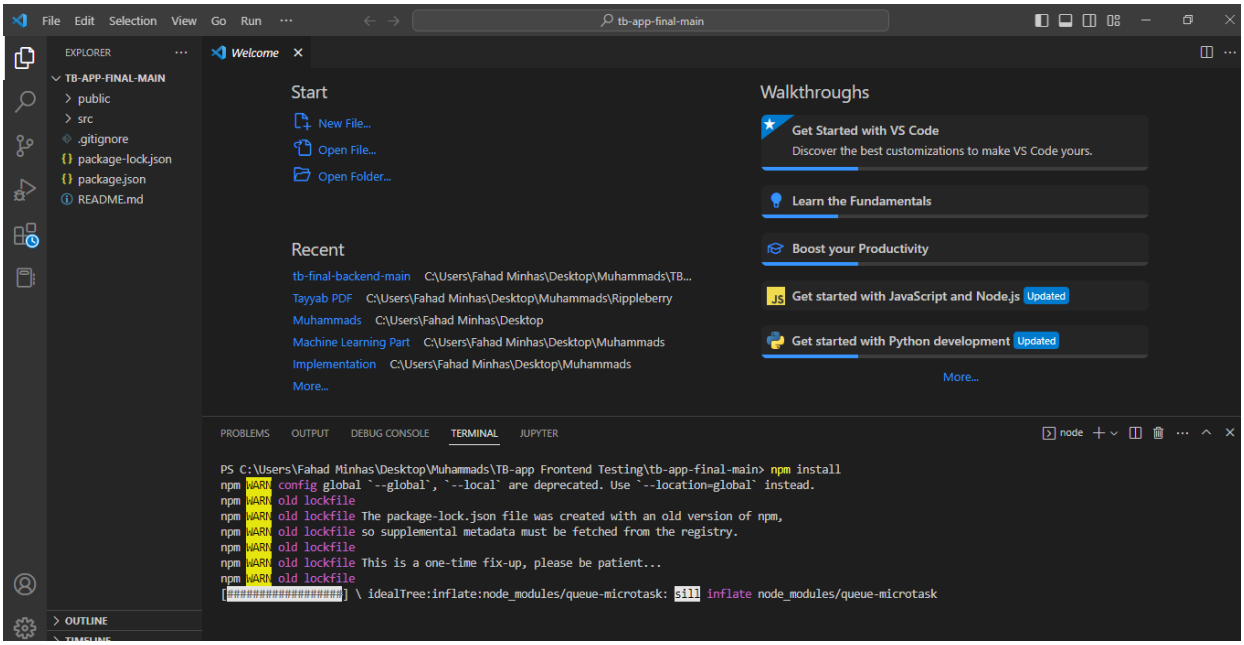


Figure 0-1: Installing Dependencies

4. After the dependencies are installed, you can start the project by using the command "npm start"

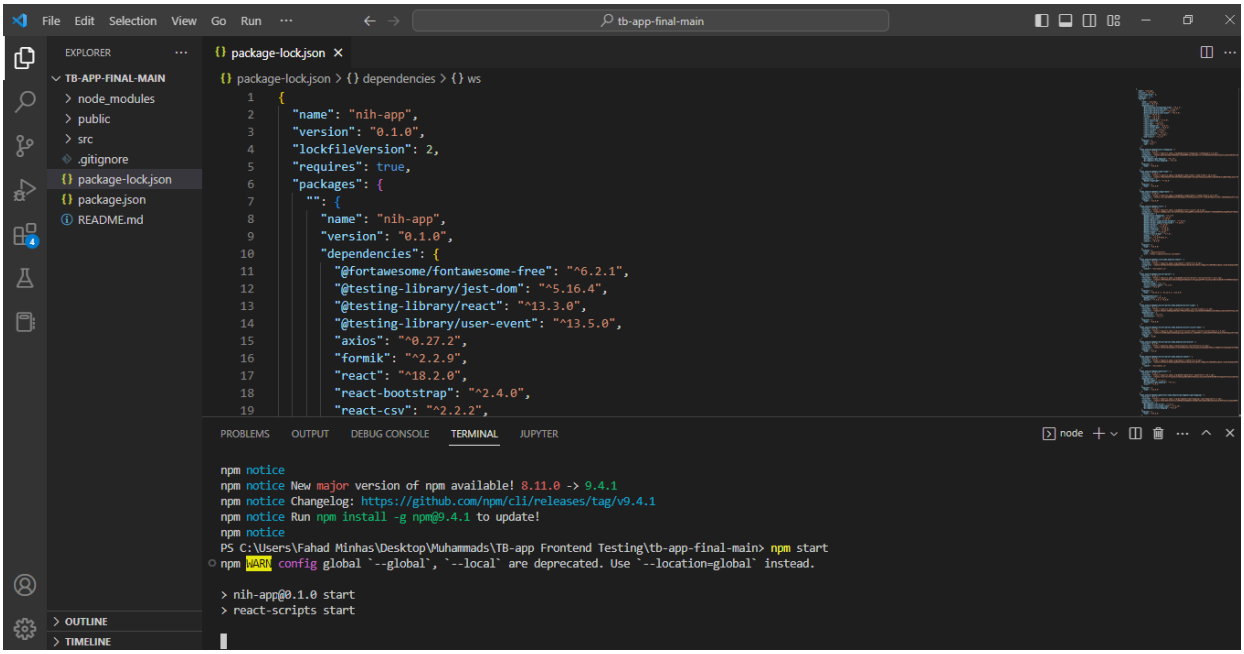


Figure 0-2:Starting React Application Screenshot

5. The React app should now be running on a local development server and can be accessed in a web browser at "http://localhost:3000".

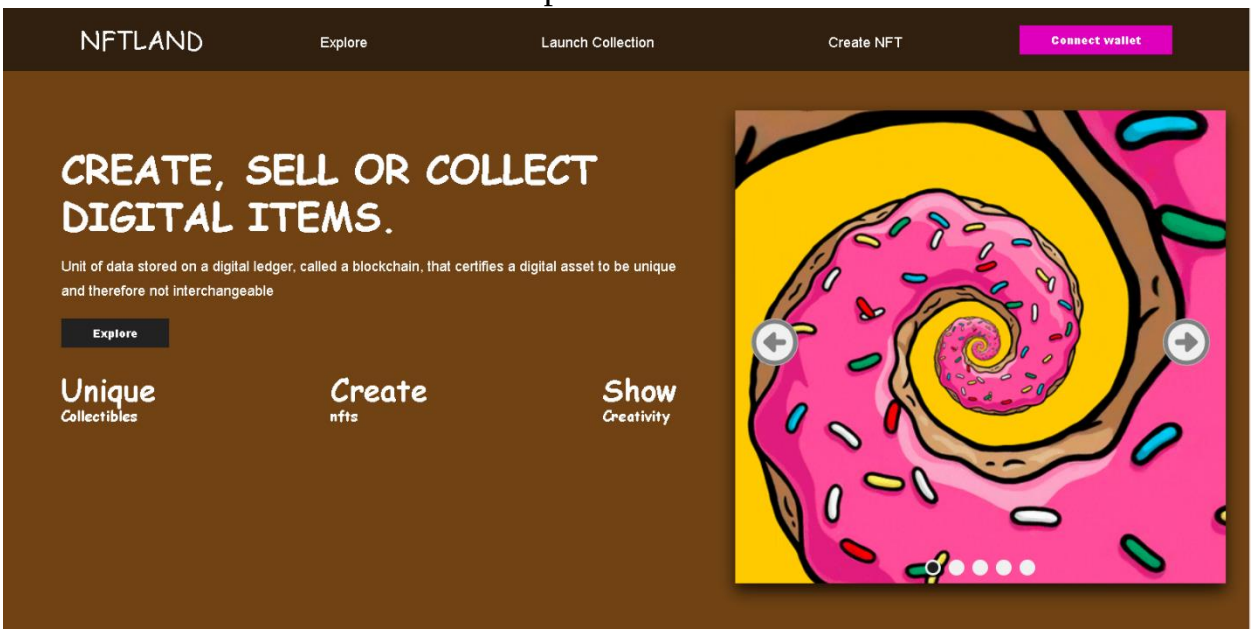


Figure 0-3:SuperSwap Interface

Backend Installation Process:

Backend can be installed in the same fashion as frontend.

Plan For Upcoming Iterations

1.Bbuild basic DAO blockchain part.

create cryptocurrency to so that we can use them as weight for voting.

Create smart contracts that will have all the logic for proposal submission, voting on proposal, and execution of proposal.

The smart contracts will include

- Governor Contract to handle all the voting logic.
- Governance Token which is cryptocurrency.
- Time lock to handle execution.

2. Build Front end for DAO.

Build the front-end for interacting with DAO.