Assignment 2 - Group 1
Backend Development for Interactive Frontend Design

COS30049 - Computing Technology Innovation Project

Nguyen Ngoc Thanh Thanh, Tran Manh Son and Nguyen Dinh Dung

1. Project Background and Introduction

1.1 Introduction

In today's dynamic digital economy, the rise of decentralized technologies is transforming conventional trading systems. Decentralized trading platforms leverage blockchain technology to facilitate secure, transparent, and trustless transactions between peers, eliminating the need for traditional centralized intermediaries. This paradigm shift results in lower transaction fees, reduced processing times, and enhanced user autonomy. The Pixie NFT Marketplace project is designed to harness these benefits by providing a comprehensive platform for creating, buying, selling, and bidding on Non-Fungible Tokens (NFTs).

The project integrates a modern front-end—developed with ReactJS—with a robust back-end system built using FastAPI. The backend manages dynamic data interactions through a relational database and incorporates smart contract functionality using Solidity to ensure that critical transactions are recorded immutably on a blockchain. By combining these elements, the system offers a user-centric trading environment that supports real-time updates and scalable operations.

1.2 Project Background

Traditional financial systems have historically relied on centralized authorities, which often result in inefficiencies such as delayed transaction processing and higher associated costs. With the advent of blockchain technology, pioneered by Bitcoin (Nakamoto, 2008), a new approach has emerged. Blockchain's decentralized ledger provides a secure and transparent framework for recording transactions, addressing many of the limitations inherent in conventional systems.

Recent studies have demonstrated that blockchain technology can fundamentally change business models by reducing operational costs and enhancing transparency (Weking, 2019; Wang et al., 2022). The Pixie NFT Marketplace project builds on this foundation by using smart contracts to automate transactions and enforce predetermined conditions, ensuring that asset exchanges are both secure and efficient. This integration of blockchain with a dynamic web application not only improves system reliability but also fosters a more inclusive digital trading environment.

1.3 Motivation

The motivation for developing the Pixie NFT Marketplace is rooted in the need for more transparent, efficient, and user-empowered trading platforms. Decentralized systems promote transparency because every transaction is recorded on a public ledger, making it easier to verify and audit operations (Harvey, Hasbrouck, and Saleh, 2024). This level of transparency significantly reduces the risk of fraud and builds greater trust among users.

Furthermore, by removing intermediaries, decentralized platforms reduce transaction costs and processing delays, which is particularly beneficial in the context of digital asset trading where speed and efficiency are critical. The combination of a responsive front-end and a dynamic back-end enables real-time data updates and seamless interactions, ensuring that the marketplace remains competitive and user-friendly.

The integration of blockchain via smart contracts further enhances the security and reliability of the system, ensuring that once a transaction is validated, it is permanently recorded and cannot be altered. These factors collectively inspire the development of the Pixie NFT Marketplace, aimed at delivering a robust, scalable, and user-friendly platform that meets the evolving needs of digital asset trading.

2. Team Introduction

The Pixie NFT Marketplace project is a collaborative effort by Group 1, where each team member has contributed equally to all aspects of the project. While individual roles were assigned based on personal expertise, the team maintained close collaboration throughout the project to ensure that all tasks—from system architecture and backend development to frontend design and integration—were evenly shared.

Nguyen Dinh Dung (104772138):
Nguyen Dinh Dung played a key role in the overall system design and back-end development. He contributed significantly to the development of the relational database schema, the implementation of API endpoints using FastAPI, and the integration of smart contract functionalities. Alongside these technical tasks, he participated in design discussions and testing processes, ensuring that all backend components functioned seamlessly with the frontend.

Nguyen Ngoc Thanh Thanh (104512172):
Nguyen Ngoc Thanh Thanh was instrumental in leading the front-end development and user interface design. She focused on creating an intuitive and responsive UI using ReactJS, while also ensuring effective integration with backend APIs. In addition to his primary responsibilities, Thanh Thanh collaborated closely with the rest of the team during the system design and testing phases, contributing to overall project decisions and refinements.

Tran Manh Son (104191018):
Tran Manh Son served as the bridge between the front-end and back-end systems, ensuring smooth integration and comprehensive testing across all components. His contributions spanned API development support, debugging, and the deployment process. Son’s collaborative approach meant that he was equally involved in both the technical execution and the strategic planning phases of the project, ensuring balanced participation in all areas.

Together, the team ensured that work was distributed evenly across all project stages, from initial planning and design to final testing and deployment. This balanced approach not only

enhanced the overall quality of the project but also fostered a strong sense of teamwork and shared responsibility.

3. Project Requirement List and Description

3.1 Project Description

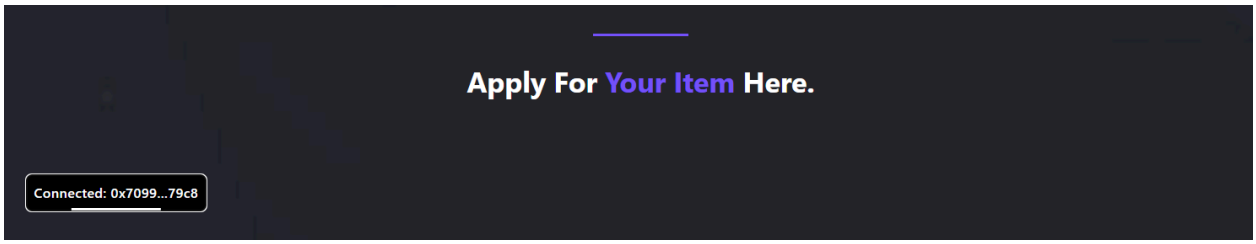
The Pixie NFT Marketplace is a decentralized trading system designed to facilitate the creation, buying, selling, and bidding of Non-Fungible Tokens (NFTs). The platform leverages blockchain technology to enable peer-to-peer transactions without the need for intermediaries, ensuring that all transactions are secure, transparent, and immutable. By integrating a modern ReactJS-based front-end with a robust FastAPI back-end and smart contract functionality written in Solidity, the marketplace provides a comprehensive environment for digital asset trading. This project not only addresses the inefficiencies of traditional centralized systems but also empowers users through enhanced transparency and reduced transaction costs.

3.2 Functional Requirements

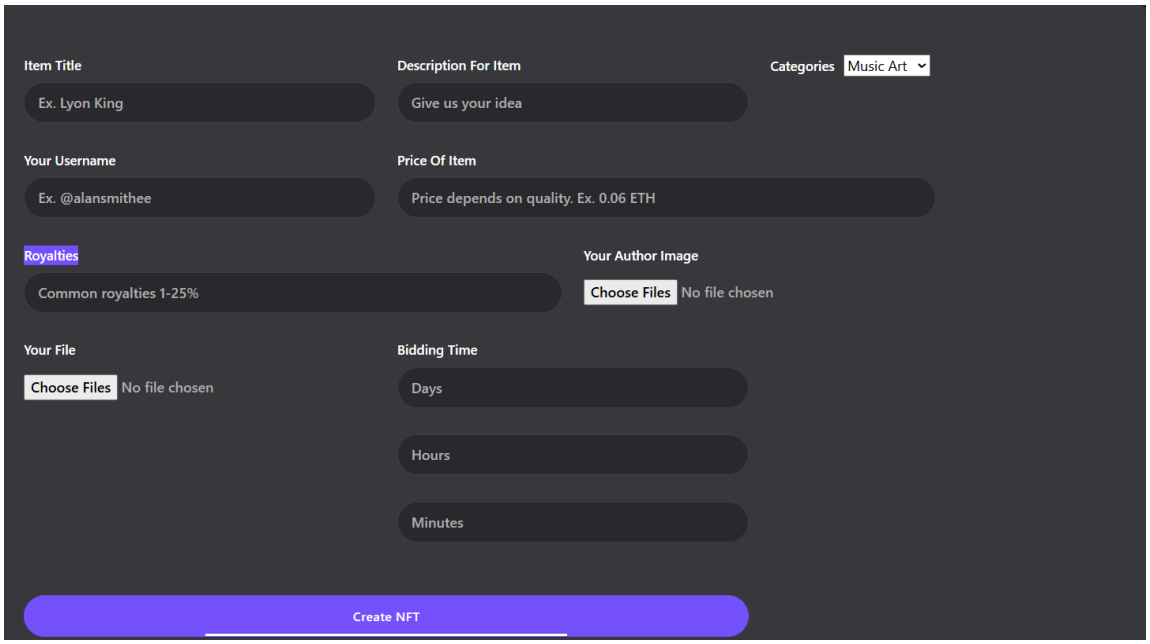
The system is designed to support a range of operations essential for a dynamic digital marketplace. The core functional requirements include:

Create Your Page

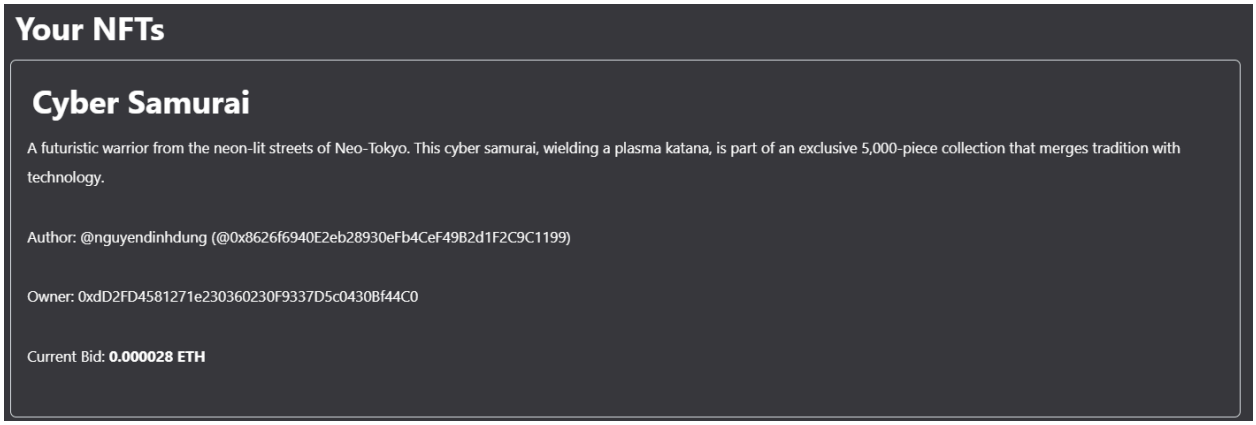
- **MetaMask Wallet Connection**: Connect wallet, fetch, and display NFTs



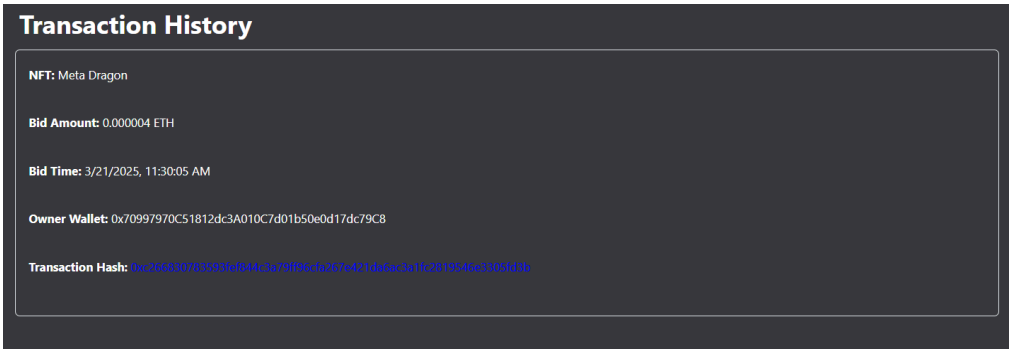
- **NFT Creation Form**: Allow users to input necessary details (title, description, username, price, RoRoyaltie, author image, item image, bidding time).



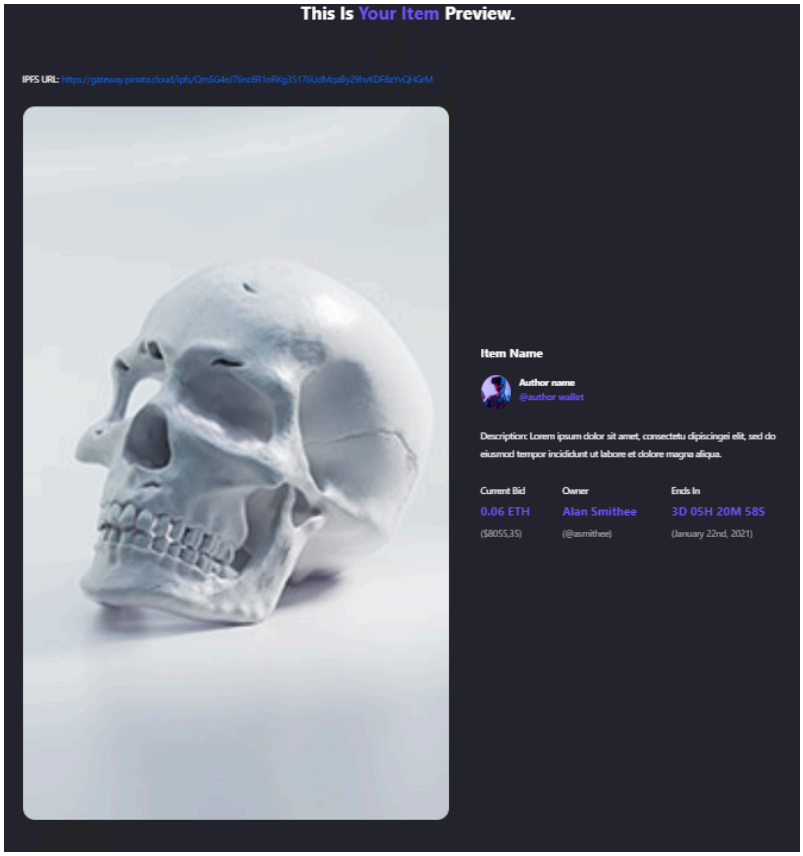
- **Your NFTs:** Fetch and display the NFTs of the connected wallet.



- **Transaction History:** Fetch and display the transaction history for the connected wallet.

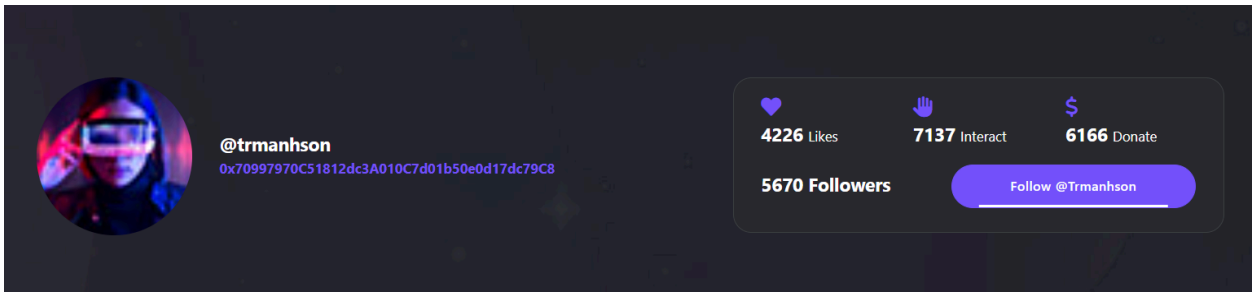


- **Preview of NFT**: Show a preview of the uploaded NFT information.

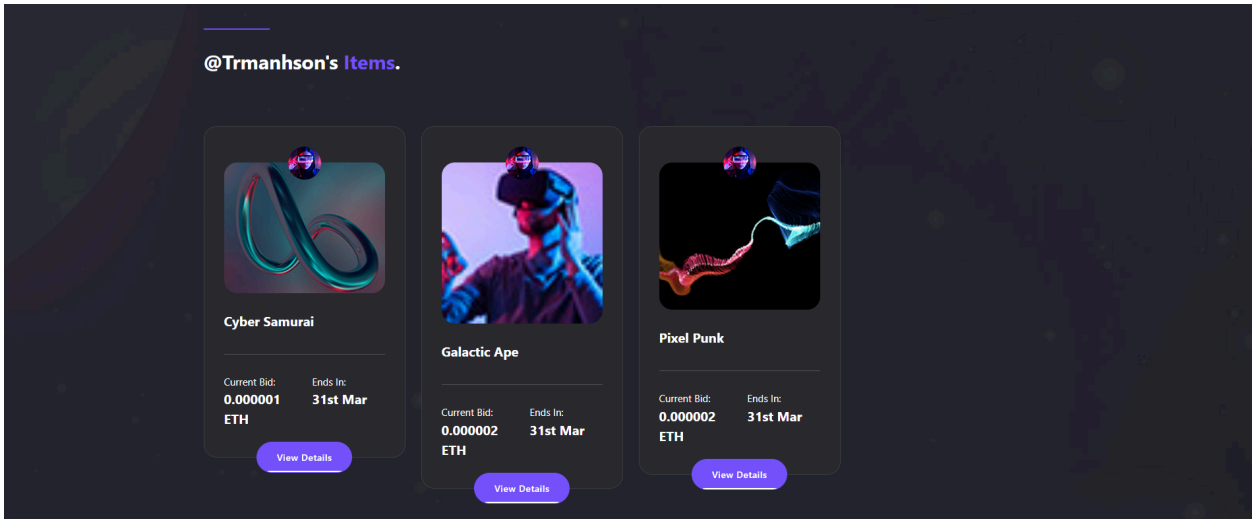


Author Page

- **Fetch Authors Data:** Fetch data of authors and their NFTs from the backend API, display a loading message while fetching data, display an error message if data fetching fails.
- **Display Author Information:** Display the author's image, name, wallet address, number of likes, interactions, donations, and followers. Provide a link to the author's Instagram profile.

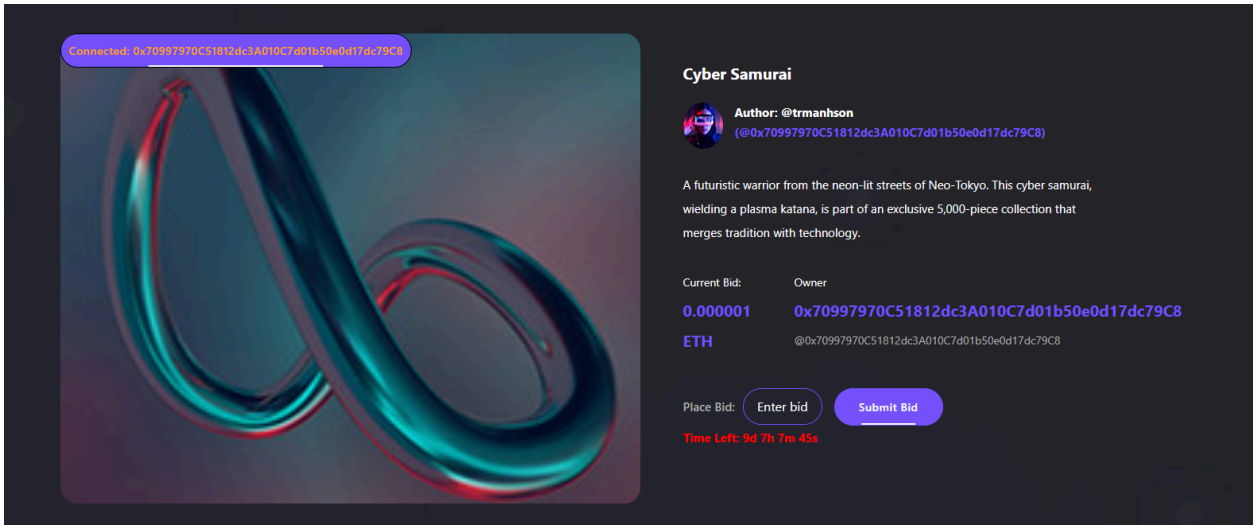


- **Display Author's NFTs:** Display a list of NFTs owned by the author. For each NFT, show the item name, current bid, currency, and the time left before the auction ends. Provide a "View Details" button for each NFT.

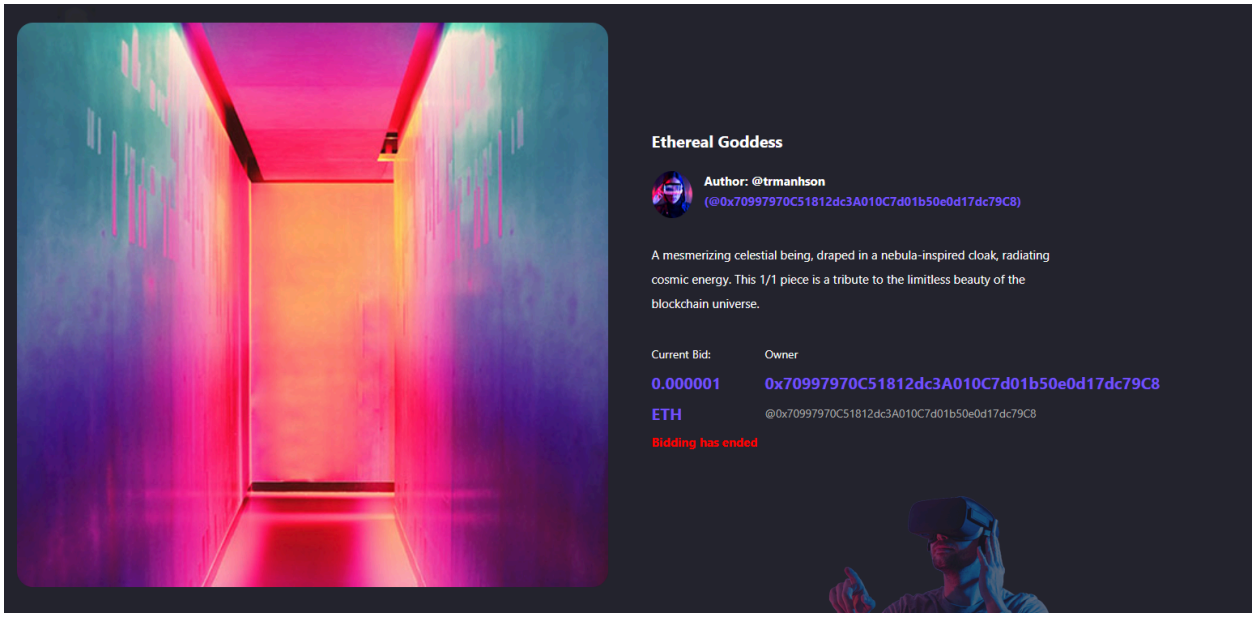


Item Details Page

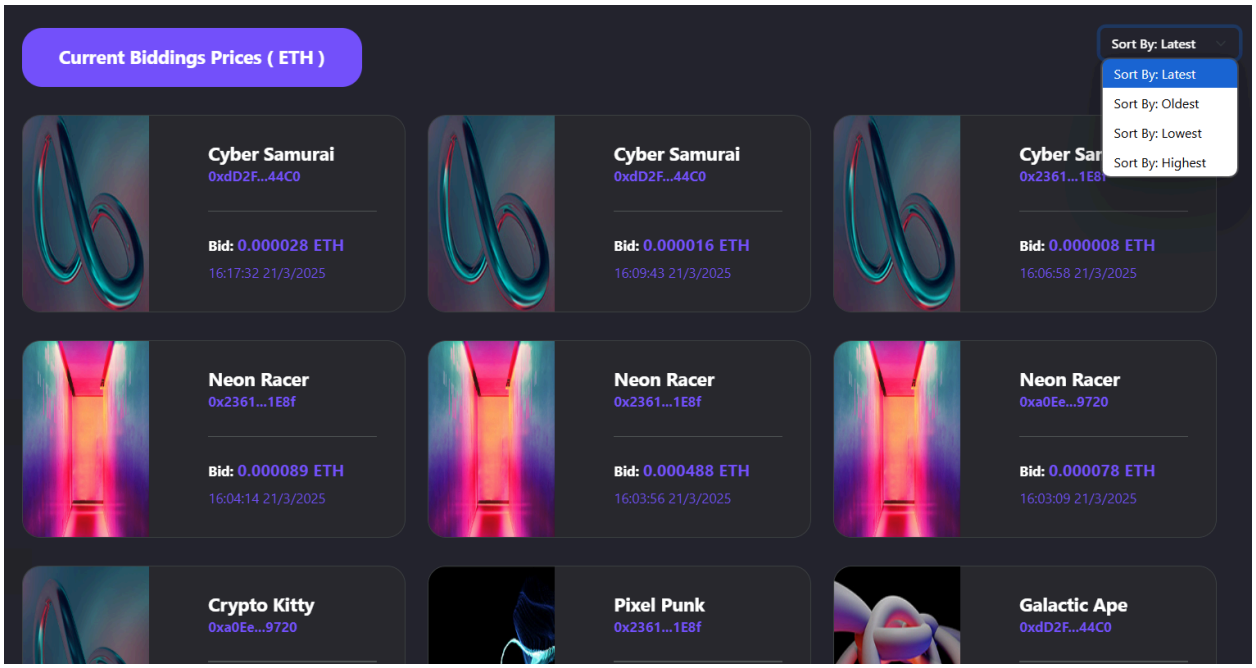
- **Item Details Section:** Display the item's image, name, and description, along with the author's name and wallet address. It should show the current bid in ETH, the owner's wallet address, and the remaining time for the auction. Users should be able to place a bid by entering the amount in an input field and submitting it. Additionally, there should be a "Submit Bid" button for finalizing the bid. And when the transaction has been confirmed by the user on-chain, the ownership and current price will be updated immediately.



- **Bidding end:**The status shows "Bidding has ended," indicating that the auction period for this NFT has concluded and the place bid field is removed and locked to preserve the current result of the highest bidder

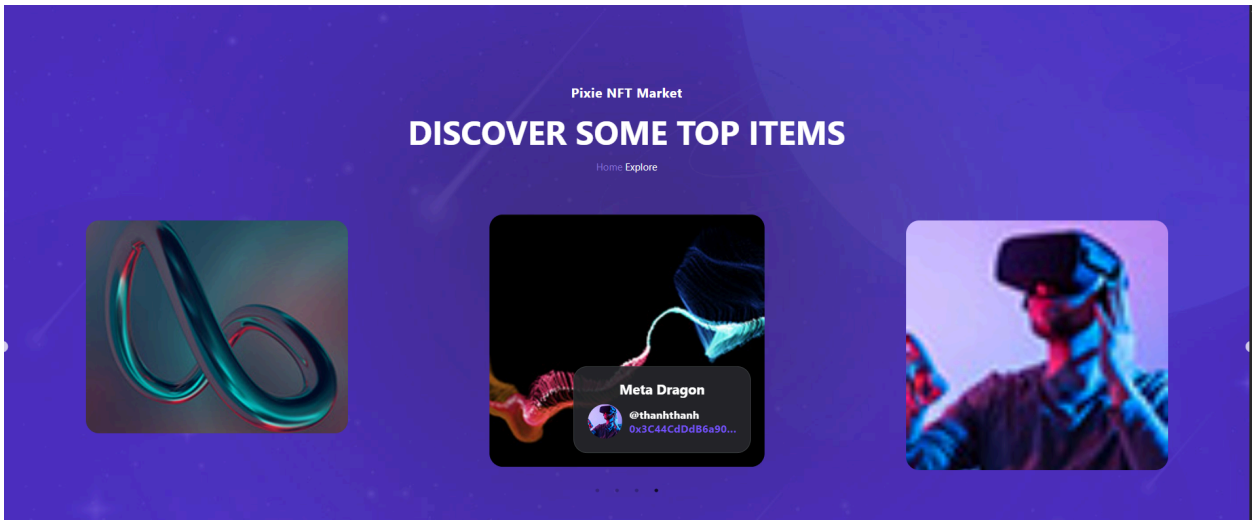


- **Current Bidding Section:** A section displaying the transaction history of all customers using this marketplace, rendered by the CurrentBidding component. Implement a filter for transactions that allows users to view them sorted by "Highest," "Lowest," "Oldest," or "Newest." Each block shows a transaction that the bidder has made and the bidder's wallet address

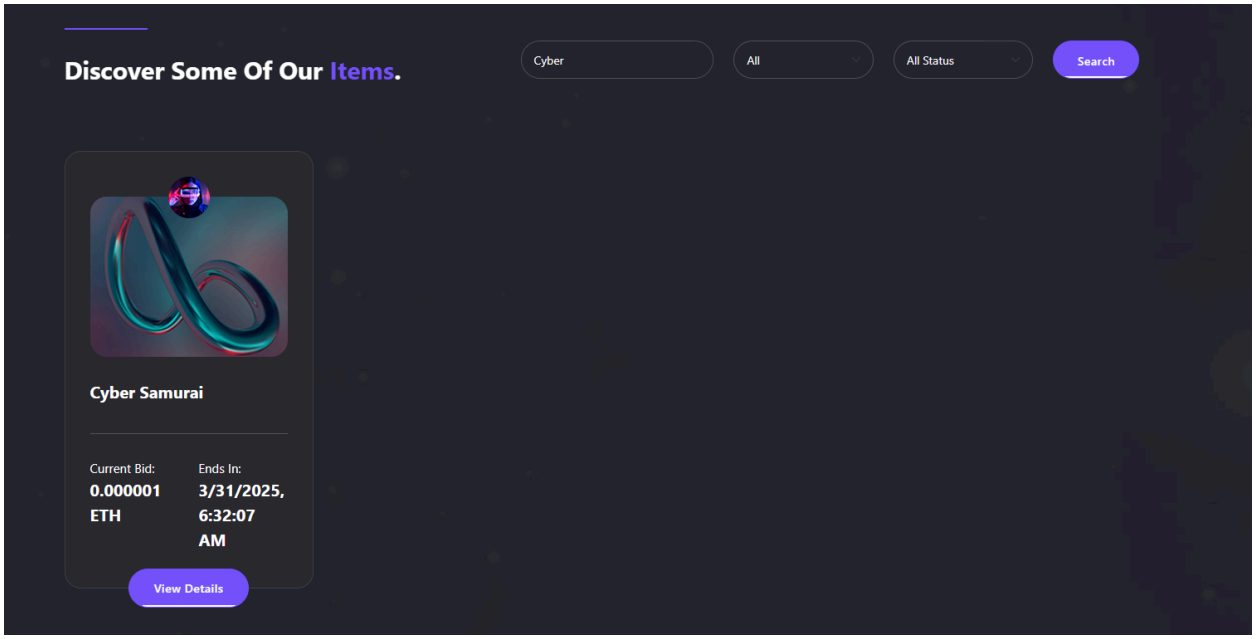


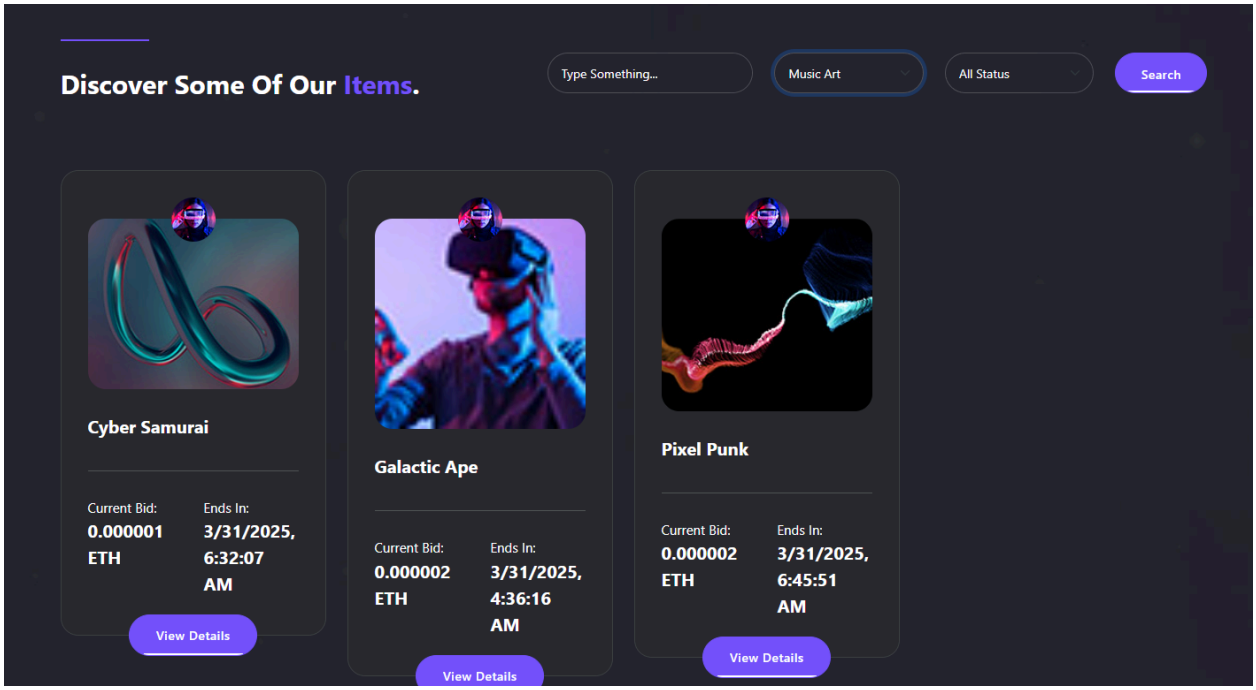
Explore Page

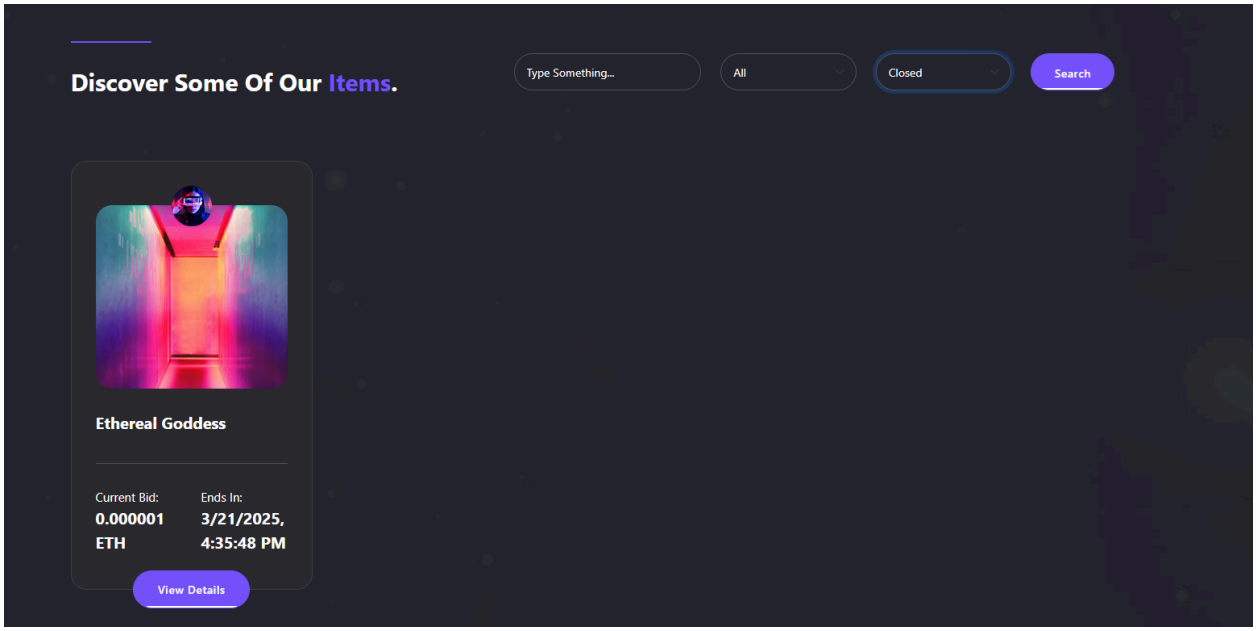
- **Top 4 NFT Section:** Displays the Top 4 NFT items from the marketplace, these NFTs are prominently featured to grab user attention.



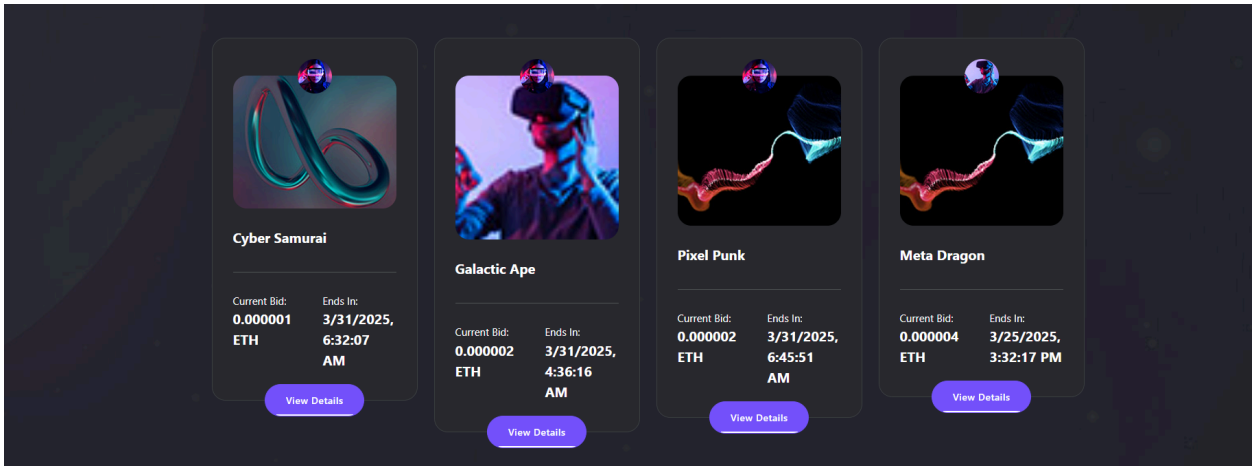
- **Search and Filters:** Search bar to search NFTs by name, category and status. Categories include 4: Music Art, Digital Art, Blockchain, Virtual. Status include 3: Available (more then 24h left), Ending soon (less then 24h left), Closed(bid time ended)



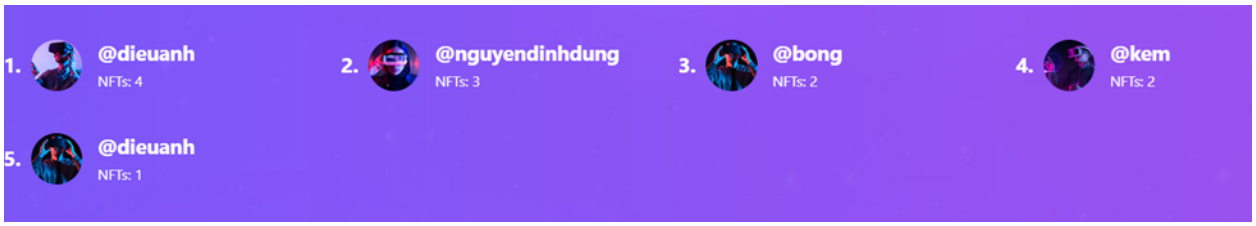




- **NFT Display and Interaction:** Display NFT cards with image, name, current bid in ETH, auction end time, and a "View Details" button.

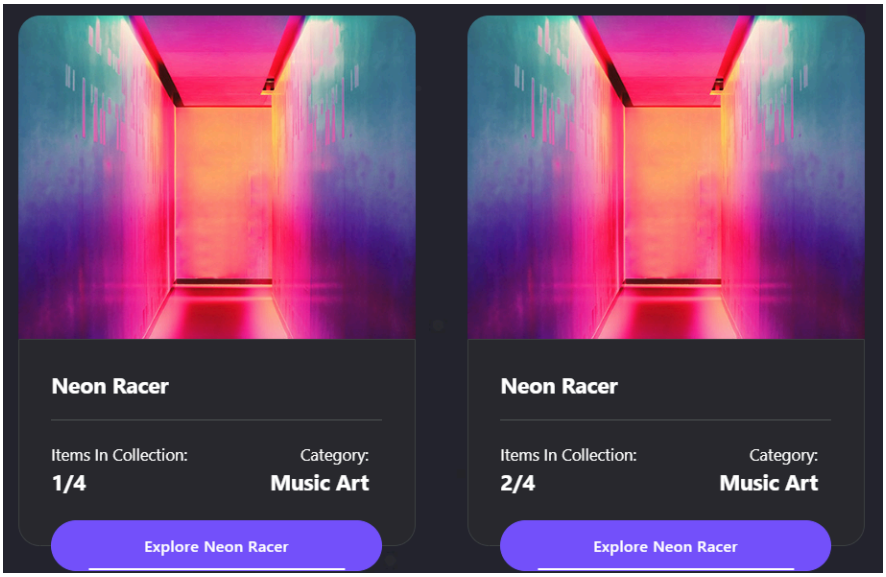


- **Top Sellers Section:** Displays a list of top sellers via the TopSellers component, highlighting the most prominent creators in the marketplace.



Home Page

- **Categories Section:** Each category has an icon and a button that navigates to the Explore page. For each category, the number of NFTs in that category is displayed dynamically, if multiple NFTs have the same name, the number of that NFT increases.



3.3 Non-functional Requirements

To ensure the robustness and ease of use of the Pixie NFT Marketplace, the following non-functional requirements are addressed:

- **Performance:**
    - While formal performance optimization strategies were not extensively implemented, the front end utilizes a **preloading mechanism** to ensure all images, components, and interactive elements fully render before users interact with the page. This method significantly enhances the perceived loading performance and user experience, even without explicit performance benchmarks.

- **Security:**
    - **Secure Blockchain Integration**:
        - Smart contracts securely handle all critical blockchain operations, including NFT creation and transaction finalization.

```
eth_getTransactionCount
eth_sendRawTransaction
  Contract call:       <UnrecognizedContract>
  Transaction:         0x1ad7a6ddafdbea6c38cceb090168bbc3a8515835c6fab60e91aa18ed30739f9b
  From:                0x23618e81e3f5cdf7f54c3d65f7fbc0abf5b21e8f
  To:                  0x73511669fd4de447fed18bb79bafeac93ab7f31f
  Value:               18000. gwei
  Gas used:            53440 of 53440
  Block #17:           0x4390b76cfd64a4c86cd2667612b638d6630f41e9587004a59f291081794f1d54
```

- All interactions and transactions are digitally signed and verified through blockchain protocols, ensuring transaction authenticity.

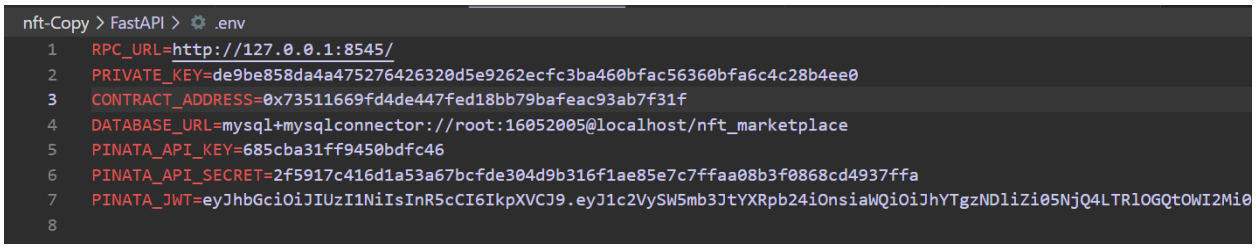
```
📌 Blockchain NFT State Before Transaction:                                    ItemDetails.jsx:109
  {0: 1n, 1: 'Cyber Samurai', 2: '0x8626f6940E2eb28930eFb4CeF49B2d1F2C9C1199', 3: 8000000000000n, 4: '0x23618e81E3f5cdF7f54C
▾ 3d65f7FBc0aBf5B21E8f', 5: true, __length__: 6, id: 1n, name: 'Cyber Samurai', owner: '0x8626f6940E2eb28930eFb4CeF49B2d1F2C
  9C1199', currentBid: 8000000000000n, …} i
    0: 1n
    1: "Cyber Samurai"
    2: "0x8626f6940E2eb28930eFb4CeF49B2d1F2C9C1199"
    3: 8000000000000n
    4: "0x23618e81E3f5cdF7f54C3d65f7FBc0aBf5B21E8f"
    5: true
    active: true
    currentBid: 8000000000000n
    highestBidder: "0x23618e81E3f5cdF7f54C3d65f7FBc0aBf5B21E8f"
    id: 1n
    name: "Cyber Samurai"
    owner: "0x8626f6940E2eb28930eFb4CeF49B2d1F2C9C1199"
    __length__: 6
  ▸ [[Prototype]]: Object
```

- **Secure Transaction Handling**:
  - Explicit transaction confirmations via wallet pop-ups require user consent, mitigating accidental or unauthorized transactions.
  - Additional informative logging during the bidding process enhances transparency and assists in debugging and verifying transactions.

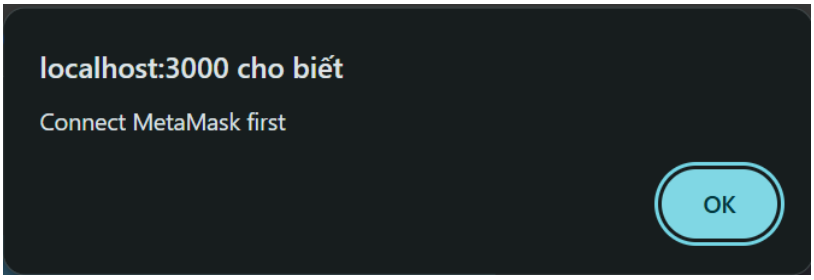
```
✅ NFTs fetched successfully:  ▸ (10) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}]   ItemDetails.jsx:33
✅ Wallet Connected: 0x23618e81e3f5cdf7f54c3d65f7fbc0abf5b21e8f                              ItemDetails.jsx:71
📌 Blockchain NFT State Before Transaction:                                                  ItemDetails.jsx:109
  {0: 1n, 1: 'Cyber Samurai', 2: '0x8626f6940E2eb28930eFb4CeF49B2d1F2C9C1199', 3: 4000000000000n, 4: '0xdD2FD4581271e2303602
▸ 30F9337D5c0430Bf44C0', 5: true, __length__: 6, id: 1n, name: 'Cyber Samurai', owner: '0x8626f6940E2eb28930eFb4CeF49B2d1F2C
  9C1199', currentBid: 4000000000000n, …}
📌 Placing bid on NFT ID: 1 | Amount: 0.000008 ETH | From: 0x23618e81E3f5cdF7f54C3d65f7FBc0aBf5B21E8f   ItemDetails.jsx:111
📌 Transaction Sent, Waiting for Confirmation...                                             ItemDetails.jsx:120
0x0f2eeb1c334f96dca060d66df66b424999bfa505bf00ee0b21b443136df8b2d9
⏳ Checking transaction receipt...                                                           ItemDetails.jsx:125
✅ Transaction Confirmed: 0x0f2eeb1c334f96dca060d66df66b424999bfa505bf00ee0b21b443136df8b2d9  ItemDetails.jsx:134
📌 Sending Data to Backend:                                                                  ItemDetails.jsx:147
▸ {item_id: 1, item_name: 'Cyber Samurai', bid_amount: '0.000008', bid_time: '2025-03-21T16:06:58.379Z', owner_wallet: '0x23
  618e81E3f5cdF7f54C3d65f7FBc0aBf5B21E8f', …}
📌 Backend Response:                                                                         ItemDetails.jsx:157
▸ {message: 'Bid recorded successfully', transaction_hash: '0x0f2eeb1c334f96dca060d66df66b424999bfa505bf00ee0b21b443136df8b2
  d9'}
📌 Fetching NFTs from backend...                                                             ItemDetails.jsx:29
✅ NFTs fetched successfully:  ▸ (10) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}]   ItemDetails.jsx:33
```

- **Sensitive Data Management**:

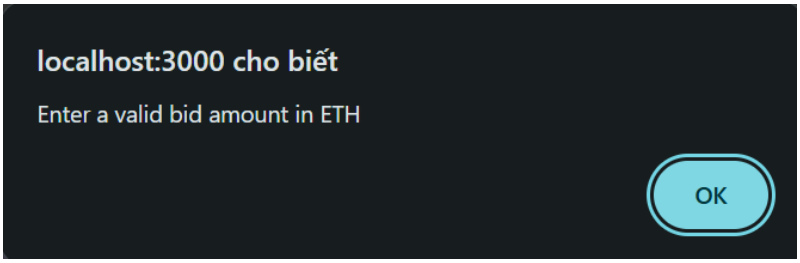
- Critical data such as database credentials, blockchain RPC endpoint, private keys, and Pinata IPFS API keys are securely stored and managed using an environment configuration, significantly reducing exposure to potential security threats.



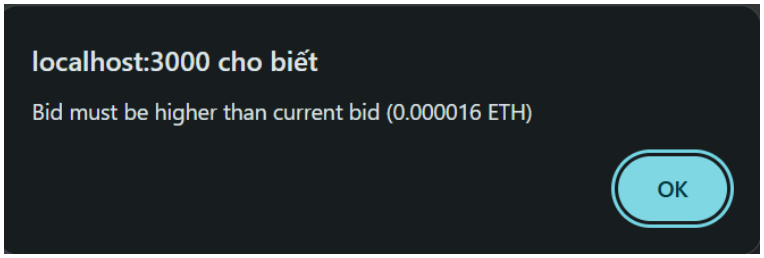
- **Usability & Accessibility:**
  - **User-Friendly Interface**:
    - Clear, explicit pop-up messages guide users through key actions and interactions, including:
      - Reminding users to connect their wallets before bidding or purchasing.



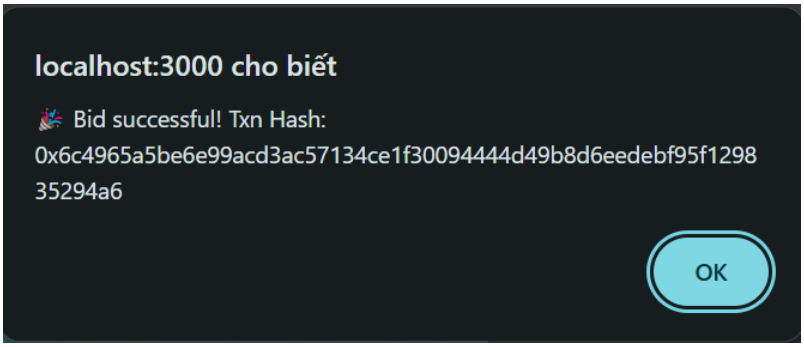
      - Prompting users clearly to input their bids.

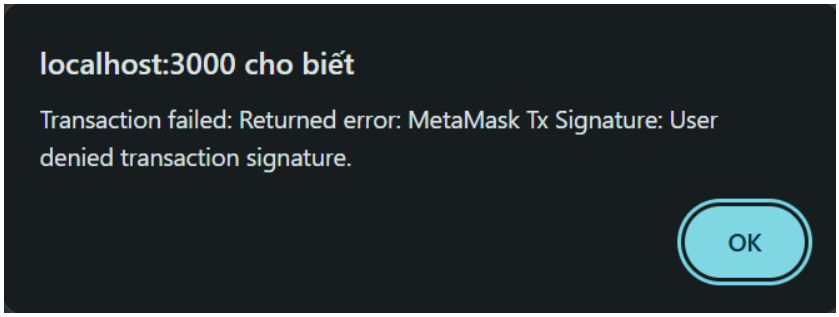


      - Alerting users if their bid needs to be higher than the current highest bid.

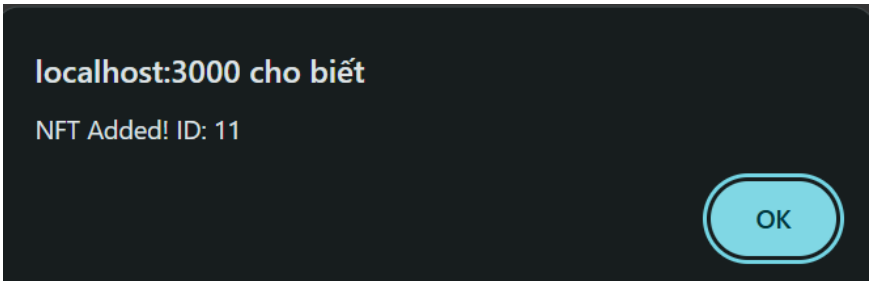


- Notifying users explicitly upon successful transactions or if a transaction is canceled, including clear indications of the transaction hash.

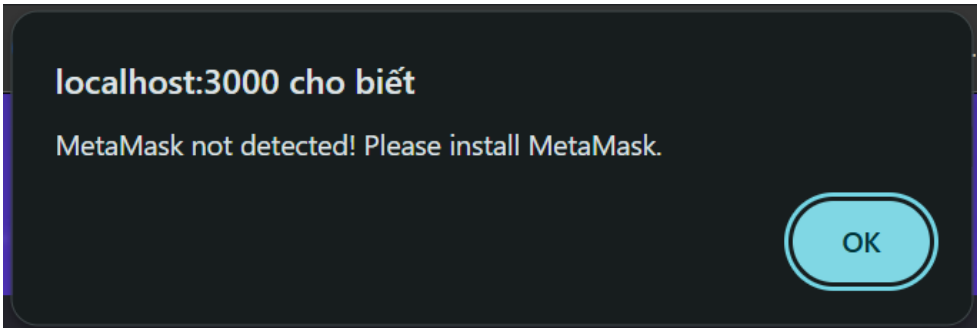




- Notifying users explicitly upon successful create NFT

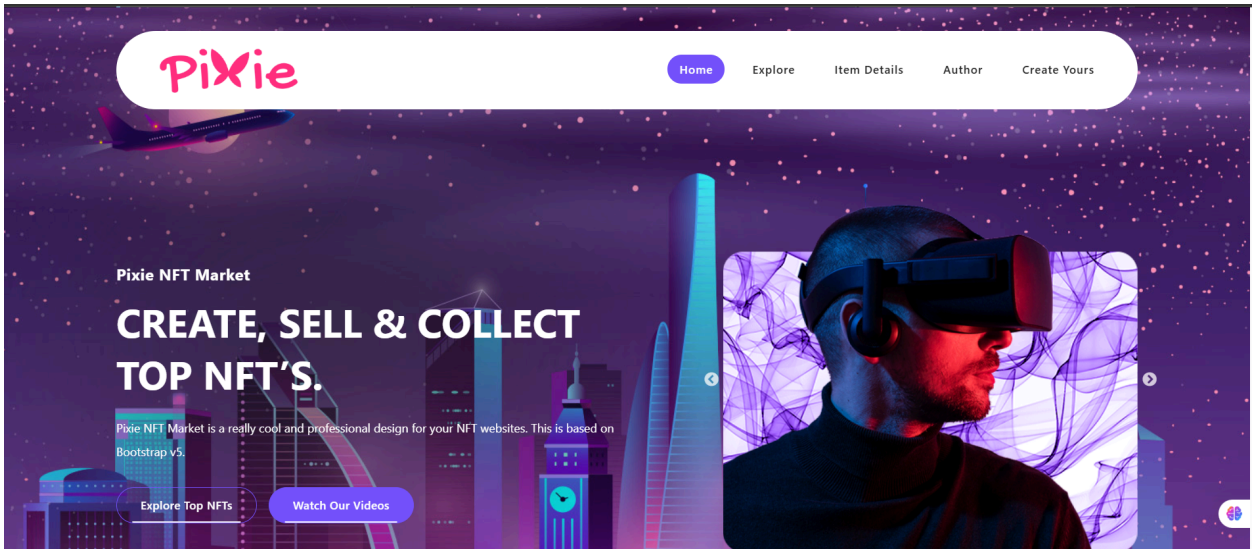


- Notifying users if they haven’t got a metamask wallet

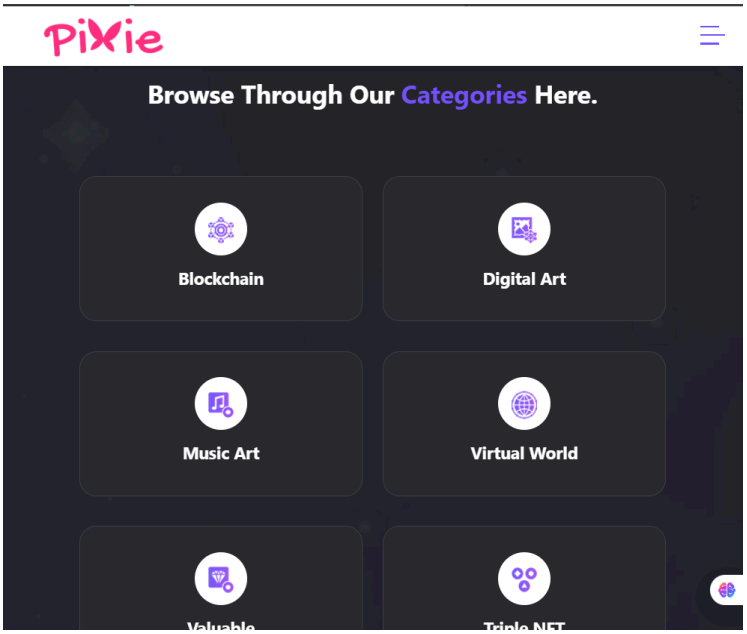


- **Responsive Design**:
  - The user interface is fully responsive, designed to maintain usability and visual clarity across various screen sizes, and specifically optimized for tablets and mobile devices down to a width of 490px.

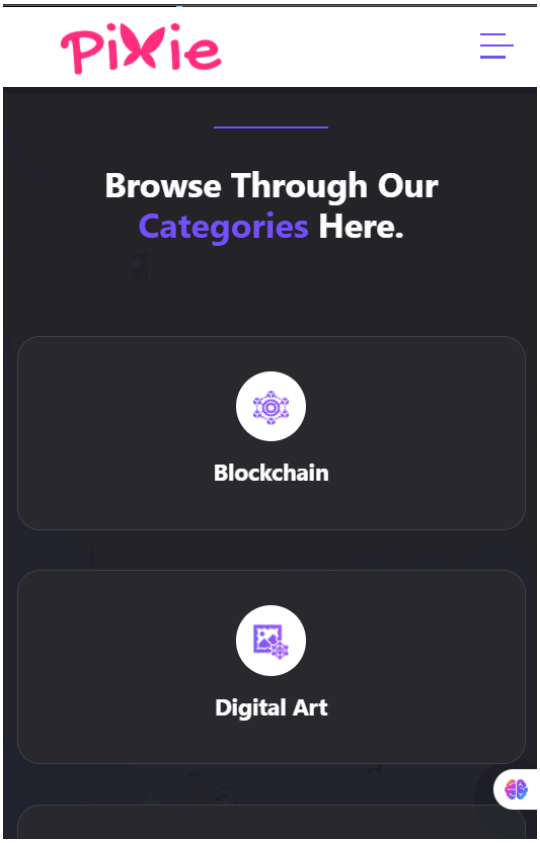
Desktop:



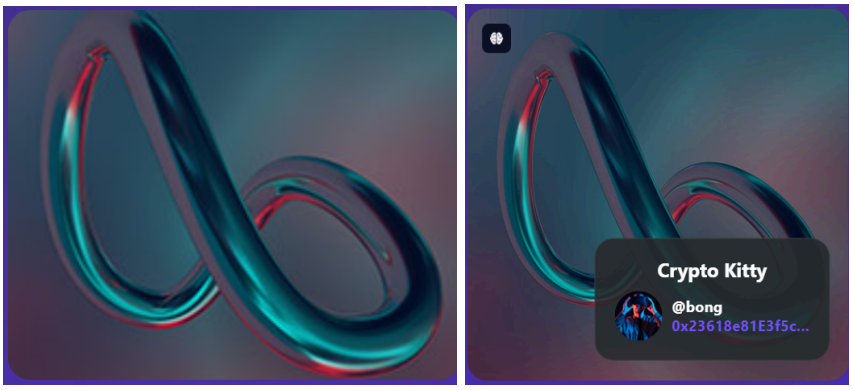
Tablet:


Pixie
Browse Through Our Categories Here.
Blockchain
Digital Art
Music Art
Virtual World

Mobile


Pixie
Browse Through Our
Categories Here.
Blockchain
Digital Art

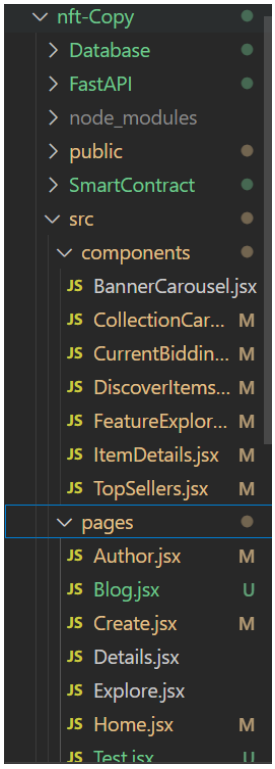
- **Enhanced Interaction and Visual Feedback**:
  - Smooth hover effects and interactive previews of NFTs provide immediate visual feedback, enhancing user engagement and simplifying content exploration.



- **Reliability & Maintainability:**

  - **High Reliability**:
    - Backend operations and smart contract interactions are implemented clearly, promoting stable and error-resistant execution. Detailed error handling mechanisms and logging facilitate rapid identification and resolution of issues.

  - **Maintainable and Modular Codebase:**
    - The modular structure of both frontend (React components) and backend (FastAPI endpoints) ensures ease of future maintenance, code readability, and streamlined expansion for additional features or deployment.



4. Project Design:

4.1 Overall System Architecture Design

The architecture of the Pixie NFT Marketplace is designed to integrate multiple key components, enabling a seamless interaction between the user interface, backend services, the database, and the blockchain network. The system is structured to ensure scalability, security, and efficient data processing. The primary components of the architecture are described below:

A. Backend API and Server-Side Logic

Endpoint Definition

- Clearly defined API endpoints manage operations like NFT creation, bidding, fetching NFT details, transaction history, and user data retrieval.
- FastAPI framework and Pydantic models ensure structured validation, efficient processing, and secure API interactions.

Blockchain Integration

- Direct Ethereum blockchain integration through Web3 ensures secure transaction execution and verification.

- Smart contracts developed in Solidity handle essential NFT marketplace functions, including minting NFTs, managing bids, and transferring NFT ownership.
- Verification processes ensure consistency between on-chain (blockchain) and off-chain (backend database) states for secure and reliable transactions.

B. Database Connectivity

- Uses a locally hosted **MySQL database** for efficient and secure data management.
- Database connections are securely managed through environment variables stored in a .env configuration file.
- Structured database schema with two main tables:
    - **NFT Items Table:** Stores NFT attributes, ownership details, bids, categories, auction statuses, and timing.
    - **Transactions Table:** Records transaction logs, detailed bid data, timestamps, participant wallet addresses, and blockchain transaction hashes.
- **Pinata IPFS** integration for decentralized and secure storage of NFT assets, addressing database limitations in handling large file sizes.
- **FastAPI endpoints** handle NFT retrieval, creation, bidding processes, transactions, author details, rankings, and IPFS image uploads.

C. Smart Contract Integration

- **Contract Overview**
    - Users can **create NFTs** and list them for bidding.
    - Other users can **place bids** (must be higher than the current bid).
    - The NFT owner can **finalize the auction** and transfer the NFT to the highest bidder.
    - The highest bid amount is **transferred to the previous owner** when the auction ends.
- **Breakdown of Code Components**
    - **State Variables**

```
mapping(uint256 => NFT) public nfts;
uint256 public nftCount;
address public contractOwner;
```

nfts: A **mapping** that stores NFTs using their id as the key.

nftCount: Keeps track of the total number of NFTs.

contractOwner: Stores the **address** of the contract deployer.

    - **Struct Definition (NFT)**

```
struct NFT {
    uint256 id;
    string name;
    address owner;
    uint256 currentBid;
    address highestBidder;
    bool active;
}
```

- Each NFT has:
  - **id** → Unique identifier.
  - **name** → NFT name.
  - **owner** → Current owner.
  - **currentBid** → Highest bid amount.
  - **highestBidder** → Address of the highest bidder.
  - **active** → Whether bidding is open.

- **Events**

```
event NFTCreated(uint256 id, string name, address owner);
event BidPlaced(uint256 id, address bidder, uint256 bidAmount);
event NFTTransferred(uint256 id, address newOwner, uint256 finalAmount);
```

These **emit logs** for:

- NFT creation
- New bids
- NFT ownership transfers

- **Modifiers**

```
modifier onlyOwner() {
    require(msg.sender == contractOwner, "Only contract owner can execute this");
    _;
}

modifier onlyNFTOwner(uint256 _id) {
    require(msg.sender == nfts[_id].owner, "Only NFT owner can finalize");
    _;
}
```

- **onlyOwner** → Restricts access to the contract owner.
- **onlyNFTOwner** → Ensures only the NFT owner can finalize an auction.

- **Functions Explained**
  - **Constructor (Sets Contract Owner)**

```
constructor() {    1043124 gas 1017800 gas
    contractOwner = msg.sender;
}
```

- **Sets the deployer's address** as the contract owner.

- **Create an NFT**

```
function createNFT(string memory _name) public {    infinite gas
    nftCount++;
    nfts[nftCount] = NFT(nftCount, _name, msg.sender, 0, address(0), true);
    emit NFTCreated(nftCount, _name, msg.sender);
}
```

- Increments nftCount to assign a new id.
- Stores NFT **with the creator as the owner**.
- Emits NFTCreated event.

- **Place a Bid**

```
function placeBid(uint256 _id) public payable {    infinite gas
    require(_id > 0 && _id <= nftCount, "Invalid NFT ID");
    require(nfts[_id].active, "NFT is not active for bidding");
    require(msg.value > nfts[_id].currentBid, "Bid must be higher than current bid");



    // Update highest bid
    nfts[_id].currentBid = msg.value;
    nfts[_id].highestBidder = msg.sender;

    emit BidPlaced(_id, msg.sender, msg.value);
}
```

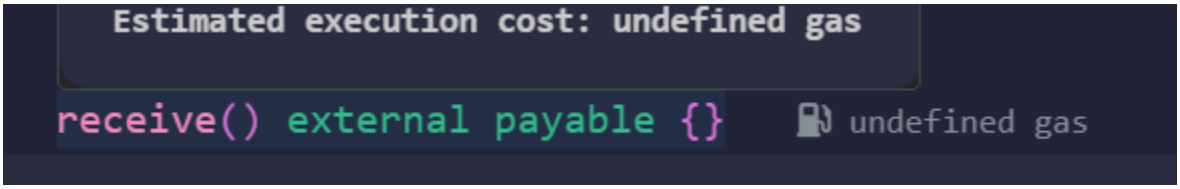
- **Checks**:
  - ☑ Valid NFT ID.
  - ☑ NFT is **active** for bidding.
  - ☑ The bid amount is **higher than the current bid**.
- **Updates bid details** (currentBid & highestBidder).
- Emits BidPlaced event.

- **Finalize the Auction (Transfer NFT & Funds)**

```
function finalizeAuction(uint256 _id) public onlyNFTOwner(_id) {    infinite gas
    require(nfts[_id].active, "Auction is already closed");
    require(nfts[_id].highestBidder != address(0), "No bids placed");

    nfts[_id].active = false; // Close auction
    address previousOwner = nfts[_id].owner;
    address winner = nfts[_id].highestBidder;
    uint256 finalPrice = nfts[_id].currentBid;

    nfts[_id].owner = winner; // Update ownership before sending ETH

    // Securely transfer funds
    (bool success, ) = payable(previousOwner).call{value: finalPrice}("");
    require(success, "Transfer to owner failed");

    emit NFTTransferred(_id, winner, finalPrice);
}
```

- **Checks**:
  - ☑ The auction is active.
  - ☑ At least one valid bid exists.
- **Updates ownership** to the highest bidder.
- **Transfers funds** securely to the previous owner.
- Emits NFTTransferred event.

○ **Allow Contract to Receive ETH**



- This **enables the contract to accept ETH** deposits.

D. Integration Overview

The overall system architecture is characterized by a well-defined data flow among the various components:

- **User Interaction and API Requests:**
  Users interact with the frontend, which sends HTTP requests to the backend API. These requests cover actions such as retrieving NFT data, submitting bids, or uploading files.

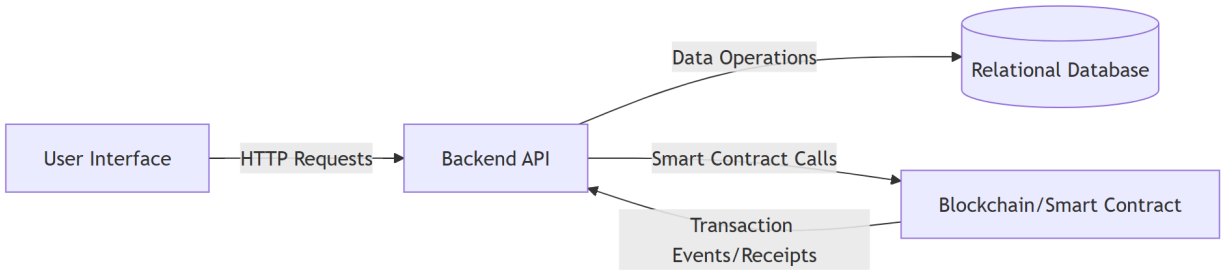
- **Database Operations:**
  When the backend receives a request, it interacts with the relational database to fetch or update records. For instance, upon placing a bid, the backend verifies the bid against existing records before updating bid details and recording the transaction.

- **Blockchain Verification:**
  In parallel, the backend communicates with the blockchain to verify the integrity of

transactions. Before finalizing operations like bid processing, it ensures that the on-chain state is consistent with off-chain data.

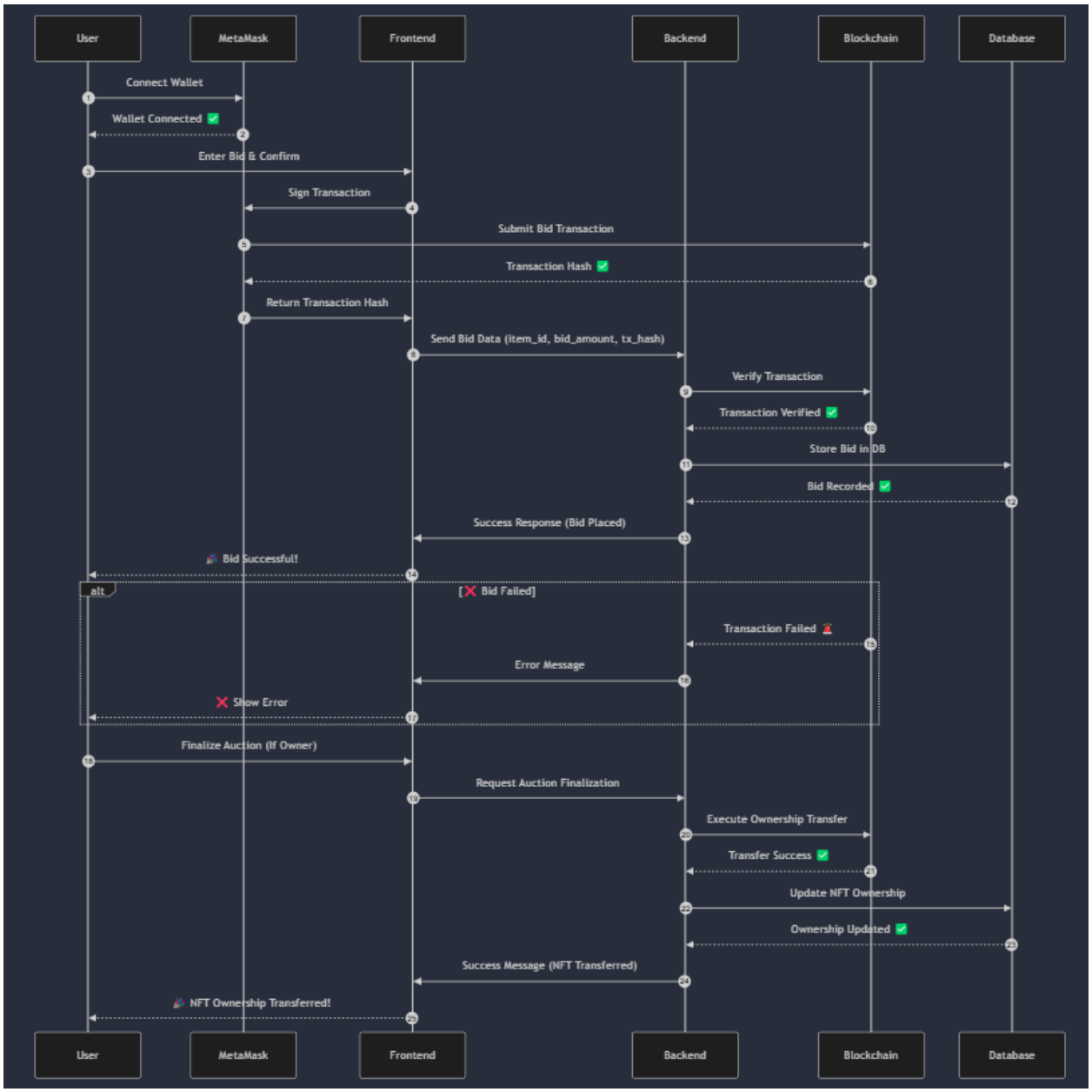
- **Smart Contract Execution:**
  Operations that alter the state of an NFT, such as creation or bidding, invoke corresponding functions in the smart contract. This ensures that all transactions are recorded immutably on-chain, providing a decentralized audit trail.

E. Diagrammatic Representation



A high-level diagram illustrates the following interactions:

- **User Interface:** Communicates with the backend via HTTP requests.
- **Backend API:** Processes requests, manages data operations, and coordinates between the database and the blockchain.
- **Database:** Securely stores NFT and transaction data.
- **Blockchain:** Provides immutable recording of critical transactions through smart contracts.



This diagram shows:

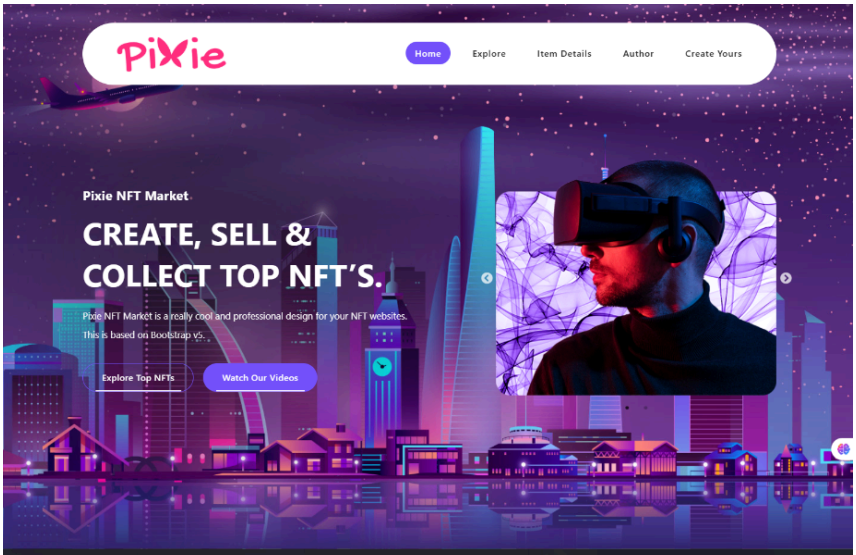
- **Wallet connection** with MetaMask
- **Bidding process** from frontend to blockchain
- **Transaction verification** and bid storage
- **Error handling** if the transaction fails

Overall, the architecture combines the rapid development capabilities of modern web frameworks with robust data management and the security of blockchain technology. This integrated approach ensures that the Pixie NFT Marketplace is scalable, secure, and capable of handling dynamic user interactions and complex financial transactions.

4.2 Front-end Prototype

The front-end prototype of the Pixie NFT Marketplace is developed using ReactJS and is designed to provide a dynamic, user-friendly interface. The prototype is composed of several key pages, each serving a distinct purpose in delivering the overall user experience. The design emphasizes responsiveness, intuitive navigation, and a visually appealing layout.

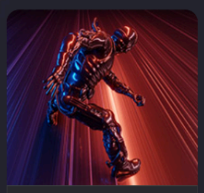
A. Home Page



Browse Through Our Categories Here.

- Blockchain
- Digital Art
- Music Art
- Virtual World
- Valuable
- Triple NFT

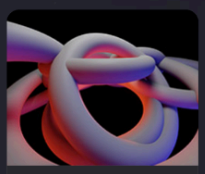
Explore Some Hot Collections In Market.



Ethereal Goddess

Items In Collection: 1/2
Category: Virtual

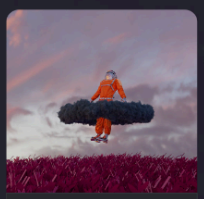
Explore Ethereal Goddess



Galactic Ape

Items In Collection: 1/4
Category: Digital Art

Explore Galactic Ape



Mecha Titan

Items In Collection: 1/2
Category: Blockchain

Explore Mecha Titan

Create Your NFT & Put It On The Market.

Create Your NFT Now

1

Set Up Your Wallet

Lorem ipsum dolor sit amet, consectetu dipiscingei elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

2

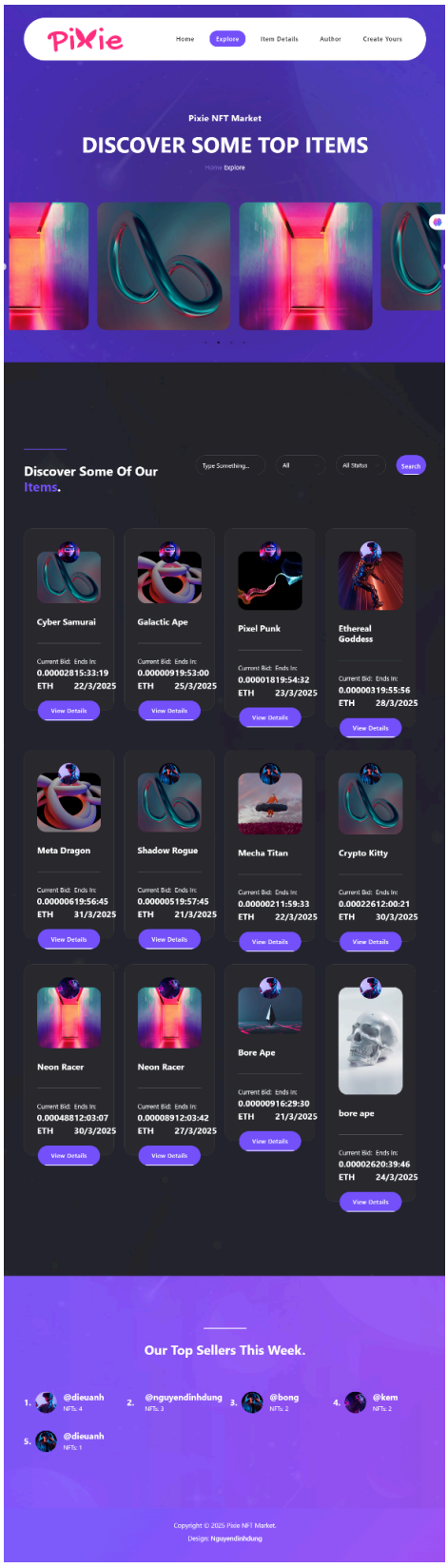
Add Your Digital NFT

Lorem ipsum dolor sit amet, consectetu dipiscingei elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

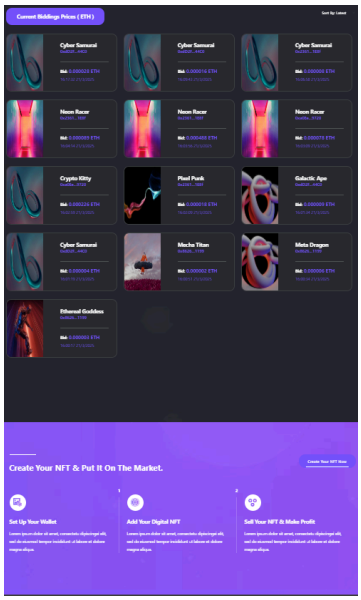
Sell Your NFT & Make Profit

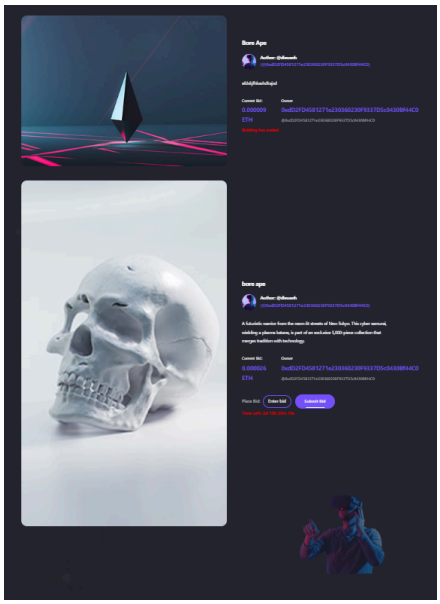
Lorem ipsum dolor sit amet, consectetu dipiscingei elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

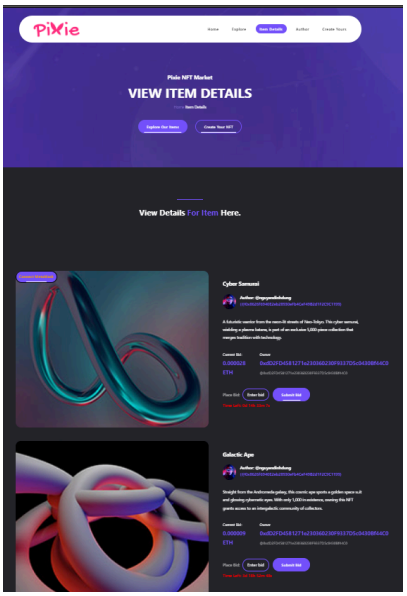
B. Explore Page



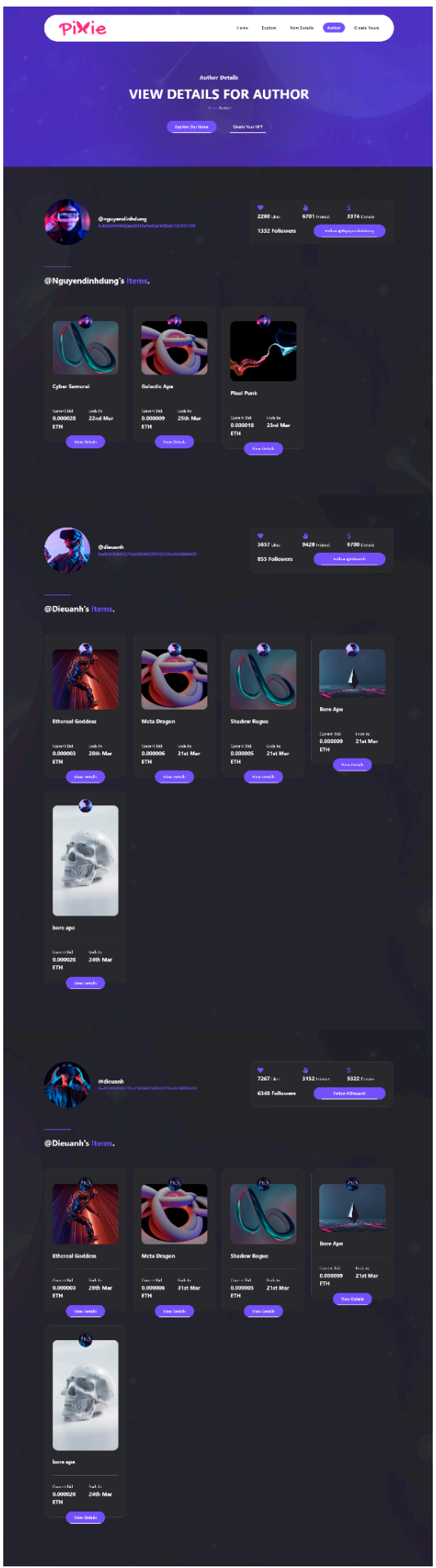
C. Item Details Page



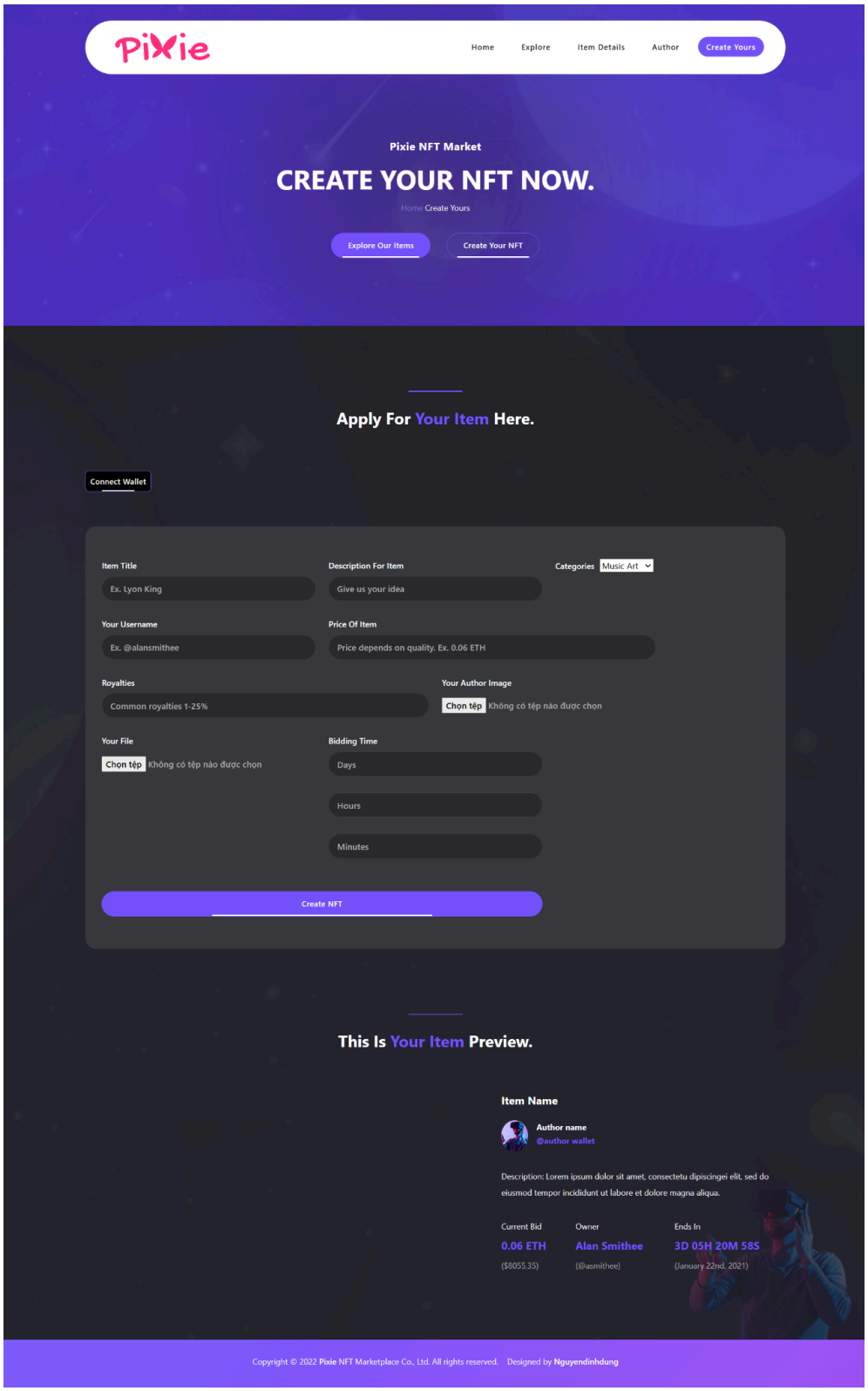




D. Author Page


PiXie
Home
Explore
Item Details
Author
Create Nuns
Author Details
VIEW DETAILS FOR AUTHOR
Explore Our Items
Create Your NFT
@nguyendinhdung
2260 Likes
6701 Interest
3374 Details
1332 Followers
Follow @Nguyendinhdung
@Nguyendinhdung's Items.
Cyber Samurai
Current Bid
0.000028 ETH
Ends In
22nd Mar
Galactic Ape
Current Bid
0.000009 ETH
Ends In
25th Mar
Pixel Punk
Current Bid
0.000018 ETH
Ends In
23rd Mar
View Details
@dieuanh
3657 Likes
9428 Interest
8700 Details
855 Followers
Follow @Dieuanh
@Dieuanh's Items.
Ethereal Goddess
Current Bid
0.000003 ETH
Ends In
28th Mar
Meta Dragon
Current Bid
0.000006 ETH
Ends In
31st Mar
Shadow Rogue
Current Bid
0.000005 ETH
Ends In
21st Mar
Bore Ape
Current Bid
0.000009 ETH
Ends In
21st Mar
bore ape
Current Bid
0.000026 ETH
Ends In
24th Mar
View Details
@dieuanh
7267 Likes
3152 Interest
9322 Details
6348 Followers
Follow @Dieuanh
@Dieuanh's Items.
Ethereal Goddess
Current Bid
0.000003 ETH
Ends In
28th Mar
Meta Dragon
Current Bid
0.000006 ETH
Ends In
31st Mar
Shadow Rogue
Current Bid
0.000005 ETH
Ends In
21st Mar
Bore Ape
Current Bid
0.000009 ETH
Ends In
21st Mar
View Details
bore ape
Current Bid
0.000026 ETH
Ends In
24th Mar
View Details
@dieuanh
7267 Likes
3152 Interest
9322 Details
6348 Followers
Follow @Dieuanh
@Dieuanh's Items.
Ethereal Goddess
Current Bid
0.000003 ETH
Ends In
28th Mar
Meta Dragon
Current Bid
0.000006 ETH
Ends In
31st Mar
Shadow Rogue
Current Bid
0.000005 ETH
Ends In
21st Mar
Bore Ape
Current Bid
0.000009 ETH
Ends In
21st Mar
View Details
bore ape
Current Bid
0.000026 ETH
Ends In
24th Mar
View Details
@bong
3116 Likes
2727 Interest
1576 Details
7271 Followers
Follow @Bong
@Bong's Items.
Mecha Titan
Current Bid
0.000002 ETH
Ends In
22nd Mar
Crypto Kitty
Current Bid
0.000226 ETH
Ends In
30th Mar
View Details
@kem
7002 Likes
9525 Interest
6522 Details
7951 Followers
Follow @Kem
@Kem's Items.
Neon Racer
Current Bid
0.000488 ETH
Ends In
30th Mar
Neon Racer
Current Bid
0.000089 ETH
Ends In
27th Mar
View Details
Create Your NFT & Put It On The Market.
Create Your NFT Now
1
Set Up Your Wallet
Lorem ipsum dolor sit amet, consectetu dipiscingel elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
2
Add Your Digital NFT
Lorem ipsum dolor sit amet, consectetu dipiscingel elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
3
Sell Your NFT & Make Profit
Lorem ipsum dolor sit amet, consectetu dipiscingel elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Copyright © 2021 Pixie NFT Marketplace Inc., Ltd. All rights reserved. Designed by Nguyendinhdung

E. Create NFT Page



F. Visual and Interaction Design

Across all pages, the prototype emphasizes:

- **Responsive Design:** The layout adjusts seamlessly across various devices, including desktops, tablets, and mobile phones.
- **Intuitive Navigation:** A consistent header with a toggleable menu ensures that users can easily move between pages.
- **Modern Aesthetics:** Use of animations, carousels, and interactive buttons contributes to an engaging user experience.
- **Component Reusability:** The front-end is built with reusable components (e.g., BannerCarousel, CollectionCarousel, and MarketItem) that promote consistency and ease of maintenance.

G. Integration with Backend Services

The front-end prototype is designed to integrate closely with the backend API. It retrieves dynamic content such as NFT listings and user data via API calls and updates the UI in real-time. This integration ensures that the data displayed is current and reflects changes (like new bids or NFT listings) as they occur.

4.3 Backend Database Design

The Pixie NFT Marketplace leverages a **relational MySQL database** to manage and persist critical data, including NFT details and bid transactions. The database schema is carefully defined to optimize data retrieval performance, maintain relational integrity, and ensure the precise handling of transactional data critical to blockchain interactions.

A. Data Models

1. NFT Items Table (nft_items)

```python
# NFT Model
class NFTItem(Base):
    __tablename__ = "nft_items"
    id = Column(Integer, primary_key=True, index=True)
    item_name = Column(String(255))
    item_description = Column(String(500))
    author = Column(String(255))
    author_wallet = Column(String(255))
    owner_wallet = Column(String(255))
    current_bid = Column(Float)
    currency = Column(String(10))
    img_url = Column(String(500))  # ✅ Add this line to store the image URL
    category = Column(String(50))  # ✅ Add category column
    bidding_end_time = Column(DateTime)  # ✅ Add bidding end time column
    author_image = Column(String(500))  # ✅ New column for author image
```

Purpose: Represents an NFT item listed in the marketplace.

Functions:

- **Stores NFT details** (name, description, author, owner, image, category, etc.).
- **Tracks the highest bid** (current_bid) and **auction deadline** (bidding_end_time).
- **Helps display NFTs** in the frontend by fetching details from the database.
- **Supports ownership changes** when an NFT is sold or transferred.

2. Transactions Table (transactions)

```python
# Transaction Model
class Transaction(Base):
    __tablename__ = "transactions"

    id = Column(Integer, primary_key=True, index=True)
    item_id = Column(Integer, ForeignKey("nft_items.id"))
    item_name = Column(String(255))
    bid_amount = Column(Float)
    bid_time = Column(TIMESTAMP, default=datetime.utcnow)
    owner_wallet = Column(String(255))
    author_wallet = Column(String(255))  # ✅ Added column
    transaction_hash = Column(String(255))  # ✅ Added column
```

Functions:

- **Logs each bid and purchase** for an NFT (bid_amount, bid_time).
- **Links to the NFT** via item_id (foreign key to NFTItem).
- **Stores transaction details** like transaction_hash for blockchain verification.
- **Keeps a history of NFT ownership and sales** for transparency.

3. Pydantic model for Bids (BidRequest)

```python
# ✅ Pydantic Model for Bids
class BidRequest(BaseModel):
    item_id: int
    item_name: str
    bid_amount: float
    bid_time: str  # ✅ Auto-validates ISO format
    owner_wallet: str
    author_wallet: str
    transaction_hash: str
```

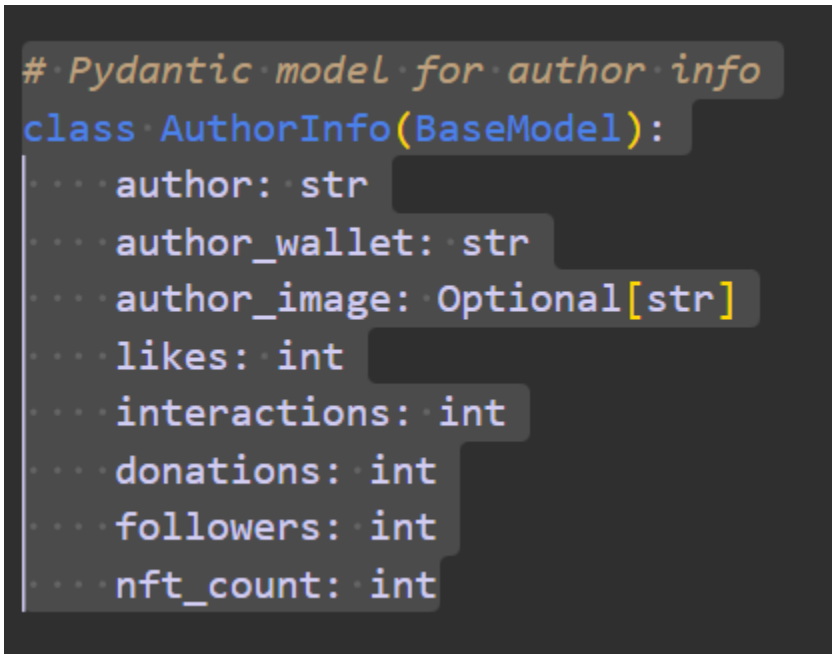
Loading...

Purpose:
Validates and structures bid data before processing in the NFT marketplace.

Functions & Use Cases:

- **Ensures valid bid data** (item_id, bid_amount, bid_time, etc.).
- **Prevents invalid or malicious data** from being stored in the database.
- **Auto-validates timestamps** (bid_time) in ISO format.
- **Stores blockchain transaction details** (transaction_hash) for verification.
- **Links bid requests to NFTs** via item_id.
- **Used in FastAPI endpoints** to handle bid submissions securely.

4. Pydantic Model for Author Info (AuthorInfo)



```python
# Pydantic model for author info
class AuthorInfo(BaseModel):
    author: str
    author_wallet: str
    author_image: Optional[str]
    likes: int
    interactions: int
    donations: int
    followers: int
    nft_count: int
```

Purpose:
Stores and validates information about NFT creators (authors) in the marketplace.

Functions & Use Cases:

- **Tracks creator details** (name, wallet address, profile image).
- **Stores engagement metrics** like likes, interactions, and followers.
- **Records financial support** through donations.
- **Keeps count of NFTs created** by the author (nft_count).
- **Ensures valid author data** before storing or displaying in the marketplace.
- **Used in API responses** for user profiles, leaderboards, and author insights.

5. Pydantic Model for NFT Collection Response (NFTCollectionResponse)

Purpose:
Represents a structured response for displaying NFT collections in the marketplace.

Functions & Use Cases:

- **Stores collection details** such as title and category.
- **Includes a representative image** (image) for the collection.
- **Tracks the number of NFTs** in the collection (items).
- **Provides a link to explore** (exploreLink) more details about the collection.
- **Defines a call-to-action button text** (buttonText) for UI interaction.
- **Ensures valid data formatting** before sending collection responses via API.

B. Database Connectivity

The backend of the Pixie NFT Marketplace utilizes a locally hosted **MySQL database** to effectively manage structured data, including NFT details, user bids, transactions, and related metadata. The database schema clearly defines relational integrity using primary keys and foreign keys, ensuring robust data management and precise transactional accuracy.

Database Connection:

The connection to the MySQL database is securely established and managed using an environment variable defined in the .env file:

- **DATABASE_URL**: The MySQL database connection string (mysql+mysqlconnector://root:88888888@localhost/nft_marketplace) ensures secure, efficient, and stable communication between the FastAPI backend and the local database instance.

Database Schema:

The Pixie NFT Marketplace's database structure is built around two primary relational tables:

1. NFT Items Table (nft_items):

- **Primary Key**:
    - id (**Integer**, auto-increment): Uniquely identifies each NFT within the marketplace.
- **Fields**:
    - item_name (**VARCHAR(255)**): Name of the NFT.
    - item_description (**VARCHAR(500)**): Detailed description provided for each NFT.
    - author (**VARCHAR(255)**): Author’s username or creator's name of the NFT.

    - author_wallet (**VARCHAR(255)**): Ethereum wallet address of the NFT author (validated Ethereum checksum address).
    - owner_wallet (**VARCHAR(255)**): Ethereum wallet address of the current NFT owner (updated after each successful bid).
    - current_bid (**DECIMAL(10,6)**): The latest (highest) bid amount recorded for the NFT.
        - **Precision Explanation**: Using DECIMAL(10,6) allows bid prices to be precise down to 6 decimal places, ensuring users can place bids as small as 0.000001 ETH.
    - currency (**VARCHAR(10)**): Currency used for transactions, typically "ETH".
    - img_url (**VARCHAR(500)**): Decentralized IPFS URL storing NFT images, retrieved via Pinata.
    - author_image (**VARCHAR(500)**): IPFS URL for the author's profile image.
    - category (**VARCHAR(50)**): NFT classification/category.
    - bidding_end_time (**DATETIME**): Timestamp marking when bidding concludes for each NFT.

2. Transactions Table (transactions):

- **Primary Key**:
    - id (**Integer**, auto-increment): Uniquely identifies each individual transaction or bid.
- **Foreign Key Relationship**:
    - item_id (**Integer**, Foreign Key): Linked directly to the id in the nft_items table, ensuring relational integrity and easy access to associated NFT metadata.
- **Fields**:
    - item_name (**VARCHAR(255)**): Name of the NFT related to the transaction (duplicated for quick referencing without table joins).
    - bid_amount (**DECIMAL(10,8)**): Precise transaction amount of the user's bid.
        - **Precision Explanation**: Using DECIMAL(10,8) allows even greater precision (up to 8 decimal places), capturing highly precise bid amounts like 0.00000187 ETH, and accommodating intricate bidding scenarios.
    - bid_time (**TIMESTAMP**): Precise timestamp of when the bid transaction was executed, automatically recorded in UTC.
    - owner_wallet (**VARCHAR(255)**): Ethereum wallet address of the bidder who placed the bid.
    - author_wallet (**VARCHAR(255)**): Wallet address of the NFT author.
    - transaction_hash (**VARCHAR(255)**): Ethereum blockchain transaction hash confirming bid validity and authenticity.

Relationship Management and Updates:

- **Relational Integrity**:
    - The transactions table maintains a robust foreign-key relationship with the nft_items table using the item_id. This ensures transactional records remain accurately connected to specific NFTs, maintaining consistency and traceability across database records.
- **Data Update Mechanics**:
    - When a new bid transaction (POST /bid/{item_id}) is successfully verified via the Ethereum blockchain (Web3), the backend securely updates the current_bid and owner_wallet fields in the related nft_items table.
    - Correspondingly, a new entry is created within the transactions table, accurately recording bid details, timestamps, and blockchain transaction hashes for complete transparency and traceability.

Summary of Data Types and Constraints:

Table Name	Field	Type	Precision	Example
nft_items	current_bid	DECIMAL(10,6)	6 decimals	0.000001 ETH
transactions	bid_amount	DECIMAL(10,8)	8 decimals	0.00000187 ETH

This precise schema definition ensures accurate storage, retrieval, and manipulation of transactional data, supporting the detailed precision required for blockchain-based NFT transactions.

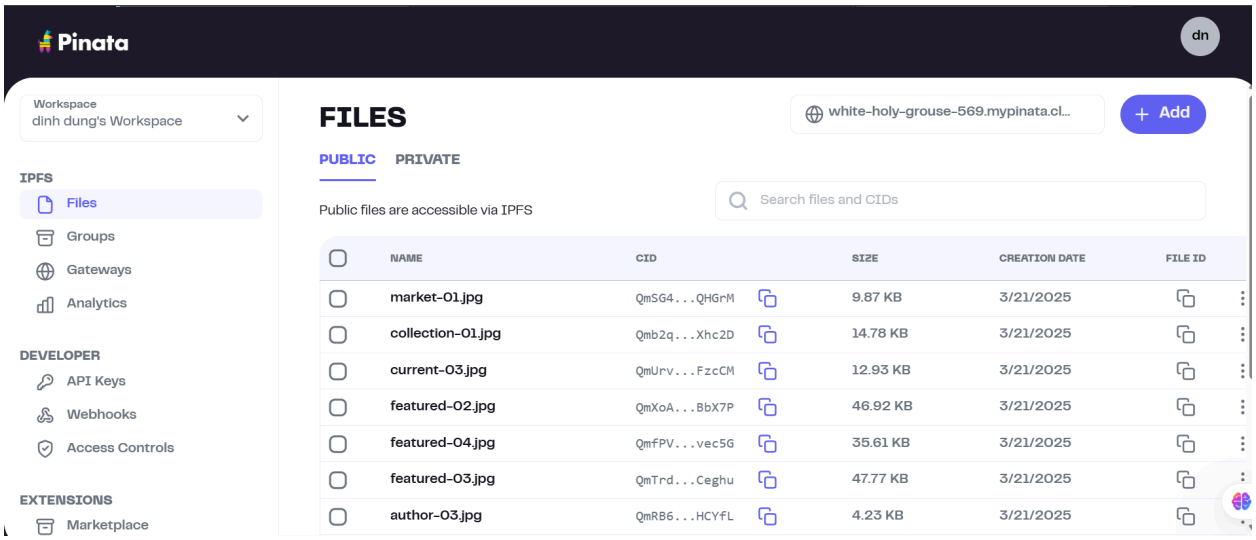
Pinata (IPFS) Integration

Given MySQL's inherent limitation in handling large files such as images, the backend leverages Pinata for decentralized IPFS storage. This integration ensures secure, efficient, and decentralized management of NFT assets:

Authentication details for Pinata are securely stored within the .env file:

- **PINATA_API_KEY** and **PINATA_API_SECRET**: Enable authenticated and secure API interactions with Pinata for file uploads.
- **PINATA_JWT**: Optionally used JWT token offering enhanced authentication security.

NFT images uploaded through Pinata's IPFS infrastructure return decentralized URLs, securely stored in the MySQL database, ensuring optimal storage performance and efficiency.



4.4 API Design

1. Get NFTs

API Name and Description: The **Get NFTs API** provides a service to retrieve information about NFTs. Users can get details of various NFTs available for trading.

API Call in Front End

```
curl -X 'GET' \
  'http://127.0.0.1:8000/nfts/' \
  -H 'accept: application/json'
```

API End Point Definition in Back End

```
# Fetch all NFT items
@app.get("/nfts/")
def get_nfts(db: Session = Depends(get_db)):
  return db.query(NFTItem).all()
```

Request Parameters: No parameters are required for this endpoint.

Response Format:

Code | Details

200

Response body


```
[
  {
    "item_description": "A futuristic warrior from the neon-lit streets of Neo-Tokyo. This cyber samurai, wielding a plasma katana, is part of an exclusive 5,000-piece collection that merges t
radition with technology.",
    "author_wallet": "0x70997970C51812dc3A010C7d01b50e0d17dc79C8",
    "item_name": "Cyber Samurai",
    "current_bid": 0.000001,
    "img_url": "https://gateway.pinata.cloud/ipfs/QmR4mpY7WNBDGNJFSBY36mZ3k3nDNmmG9SfXgyS7nGJacT",
    "bidding_end_time": "2025-03-31T06:32:07",
    "author": "@trmanhson",
    "id": 1,
    "owner_wallet": "0x70997970C51812dc3A010C7d01b50e0d17dc79C8",
    "currency": "ETH",
    "category": "Music Art",
    "author_image": "https://gateway.pinata.cloud/ipfs/QmRjGrVEPViKoD2AYyBh51gaBGjCeTLK7EP8J1jSLp32pX"
  },
  {
```

2. Get NFT

API Name and Description: The Get NFT by Item ID API allows users to retrieve detailed information about a specific NFT using its unique item_id. This endpoint is useful for fetching individual NFT details from the backend based on the item_id parameter.

API Call in Front End

```
curl -X 'GET' \
  'http://127.0.0.1:8000/nfts/2' \
  -H 'accept: application/json'
```

API End Point Definition in Back End

```
# ✅ Fetch single NFT
@app.get("/nfts/{item_id}")
def get_nft(item_id: int, db: Session = Depends(get_db)):
    nft = db.query(NFTItem).filter(NFTItem.id == item_id).first()
    if not nft:
        raise HTTPException(status_code=404, detail="NFT not found")
    return nft
```

Request Parameters: item_id

Response Format:

Code | Details

200

Response body


```
{
  "item_description": "Straight from the Andromeda galaxy, this cosmic ape sports a golden space suit and glowing cybernetic eyes. With only 1,000 in existence, owning this NFT grants access to an intergalactic community of collectors.",
  "author_wallet": "0x70997970C51812dc3A010C7d01b50e0d17dc79C8",
  "item_name": "Galactic Ape ",
  "current_bid": 0.000002,
  "img_url": "https://gateway.pinata.cloud/ipfs/QmcF1PriqkrdvxFgHEjohnedvzAwwVofs4kknZsG7uAUpG",
  "bidding_end_time": "2025-03-31T04:36:16",
  "author": "@trmanhson",
  "id": 2,
  "owner_wallet": "0x70997970C51812dc3A010C7d01b50e0d17dc79C8",
  "currency": "ETH",
  "category": "Music Art",
  "author_image": "https://gateway.pinata.cloud/ipfs/QmRjGrVEPViKoD2AYyBh51gaBGjCeTLK7EP8J1jSLp32pX"
}
```


Download

3. Store Bid

API Name and Description: The Store Bid for NFT API allows users to submit a bid for a specific NFT by providing the item_id and the bid details. This requires user confirmation on meta mask and on-chain transactions so it cannot be shown in FastAPI documents

API Call in Front End

```
curl -X 'POST' \
  'http://127.0.0.1:8000/bid/1' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "item_id": 0,
  "item_name": "string",
  "bid_amount": 0,
  "bid_time": "string",
  "owner_wallet": "string",
  "author_wallet": "string",
  "transaction_hash": "string"
}'
```

API End Point Definition in Back End

```
# Store transaction after MetaMask execution
@app.post("/bid/{item_id}")
async def store_bid(
    item_id: int,
    bid: BidRequest,  # Receives JSON payload
    db: Session = Depends(get_db),
):
```

Request Parameters: item_id

Response Format:

400 *Undocumented* Error: Bad Request

Response body

```
{
  "detail": "Bid must be higher than current bid"
}
```

4. Upload Author Image

API Name and Description: The Store Bid for NFT API allows users to submit a bid for a specific NFT by providing the item_id and the bid details.

API Call in Front End

```
curl -X 'POST' \
  'http://127.0.0.1:8000/upload_author_image/' \
  -H 'accept: application/json' \
  -H 'Content-Type: multipart/form-data' \
  -F 'file=@author.jpg;type=image/jpeg'
```

API End Point Definition in Back End

```
# ✅ Update API to Handle Author Image Upload
@app.post("/upload_author_image/")
```

```
async def upload_author_image(file: UploadFile = File(...)):
```

Request Parameters: file

Response Format:

Code | Details

200

Response body

```
{
  "ipfs_url": "https://gateway.pinata.cloud/ipfs/QmRjGrVEPViKoD2AYyBh51gaBGjCeTLK7EP8J1jSLp32pX"
}
```

Response headers

```
access-control-allow-credentials: true
content-length: 95
content-type: application/json
date: Fri,21 Mar 2025 11:00:26 GMT
server: uvicorn
```

5. Upload Image

API Name and Description: The Upload Image API allows users to upload an image file to the server. The image file is sent via a multipart/form-data request body. This endpoint is typically used for image uploads, such as profile pictures or content images, and requires the user to submit the file in binary format.

API Call in Front End

```
curl -X 'POST' \
  'http://127.0.0.1:8000/upload/' \
  -H 'accept: application/json' \
  -H 'Content-Type: multipart/form-data' \
  -F 'file=@discover-01.jpg;type=image/jpeg'
```

API End Point Definition in Back End

```
# Upload image to IPFS
@app.post("/upload/")
async def upload_image(file: UploadFile = File(None)):
```

Request Parameters: file

Response Format:

6. Add NFT

API Name and Description: The Add NFT API allows users to add a new NFT to the platform by providing necessary information. This endpoint is used to create a new NFT listing for sale or display on the platform.

API Call in Front End

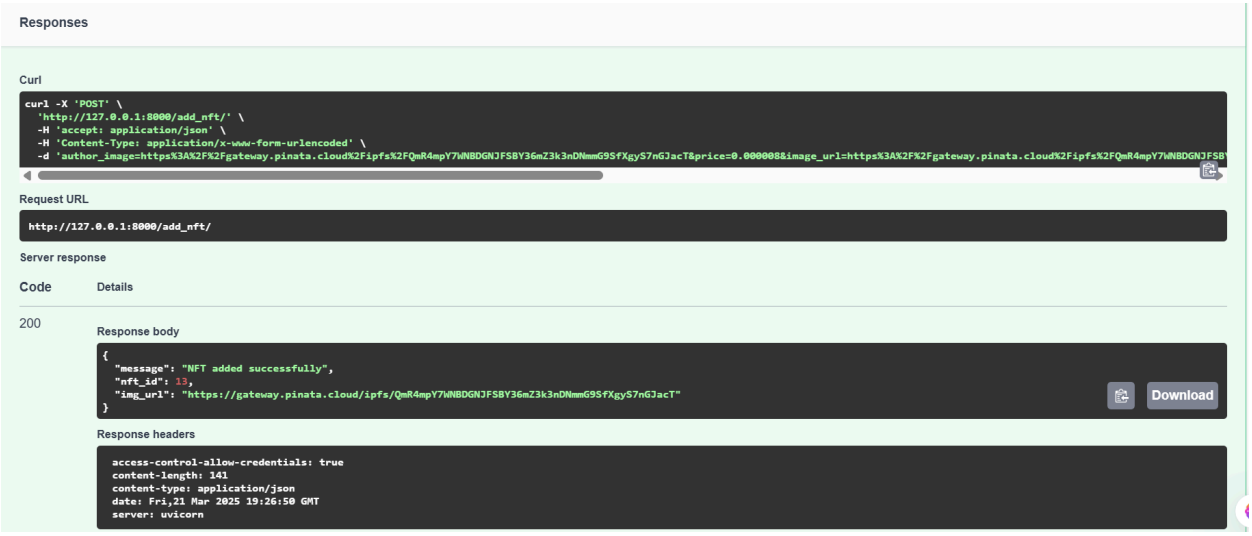
```
curl -X 'POST' \
  'http://127.0.0.1:8000/add_nft/' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/x-www-form-urlencoded' \
  -d 'title=string&description=string&username=string&price=0&author_wallet=string&image_url=string&author_image=string&category=string&days=0&hours=0&minutes=0'
```

API End Point Definition in Back End

```python
# Add NFT with image
@app.post("/add_nft/")
async def add_nft(
    title: str = Form(...),
    description: str = Form(...),
    username: str = Form(...),
    price: float = Form(...),
    author_wallet: str = Form(...),
    image_url: str = Form(...),
    author_image: str = Form(None),  # ✅ Accept author image URL
    category: str = Form(...),  # ✅ Accept category from frontend
    days: int = Form(...),
    hours: int = Form(...),
    minutes: int = Form(...),
    db: Session = Depends(get_db)
):
```

Request Parameters: title, description, username, price, author_wallet, image_url, author_image, category, days, hours, minutes

Response Format:



7. Get NFTs By Owner

API Name and Description: The Get NFTs by Owner API allows users to retrieve a list of NFTs owned by a specific user. This endpoint is useful for checking all the NFTs associated with a particular wallet on the platform.

API Call in Front End

```
curl -X 'GET' \
  'http://127.0.0.1:8000/nfts/owner/0x70997970C51812dc3A010C7d01b50e0d17dc79C8' \
  -H 'accept: application/json'
```

API End Point Definition in Back End

```
# Fetch NFTs by owner wallet
@app.get("/nfts/owner/{owner_wallet}")
def get_nfts_by_owner(owner_wallet: str, db: Session = Depends(get_db)):
    nfts = db.query(NFTItem).filter(NFTItem.owner_wallet == owner_wallet).all()
    if not nfts:
        raise HTTPException(status_code=404, detail="No NFTs found for this wallet")
    return nfts
```

Request Parameters: owner_wallet

Response Format:

200

Response body


```
[
  {
    "item_description": "A futuristic warrior from the neon-lit streets of Neo-Tokyo. This cyber samurai, wielding a plasma katana, is part of an exclusive 5,000-piece collection that merges t
radition with technology.",
    "author_wallet": "0x70997970C51812dc3A010C7d01b50e0d17dc79C8",
    "item_name": "Cyber Samurai",
    "current_bid": 0.000001,
    "img_url": "https://gateway.pinata.cloud/ipfs/QmR4mpY7WNBDGNJFSBY36mZ3k3nDNmmG9SfXgyS7nGJacT",
    "bidding_end_time": "2025-03-31T06:32:07",
    "author": "@trmanhson",
    "id": 1,
    "owner_wallet": "0x70997970C51812dc3A010C7d01b50e0d17dc79C8",
    "currency": "ETH",
    "category": "Music Art",
    "author_image": "https://gateway.pinata.cloud/ipfs/QmRjGrVEPViKoD2AYyBhS1gaBGjCeTLK7EP8J1jSLp32pX"
  },
```


8. Get Transactions By Owner

API Name and Description: The Get NFTs by Owner API allows users to retrieve a list of NFTs owned by a specific user. This endpoint is useful for checking all the NFTs associated with a particular wallet on the platform.

API Call in Front End

```
curl -X 'GET' \
  'http://127.0.0.1:8000/nfts/owner/0x70997970C51812dc3A010C7d01b50e0d17dc79C8' \
  -H 'accept: application/json'
```

API End Point Definition in Back End

```
# Fetch NFTs by owner wallet
@app.get("/nfts/owner/{owner_wallet}")
def get_nfts_by_owner(owner_wallet: str, db: Session = Depends(get_db)):
    nfts = db.query(NFTItem).filter(NFTItem.owner_wallet == owner_wallet).all()
    if not nfts:
        raise HTTPException(status_code=404, detail="No NFTs found for this wallet")
    return nfts
```

Request Parameters: title, description, username, price, author_wallet, image_url, author_image, category, days, hours, minutes

Response Format:

200

Response body


```
[
  {
    "item_description": "A futuristic warrior from the neon-lit streets of Neo-Tokyo. This cyber samurai, wielding a plasma katana, is part of an exclusive 5,000-piece collection that merges t
radition with technology.",
    "author_wallet": "0x70997970C51812dc3A010C7d01b50e0d17dc79C8",
    "item_name": "Cyber Samurai",
    "current_bid": 0.000001,
    "img_url": "https://gateway.pinata.cloud/ipfs/QmR4mpY7WNBDGNJFSBY36mZ3k3nDNmmG9SfXgyS7nGJacT",
    "bidding_end_time": "2025-03-31T06:32:07",
    "author": "@trmanhson",
    "id": 1,
    "owner_wallet": "0x70997970C51812dc3A010C7d01b50e0d17dc79C8",
    "currency": "ETH",
    "category": "Music Art",
    "author_image": "https://gateway.pinata.cloud/ipfs/QmRjGrVEPViKoD2AYyBhS1gaBGjCeTLK7EP8J1jSLp32pX"
  },
```

9. Get Transactions

API Name and Description: The Get Transactions API allows users to retrieve a list of all transactions on the platform. This endpoint provides details about the transaction data.

API Call in Front End

```
curl -X 'GET' \
  'http://127.0.0.1:8000/transactions/' \
  -H 'accept: application/json'
```

API End Point Definition in Back End

```
@app.get("/transactions/")
def get_transactions(db: Session = Depends(get_db)):
    transactions = db.query(Transaction).order_by(Transaction.bid_time.desc()).all()
    return transactions
```

Request Parameters: No parameters required for this endpoint.

Response Format:

200

Response body


```
[
  {
    "item_id": 4,
    "item_name": "Meta Dragon",
    "bid_time": "2025-03-21T11:30:05",
    "author_wallet": "0x3C44CdDdB6a900fa2b585dd299e03d12FA4293BC",
    "bid_amount": 0.000004,
    "id": 11,
    "owner_wallet": "0x70997970C51812dc3A010C7d01b50e0d17dc79C8",
    "transaction_hash": "0xc266830783593fef844c3a79ff96cfa267e421da6ac3a1fc2819546e3305fd3b"
  }
]
```


10. Get Author with Nfts

API Name and Description: The Get Authors with NFTs API allows users to retrieve a list of authors who own NFTs. This endpoint provides a comprehensive list of authors along with the NFTs associated with their wallets.

API Call in Front End

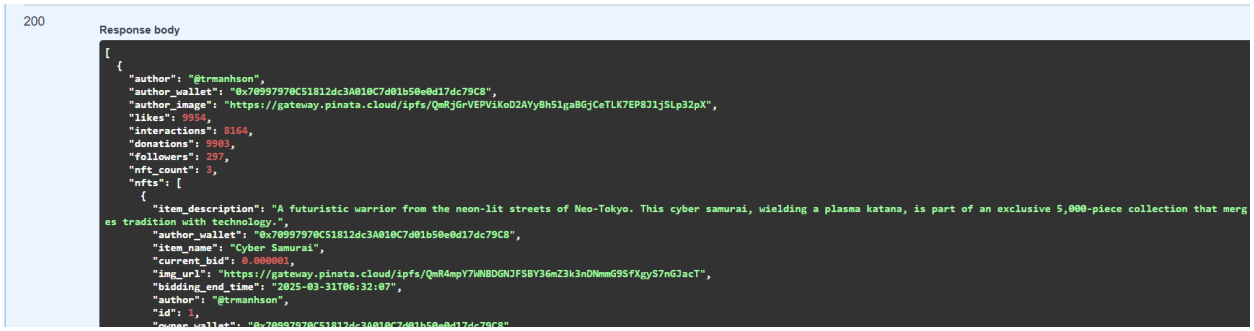
```
curl -X 'GET' \
  'http://127.0.0.1:8000/authors_with_nfts/' \
  -H 'accept: application/json'
```

API End Point Definition in Back End

```
# Endpoint to get all authors with their NFTs
@app.get("/authors_with_nfts/")
def get_authors_with_nfts(db: Session = Depends(get_db)):
```

Request Parameters: No parameters required for this endpoint.

Response Format:



11. Get Author Info

API Name and Description: The Get Author Info API allows users to retrieve information about an author based on their author_wallet. This endpoint is used to fetch the details of an author's profile associated with a specific wallet address.

API Call in Front End

```
curl -X 'GET' \
  'http://127.0.0.1:8000/author/0x70997970C51812dc3A010C7d01b50e0d17dc79C8' \
  -H 'accept: application/json'
```

API End Point Definition in Back End

```
# Endpoint to get author information
@app.get("/author/{author_wallet}")
def get_author_info(author_wallet: str, db: Session = Depends(get_db)):
```

Request Parameters: author_wallet.

Response Format:

Code | Details

200

Response body


```
{
  "author": "@trmanhson",
  "author_wallet": "0x70997970C51812dc3A010C7d01b50e0d17dc79C8",
  "author_image": "https://gateway.pinata.cloud/ipfs/QmRjGrVEPViKoD2AYyBh51gaBGjCeTLK7EP8J1jSLp32pX",
  "likes": 318,
  "interactions": 2921,
  "donations": 6264,
  "followers": 2977,
  "nft_count": 3
}
```

12. Get Top Nfts

API Name and Description: The Get Top NFTs API allows users to retrieve the most popular or top-performing NFTs on the platform. This endpoint provides a list of top NFTs based on predefined criteria, such as sales volume or popularity.

API Call in Front End

```
curl -X 'GET' \
  'http://127.0.0.1:8000/top-nfts/' \
  -H 'accept: application/json'
```

API End Point Definition in Back End

```
@app.get("/top-nfts/")
def get_top_nfts(db: Session = Depends(get_db)):
```

Request Parameters: No parameters required for this endpoint.

Response Format:

Code Details

200

Response body


```
[
  {
    "id": 4,
    "item_name": "Meta Dragon",
    "item_description": "Forged in the fiery depths of the Ethereum blockchain, this legendary dragon NFT symbolizes power, wisdom, and scarcity. Only 500 exist, making it a prized digital col
lectible.",
    "author": "@thanhthanh",
    "author_wallet": "0x3C44CdDdB6a900fa2b585dd299e03d12FA4293BC",
    "owner_wallet": "0x70997970C51812dc3A010C7d01b50e0d17dc79C8",
    "current_bid": 0.000004,
    "currency": "ETH",
    "img_url": "https://gateway.pinata.cloud/ipfs/QmNfuUhtJE3pTFqs9jWnFGNHYCERLfEpFi8FxWFi6BpVBn",
    "category": "Digital Art",
    "bidding_end_time": "2025-03-25T15:32:17",
    "author_image": "https://gateway.pinata.cloud/ipfs/QmcF1PriqkrdvxFgHEjohnedvzAwwVofs4kknZsG7uAUpG"
```


13. Get Top Seller

API Name and Description: The Get Top Sellers API allows users to retrieve a list of the top sellers on the platform. This endpoint provides a list of sellers who have achieved the highest sales or other relevant criteria.

API Call in Front End

```
curl -X 'GET' \
  'http://127.0.0.1:8000/top-sellers/' \
  -H 'accept: application/json'
```

API End Point Definition in Back End

```
@app.get("/top-sellers/")
def get_top_sellers(db: Session = Depends(get_db)):
```

Request Parameters: No parameters required for this endpoint.

Response Format:

Code | Details

200

Response body


```
[
  {
    "rank": 1,
    "name": "@trmanhson",
    "nft_count": 3,
    "author_image": "https://gateway.pinata.cloud/ipfs/QmRjGrVEPViKoD2AYyBh51gaBGjCeTLK7EP8J1jSLp32pX"
  },
  {
    "rank": 2,
    "name": "@thanhthanh",
    "nft_count": 1,
    "author_image": "https://gateway.pinata.cloud/ipfs/QmcF1PriqkrdvxFgHEjohnedvzAwwVofs4kknZsG7uAUpG"
  }
]
```


14. Get Nft Collections

API Name and Description: The Get NFT Collections API allows users to retrieve a list of available NFT collections on the platform. This endpoint provide insights into the collections available for viewing or trading.

API Call in Front End

```
curl -X 'GET' \
  'http://127.0.0.1:8000/nft-collections' \
  -H 'accept: application/json'
```

API End Point Definition in Back End

```
@app.get("/nft-collections", response_model=List[NFTCollectionResponse])
def get_nft_collections(db: Session = Depends(get_db)):
```

Request Parameters: No parameters required for this endpoint.

Response Format:

Code | Details

200

Response body


```
[
  {
    "title": "Cyber Samurai",
    "image": "https://gateway.pinata.cloud/ipfs/QmR4mpY7WNBDGNJFSBY36mZ3k3nDNmmG9SfXgyS7nGJacT",
    "items": "1/3",
    "category": "Music Art",
    "exploreLink": "/explore",
    "buttonText": "Explore Cyber Samurai"
  },
  {
    "title": "Galactic Ape ",
    "image": "https://gateway.pinata.cloud/ipfs/QmcF1PriqkrdvxFgHEjohnedvzAwwVofs4kknZsG7uAUpG",
    "items": "1/3",
    "category": "Music Art",
    "exploreLink": "/explore",
    "buttonText": "Explore Galactic Ape "
  },
```


4.5 Function Description

In each use case, we have included detailed images in the **Functional Requirements List (3.2)** section. These images provide a clear visual representation of how the feature should function and how users will interact with the system. You can refer to that section to better understand the flow and user interface for each functionality. The images complement the written descriptions, offering a more intuitive understanding of the process and expected outcomes.

Functionality Name: MetaMask Wallet Connection

Purpose:
This functionality allows users to connect their MetaMask wallet, fetch the NFTs owned by the wallet, and display them within the application.

Use Cases:

1. **Step 1**: User clicks on the "Connect Wallet" button to initiate MetaMask connection.
2. **Step 2**: MetaMask prompts the user to connect their wallet.
3. **Step 3**: Once connected, the user's wallet address is displayed, and the NFTs owned by the connected wallet are fetched and displayed.
4. **Step 4**: If the wallet is not connected, an error message will be displayed prompting the user to install MetaMask.

Functionality Name: NFT Creation Form

Purpose:
Allows users to input the necessary details to create a new NFT. This includes title, description, username, price, royalties, author image, item image, and bidding time.

Use Cases:

1. **Step 1**: Go to Create Yours page. User connects their wallet, fills out a form with the following fields: title, description, category, username, price, royalties, author image, item image, and bidding time.
2. **Step 2**: The form submits the data to the backend, creating the new NFT with the provided details.
3. **Step 3**: A preview of the created NFT is displayed for the user to confirm the information.

Functionality Name: Display Your NFTs

Purpose:
Fetches and displays the NFTs that are owned by the connected wallet and the transaction history, providing users a quick overview of their collection.

Use Cases:

1. **Step 1**: Go to Create Yours page, Once the wallet is connected, the NFTs owned by the wallet are fetched from the backend API.
2. **Step 2**: Display the NFTs owned, showcasing each NFT's details

Functionality Name: Transaction History

Purpose:
Fetches and displays the transaction history associated with the connected wallet.

Use Cases:

1. **Step 1**: Go to Create Yours page, User clicks on the “connect wallet” button.
2. **Step 2**: The transaction history of the wallet is fetched from the backend API.
3. **Step 3**: Display all transactions with relevant details such as the NFT name, bid amount, and time of bid, transaction hash.

Functionality Name: Item Details

Purpose:
Displays detailed information about a specific NFT item, including the auction status, current bid, and remaining time for the auction. Allows users to place a bid.

Use Cases:

1. **Step 1**: User clicks on Item Details tab to view more details.
2. **Step 2**: The item’s image, author image, author wallet, owner wallet, name, description, current bid, and auction time are displayed.
3. **Step 3**: User can enter a bid amount and submit it after connecting their wallet.
4. **Step 4**: The transaction is confirmed on-chain and updates the item's current bid and ownership in the database.

Functionality Name: Bidding End

Purpose:
Indicates when the auction for an NFT has concluded, removing the "Place Bid" field and locking the current highest bid.

Use Cases:

1. **Step 1**: If the auction has ended, the status "Bidding has ended" is displayed.
2. **Step 2**: The "Place Bid" field is disabled to prevent further bidding.

Functionality Name: Current Bidding Section

Purpose:
Displays the transaction history of all users on the marketplace, providing a filter to view them by "Highest," "Lowest," "Oldest," or "Newest."

Use Cases:

1. **Step 1**: Go to Item Details; the user scrolls to the "Current Bidding" section.
2. **Step 2**: A list of all past transactions is fetched and displayed.
3. **Step 3**: User can filter the transactions by different criteria (highest bid, lowest bid, oldest, newest).

Functionality Name: Top 4 NFT Section

Purpose:
Displays the top 4 NFTs from the marketplace, highlighting the most prominent NFTs to attract user attention.

Use Cases:

1. **Step 1**: Go to Explore page. The top 4 NFTs are fetched from the backend API based on specific criteria (e.g., highest bid).
2. **Step 2**: Display the top 4 NFTs with details such as name and current owner.

Functionality Name: Search and Filters

Purpose:
Allows users to search and filter NFTs by name, category, and auction status.

Use Cases:

1. **Step 1**: Go to Explore page, User enters a search query in the search bar to find NFTs.
2. **Step 2**: User can apply filters such as category (e.g., Digital Art, Music Art) or auction status (e.g., Available , Closed).
3. **Step 3**: NFTs matching the search and filter criteria are displayed.

Functionality Name: Top Sellers Section

Purpose:
Displays a list of the top sellers in the marketplace, highlighting the most prominent creators.

Use Cases:

1. **Step 1**: The list of top sellers is fetched from the backend API.
2. **Step 2**: Display the list of top sellers along with details like their profile picture, name, and the number of NFTs they’ve sold.

Functionality Name: Categories Section

Purpose:
Displays different NFT categories with an icon and a button that links to the Explore page. The number of NFTs in each category is dynamically updated.

Use Cases:

1. **Step 1**: The list of NFT categories (e.g., Digital Art, Music Art) is fetched and displayed.
2. **Step 2**: Each category has an icon and a button that navigates users to the Explore page for that category.
3. **Step 3**: The number of NFTs in each category is dynamically shown based on the current marketplace inventory.

4.6 Project Deployment Instruction

https://drive.google.com/drive/folders/1h5-gXC_I7AWxX0DHC9I5xbVQs6-NM41c?usp=sharing

Check out this Google Drive folder

5. Conclusion

The Pixie NFT Marketplace successfully integrates decentralized blockchain technologies with a responsive and interactive web application, effectively transforming conventional digital asset trading. Built upon a ReactJS frontend and a robust FastAPI backend, the platform ensures efficient data handling, real-time interactions, and a user-friendly experience. By utilizing a locally hosted MySQL database with precise endpoint definitions, it maintains structured data management and relational integrity, crucial for accurate NFT tracking and transaction logging.

The platform's blockchain integration, through Solidity smart contracts and secure Ethereum connections, ensures that all NFT transactions are transparent, secure, and immutable, addressing key issues of trust and security prevalent in traditional centralized marketplaces. Moreover, the decentralized storage provided by Pinata IPFS integration further enhances performance and reliability, particularly for managing large NFT assets.

Overall, the Pixie NFT Marketplace demonstrates a comprehensive and scalable approach, efficiently addressing both functional and non-functional requirements. It stands as a robust, reliable, and user-centric solution well-equipped to adapt to future demands in the rapidly evolving landscape of digital asset trading.

6. References

Nakamoto, S., 2008. *Bitcoin: A Peer-to-Peer Electronic Cash System.* [online] Available at: https://bitcoin.org/bitcoin.pdf

Weking, J., Mandalenakis, M., Hein, A., Hermes, S., Böhm, M. and Krcmar, H., 2020. The impact of blockchain technology on business models – a taxonomy and archetypal patterns. *Electronic Markets*, 30(2), pp.285–305. Available at: https://doi.org/10.1007/s12525-019-00386-3

Wang, L., Luo, X.R., Lee, F. and Benitez, J., 2022. Value creation in blockchain-driven supply chain finance. *Information & Management*, 59(7), Article 103510. Available at: https://doi.org/10.1016/j.im.2021.103510

Harvey, C.R., Hasbrouck, J. and Saleh, F., 2024. The Evolution of Decentralized Exchange: Risks, Benefits, and Oversight. *Wharton Financial Regulation Initiative - White Paper*. SSRN Working Paper Series. Available at: https://ssrn.com/abstract=5132556

Nguyen, D.D., Nguyen, N.T.T. and Tran, M.S., 2025. *Decentralized Trading System (Assignment 1 – COS30049)*. Swinburne University of Technology. Unpublished assignment.

Swinburne University of Technology, 2024. *Backend Development for Interactive Frontend Design (Assignment 2 – COS30049)* [online]. Available at: https://swinburne.instructure.com/courses/64239/assignments/670433