Project Design Document: Decentralized Trading System

COS30049 – Computing Technology Innovation Project

Group 1-25

1 Project Background and Introduction 3

2 Team Introduction 3

3 Project Requirement List and Description 3

3.1 Requirement 1 3

3.2 Requirement 2 5

3.3 Requirement 3 6

3.4 Requirement 4 6

3.5 Requirement 5 7

4 Project Design 8

4.1 Overall System Architecture Design 8

4.2 Front-end Prototype 8

4.3 Backend Database Design 11

4.4 API Design: 11

4.4.1 API 1: Get a list of all assets from the database (and filterings) 11

4.4.2 API 2: get a user's details by their username 12

4.4.3 API 3: Get a list of all transactions associated with a user 13

4.4.4 API 4: Bid an asset of another user with a particular amount of 'money' 13

4.4.5 API 5: Get a list of buying requests for each asset of a user. 13

4.4.6 API 6: used by the owner of an asset to approve the request to buy of another user 14

4.4.7 API 7: authenticates users 14

4.5 Functionality Description 14

4.5.1 Overview of functionalities 14

4.5.2 Details of functionalities 21

4.6 Project Deployment Instructions 22

5 Reference 28

1 Project Background and Introduction

This project is based on the fundamental concepts of a decentralized trading system underpinned by modern blockchain technology. The term “decentralized” means the developed platform is where participants can engage in the trading of digital assets directly with one another without any control or intermediate body. Additionally, by leveraging blockchain technology, it aims to facilitate secure, immutable, transparent transactions and trading processes. Moreover, this project utilizes the power of contemporary front-end technologies (React.js - a JavaScript library) to create an online website that offers users a user-friendly interface and allows them to view and trade available digital assets, as well as to access their transaction history seamlessly.

Conclusively, the project is primarily about creating a decentralized trading platform where the trading of digital assets is conducted transparently and securely. It is freed from intermediaries, letting users trade directly with ease thanks to blockchain technology and a user-friendly website built with modern web frameworks. In this initial project phase (this assignment), the tasks involve designing and developing static front-end web pages that meet specific requirements.

2 Team Introduction

1. Gia Bao Bui - 103533680
2. Khanh Duy Nguyen - 103843994
3. Khanh Linh Nhi Ta – 103809077

3 Project Requirement List and Description

3.1 Requirement 1

Users can view digital assets available for trading.

Per this requirement, the following features have been implemented on the web pages:

Display of Digital Assets: The homepage displays a list of digital assets available for trading.

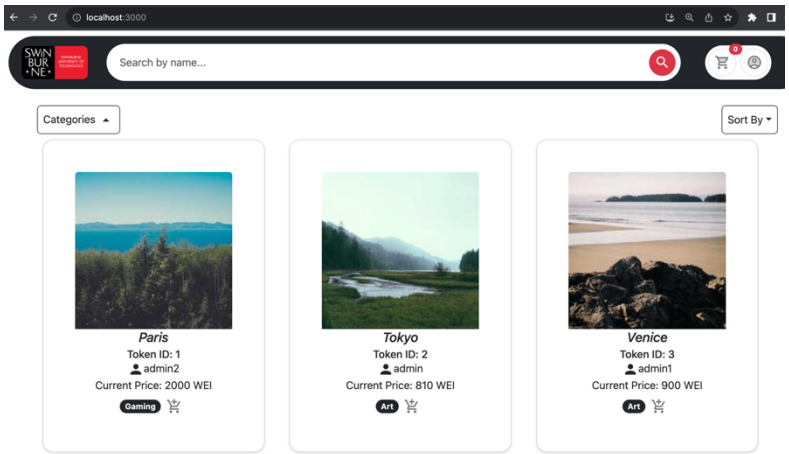

Fig 1. Main web page

Asset Information Cards: Each digital asset is presented in a card format. These cards provide users with essential information about each asset, including its image, token ID, creator's identifications, categories it belongs to, and its price in Wei.

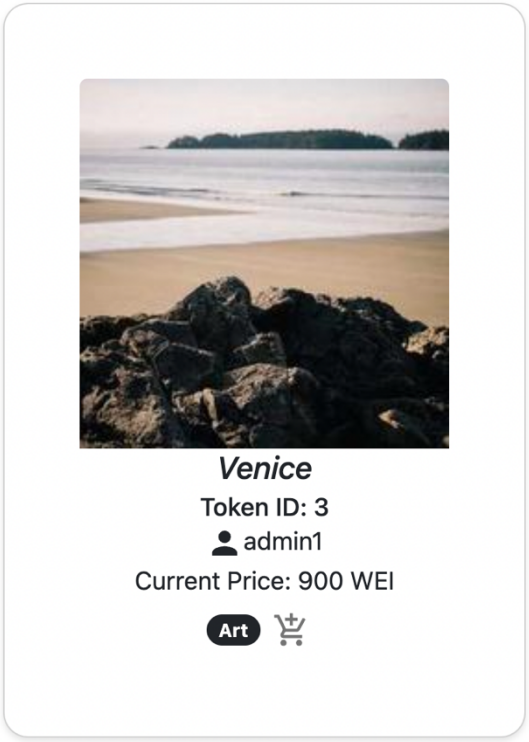

Fig 2. Card-based design of an asset

Consistency and Responsiveness: Besides meeting the core requirement of enabling users to view digital assets available for trading, our website also demonstrates a commitment to enhancing the user experience through consistency in design elements and responsiveness. First, our consistent black-and-white color scheme across all pages enhances user navigation and content focus, making it easy for users to explore digital assets and view assets' images. In addition, our web pages feature a header and logo, providing intuitive navigation. Clicking on the logo seamlessly redirects users to the homepage, where all digital assets are displayed.

Furthermore, the responsiveness of our website ensures that users can view the website on various devices using multiple grid configurations that each fit their respective screen sizes, from desktops to smartphones. This makes the user experience seamless and enjoyable for all users, complementing our primary goal of facilitating digital asset viewing and trading.

We also ensure the responsiveness of overflow contents like tables on devices with narrow screens by making them scrollable, which allows the components of the table to be viewable and have reasonable sizing on any screen size.

In summary, this website effectively enables users to view and gather information about available digital assets for trading and effortlessly add an asset to their shopping cart by clicking on the add button, thus satisfying the specified requirement.

3.2 Requirement 2

All the listed digital assets information should be stored in the database.

The assets' metadata is now stored in a MySQL database. There are API routes defined for the front end to assess those resources.

+ Options

	tokenID	name	category	price	description	currentOwner	contractAddress
Edit Copy Delete	1	Paris	Gaming	2000	The only way to do great work is to love what you ...	admin2	0x511df4fd9890090B5B504a3e0003C7CE622dd73a
Edit Copy Delete	2	Tokyo	Art	810	In the middle of every difficulty lies opportunity...	admin	0x1f2517a6bf00edf4b5c09844a011e2eBD325aB9e
Edit Copy Delete	3	Venice	Art	900	Life is what happens when you're busy making other...	admin1	0xC604A38aF3bf69FBf7eD1B449626883553BD0269

Fig 3. Asset table

```python
# get all the available assets for trading
def getAllAssets(self, sortBy=None, sortOrder="ASC"):
    # connect to the db
    self.connect()
    # take the cursor
    cur = self.con.cursor()
    if sortBy is None:
        query = '''SELECT * FROM Asset;'''
    else:
        query = f'''SELECT * FROM Asset ORDER BY {sortBy} {sortOrder};'''
    # execute the query
    print("Query to be executed: " + query)
    cur.execute(query)
    # fetch all rows
    rows = cur.fetchall()
    assets = [dict(zip(cur.column_names, row)) for row in rows]
    # close the connection
    self.disconnect()
    return assets
```

Fig 4. getAllAsset function

```python
# F1 + F2: Users can view digital assets available for trading.
@app.get("/getAllAssets")
async def getAllAssets():
    data = dtb.getAllAssets()
    return data
```

Fig 5. getAllAssets route

3.3 Requirement 3

The system should provide a search and filter functionality for users to discover specific assets of interest.

This requirement has been effectively addressed through the following features:

1. Search bar: A search bar has been integrated into the website's header. This search bar enables users to input the names (or keywords) of assets they are interested in viewing. Upon submission of their query, the system sends API requests to the backend server and returns the matched data.
2. Category tabs: Additionally, a list of tabs has been placed below the navigational bar. When users click on a category of their choice, the website will fetch API data based on that chosen category and return the corresponding list of assets.
3. Price & Alphabetical sorting: In addition to the two functionalities, Price & Alphabetical sorting has been implemented at the front-end website to further the search capability. Introducing those fragments enhances the system's functionality, providing users with a more customized and user-friendly experience, and facilitating the discovery of digital assets that match their budget and preferences.

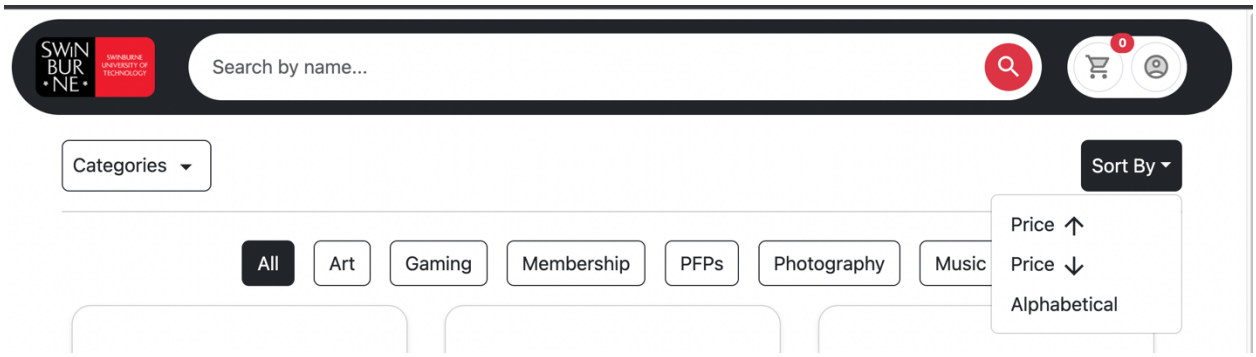

Fig 6. Options to filter or sort data

3.4 Requirement 4

The website will implement smart contracts to act as escrow during the trading process, and smart contracts should ensure that assets are held securely until the trade is completed or cancelled.

Overall, our smart contracts play a key role as reliable go-betweens. They make sure assets are safe until deals are done or stopped. These contracts lock in ownership and token info to keep things safe and prevent anyone from messing with them. Let's dig into how these contracts work and why they're so important for keeping assets secure.

Deploy the Smart Contract: It all begins when the website starts running. For each asset up for grabs, the owner deploys a unique contract using their address and signs with their private key. This contract has three private variables: token ID, current owner address and the initial address. These private variables are integral to the secure and efficient management of asset transactions.

Request to Buy Process: When someone wants to buy an item, they engage in a bidding process, specifying the amount they are willing to pay. A fundamental feature of the contract is its ability to prevent asset owners from initiating purchases of their own assets. This mechanism ensures fairness and transparency in the trading process.

Tracking Participants: The smart contract has a private list called 'participants' with information on the users' requests. To maintain transparency, the 'getParticipants' function allows anyone to access the list of users interested in purchasing the asset, enhancing trade integrity. Furthermore, the owner's address is made visible, reinforcing trust in the individual in charge.

Approve Process: The approval of a sale is a privilege reserved exclusively for the asset's owner. This approval is facilitated through a dedicated function, wherein the owner specifies the recipient of the asset and the proposed price. The smart contract undertakes a comprehensive evaluation of all prospective buyers and selects the chosen one. Subsequently, it initiates the transfer of the funds to the owner and updates the owner's address. In cases where approval is not granted, the participants will receive a refund from the smart contract. Ultimately, the list of potential buyers is cleared, concluding the auction process.

Significance of this Approach: Throughout the entirety of the process, the smart contract ensures the security of both funds and ownership. It mitigates the possibility of external interference, establishing a reliable trading system where trust is paramount. In summary, our smart contract serves as a dependable guardian, ensuring the security of assets and the fairness of transactions.

Why This Matters: All the way through, the smart contract keeps money and ownership safe. It makes sure no one can mess with the process. This way, everyone can trust the trading system. In a nutshell, our smart contract acts as a reliable helper, keeping assets safe and making sure trades are fair and secure.

To sum it up, our smart contract design makes the owner a trusted guardian, ensuring assets are safe and the trading process is efficient. It keeps things clear, fair, and secure, protecting the ownership of assets and how money is handled. Throughout the trading process, the smart contract acts as a strong protector, making sure nothing goes wrong and upholding the need for safe and dependable asset management.

3.5 Requirement 5

Users should have access to a transaction history to view their past trades.

To fulfil this requirement, the website allows users to access their transaction history stored within their User Profile. First off, users need to click the "User" button placed in the website's header, to navigate to their profile. Once users access their profile interface, they can click on the menu button in the following screenshot to trigger the off-canvas sidebar, which contains the menu that navigates to some sections of their account, including the "Transaction History". Users can conveniently access their transaction history page by simply clicking on this option. This not only serves the purpose of meeting the requirement but also underscores our website's commitment to a user-friendly and intuitively logical organization of all elements.

Each entry in the transaction history is equipped with all details, as shown in the accompanying screenshot below. These comprehensive details provide users with a holistic perspective of their previous trades, empowering them to efficiently oversee and manage their transaction history. This implementation ensures that users can effortlessly access and review their transaction history, with each entry supplying crucial information for their reference and effective management of past trades.

4 Project Design

4.1 Overall System Architecture Design

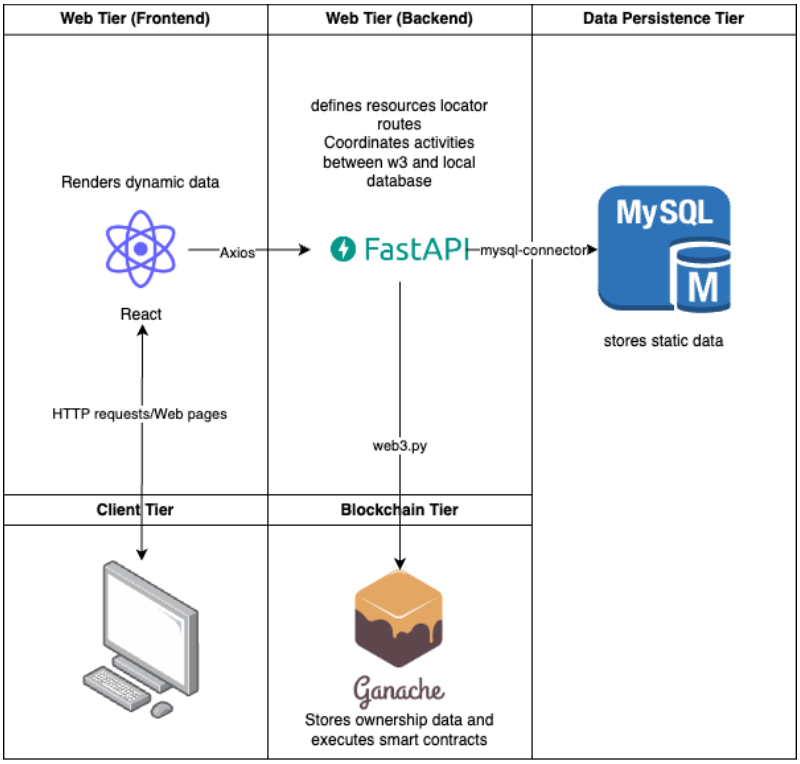

Fig 7. Overall architecture design

4.2 Front-end Prototype

Home Page

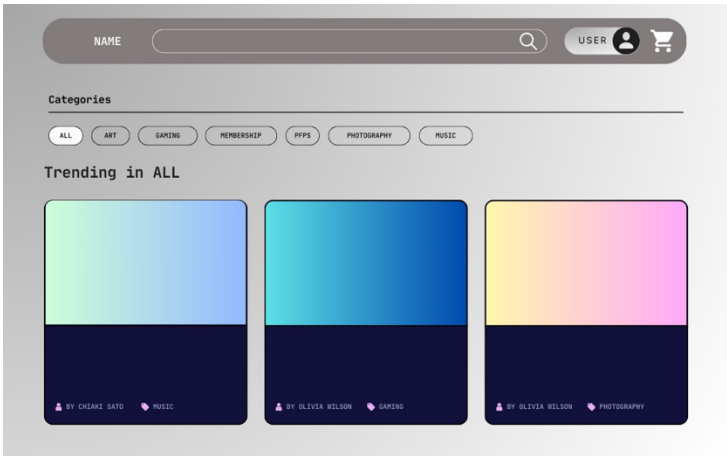

Fig 8. Home page prototype

Home Page with category

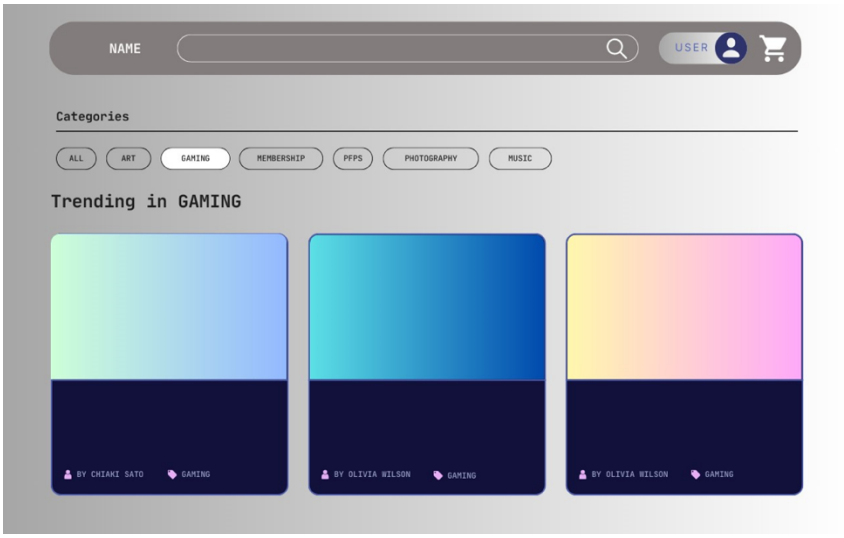

Fig 9. Home page prototype

Home Page with search input

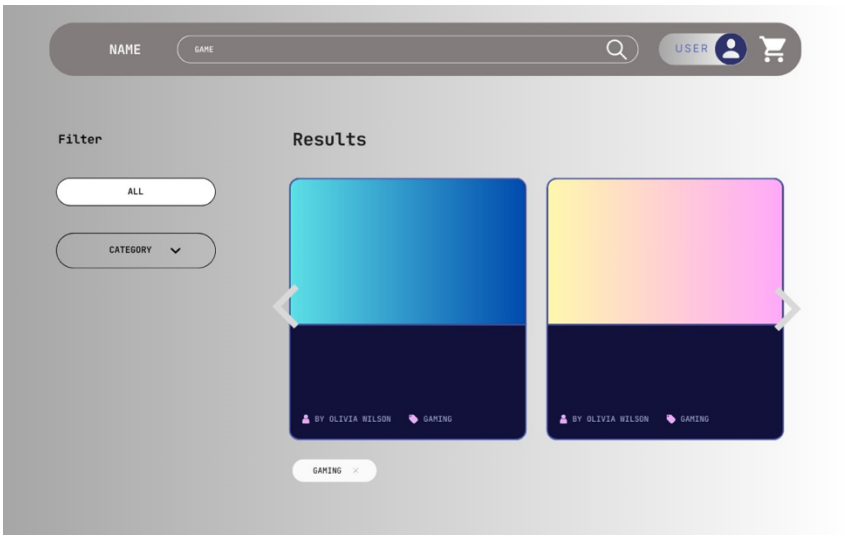

Fig 10. Search prototype

User's Shopping Cart

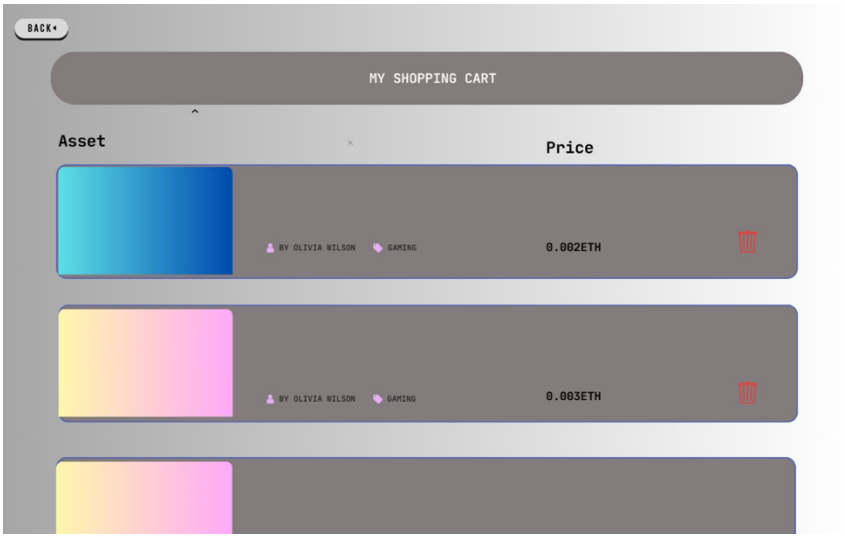

Fig 11. Shopping cart prototype

User's Transaction History

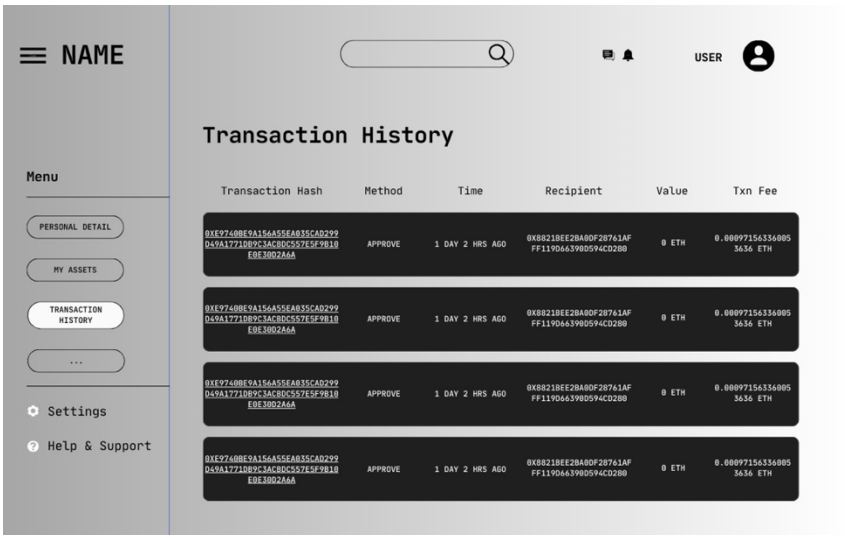

Fig 12. Transaction history prototype

4.3 Backend Database Design

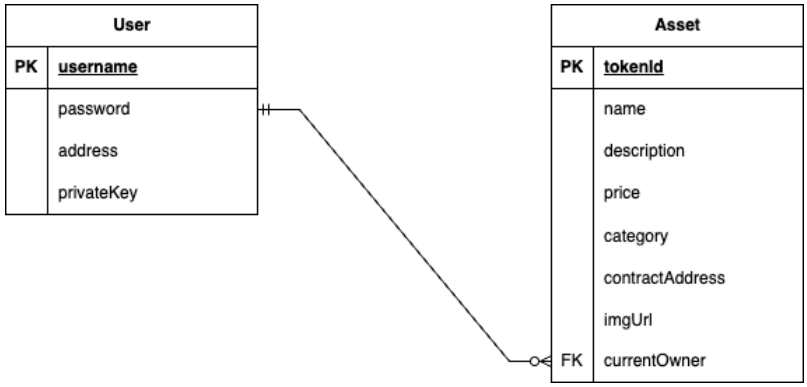

Fig 13. The relational database schema

The local database stores static data for displaying assets, authenticating users, and keeping track of data needed to communicate with the Blockchain server (contractAddress for an Asset, privateKey, address for User)

The local database established a connection between the token ID and its owner. The actual ownership is stored in the database. With this design, users can ensure that the ownership information is immutably recorded somewhere in the system.

4.4 API Design:

API 1: Get a list of all assets from the database (and filterings)

Endpoint:

- **URL:** http://localhost:8000/getAllAssets
- **Method:** GET

Request Parameters:

- /search/{keyword}: to return assets whose name aligns with the keyword
- /category/{category}: to return assets whose category is chosen

Response Format: (json)


```
[
  {
    "tokenID": 1,
    "name": "Paris",
    "category": "PFPs",
    "price": 106.0,
    "description": "The only way to do great work is to love what you do. - Steve Jobs",
    "currentOwner": "admin1",
    "contractAddress": "0x5C41D0DD4c3b879685E8e303175811cFe789556b",
    "imgUrl": "https://picsum.photos/id/10/200/200"
  },
  {
    "tokenID": 2,
    "name": "Tokyo",
    "category": "Gaming",
    "price": 229.0,
    "description": "In the middle of every difficulty lies opportunity. - Albert Einstein",
    "currentOwner": "admin2",
    "contractAddress": "0x00f7db717b45f0C48511BE3cc369c38Bb0fe9F67",
    "imgUrl": "https://picsum.photos/id/11/200/200"
  }
]
```


API 2: get a user's details by their username

Endpoint:

- **URL:** http://localhost:8000/getUserDetails/admin
- **Method:** GET

Request Parameters:

- /{username}: to specify the username of the user that is required

Response Format:


```
{"userDetails":

{"username":"admin",

"password":null,

"address":"0x3446F14EE3628345eB5eB2FE2EABf6D26cC947D1",

"privateKey":"0x1d9e6d130d08f435bf3eda23bfefd8d47ca386a2dbaa49d4a5f05a58a956e788"},
```

"balance":99.9022946}

API 3: Get a list of all transactions associated with a user

Endpoint:

- **URL:** http://localhost:8000/getTransactions/admin
- **Method:** GET

Request Parameters:

- /{username}: to specify the username of the user that is required

Response Format:

{"Transactions":
[{"TxHash":"0xaba3fb5fb06d8ca117aebd55ac2aaeb032362a26b40ed07f82d6e2fe04b3c5ea",
"From":"0x3446F14EE3628345eB5eB2FE2EABf6D26cC947D1",
"To":null,"Value":0,
"BlockNumber":1,
"Method":"Create Asset"}]}

API 4: Bid an asset of another user with a particular amount of 'money'

Endpoint:

- **URL:** http://localhost:8000/admin/1/1000
- **Method:** GET

Request Parameters:

- /{username}: to specify the username of the user that is required
- /{username}/{tokenId}: the token Id of the desired asset
- /{username}/{tokenId}/{amount}: the amount of certain currency user wants to bid on

Response Format:

{"result": "Registered successfully!"}

API 5: Get a list of buying requests for each asset of a user.

Endpoint:

- **URL:** http://localhost:8000/getRequestsToBuyAssets/admin
- **Method:** GET

Request Parameters:

- /{username}: to specify the username of the user that is required

Response Format:

[{"tokenId":2,"participants":[]},{"tokenId":5,"participants":[]},{"tokenId":7,"participants":[]},{"tokenId":9
,"participants":[]}]

API 6: used by the owner of an asset to approve the request to buy of another user

Endpoint:

- **URL:** http://localhost:8000/approve/admin/0x3446F14EE3628345eB5eB2FE2EABf6D26cC947D1/1000/1
- **Method:** GET

Request Parameters:

- /{username}: to specify the username of the current owner
- /{username}/{newOwnerAddress}: to specify the address of the new owner
- /{username}/{newOwnerAddress}/{value}: to get the bid value of the user who initiated the request
- /{username}/{newOwnerAddress}/{value}/{tokenId}: to get the token Id of the traded asset

Response Format:

{“result”: ”Approved!”}

API 7: authenticates users

Endpoint:

- **URL:** http://localhost:8000/authenticate/admin/admin
- **Method:** GET

Request Parameters:

- /{username}: to specify the username of the current user
- /{username}/{password}: to specify the password to be validated against the database

Response Format:

{"result": "authenticated"}, {"result": "Invalid credentials!"}

4.5 Functionality Description

Overview of functionalities

The website displays the assets of users on the platforms. Users browse assets of interest and add them to their shopping cart. Users bid money on the asset they want to buy (must be higher than the current highest price) and initiate a request to buy the asset. The owner of the assets has an interface to view current requests for their assets and is able to approve the request of one particular user. After the approval, the asset belongs to the new owner (the ownership is established in both the local database and the local blockchain)

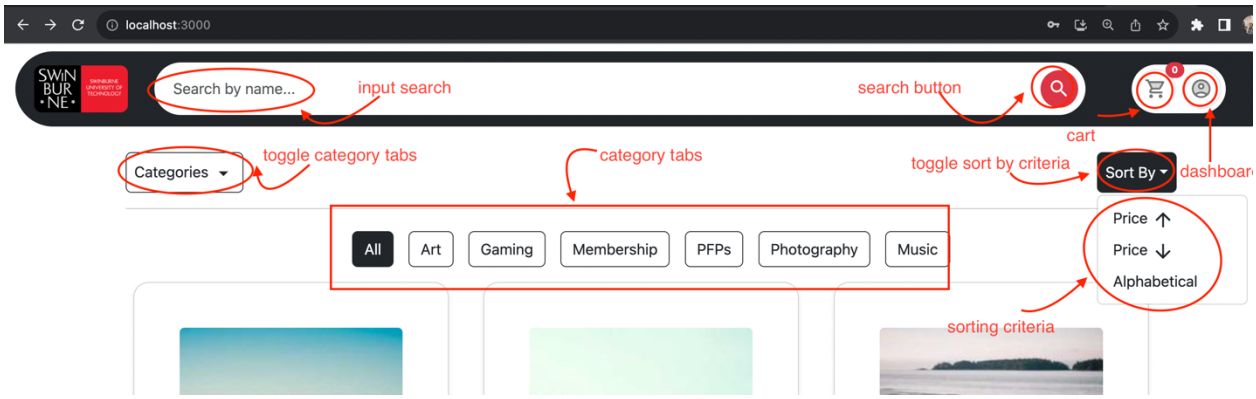

Fig 14. Main web page

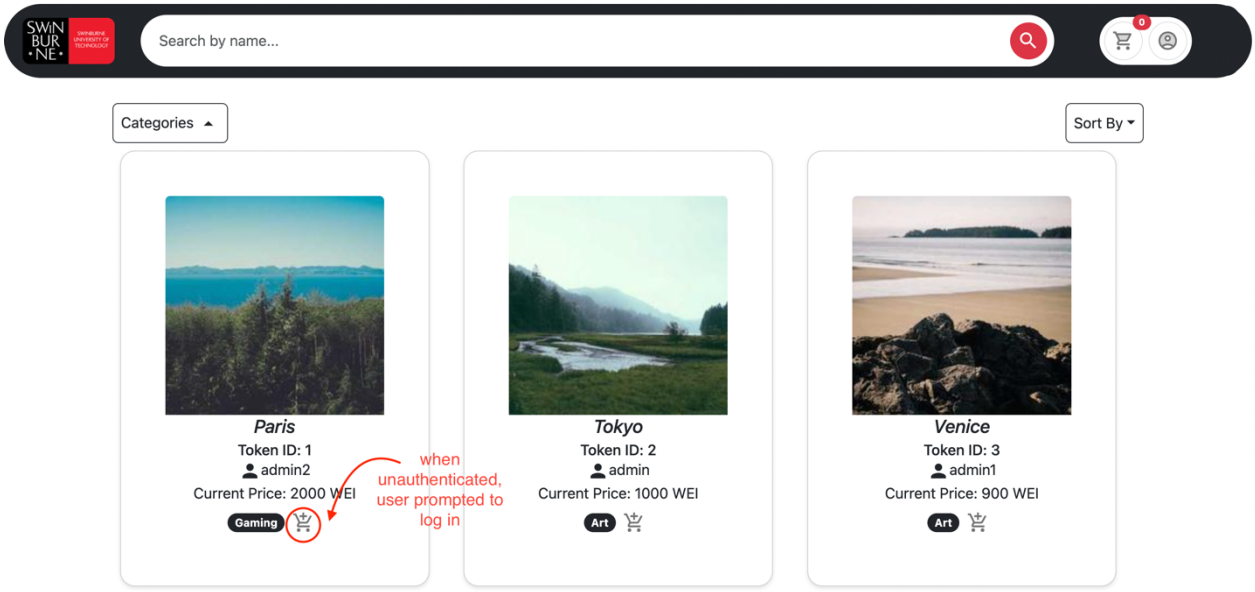

Fig 15. Item card

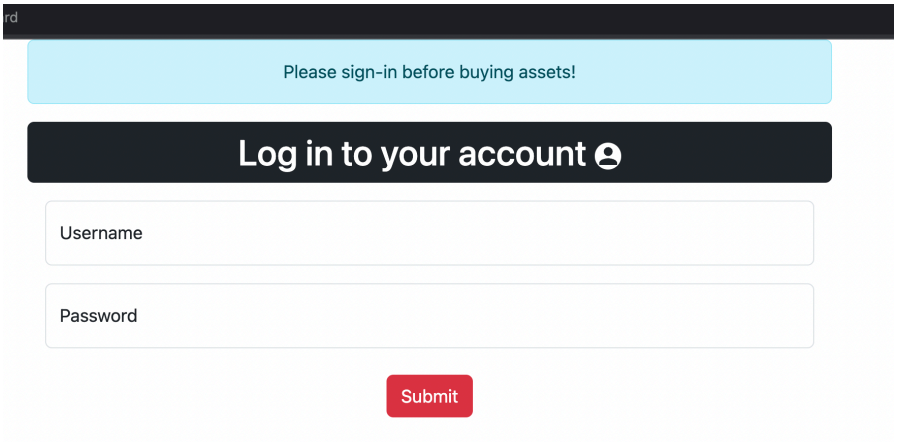

Fig 16. Log in page

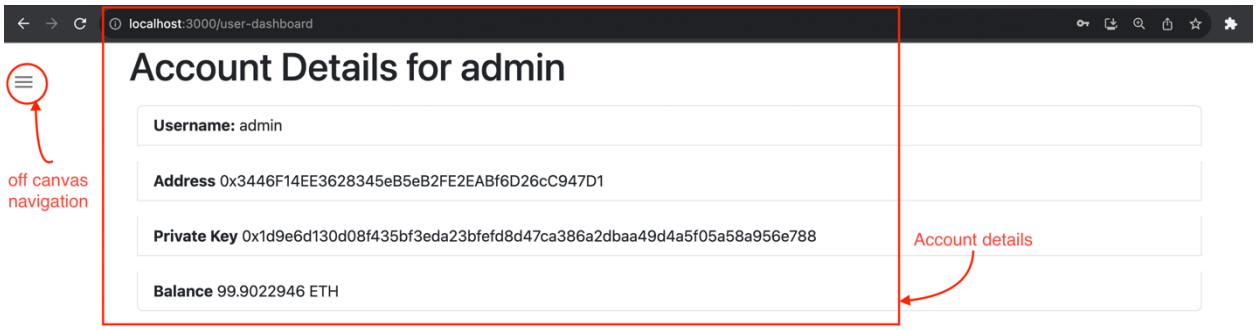

Fig 17. Account Details page

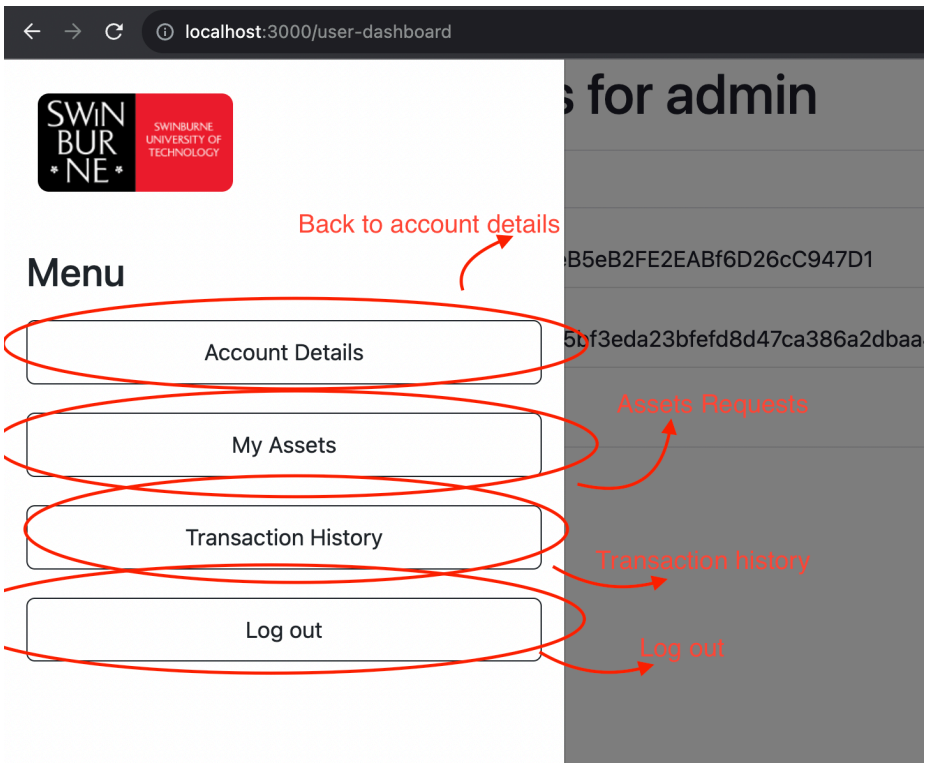

Fig 18. Menu

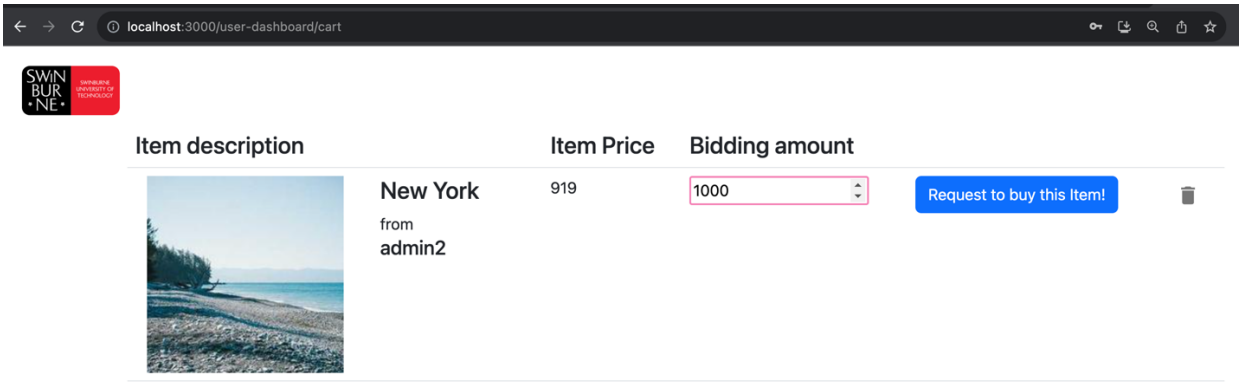

Fig 19. Bidding page

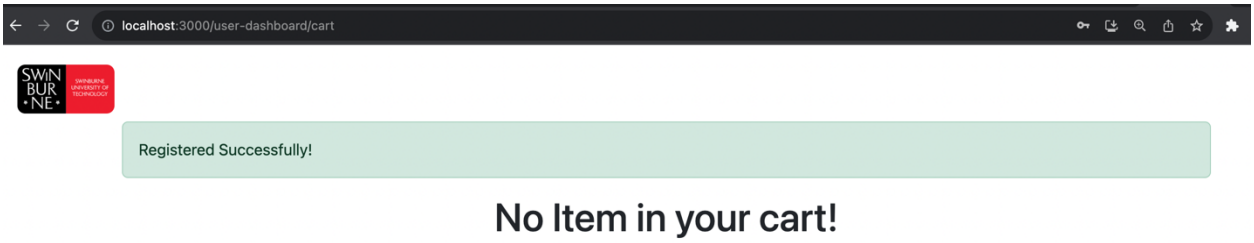

Fig 20. Send Request Successfully

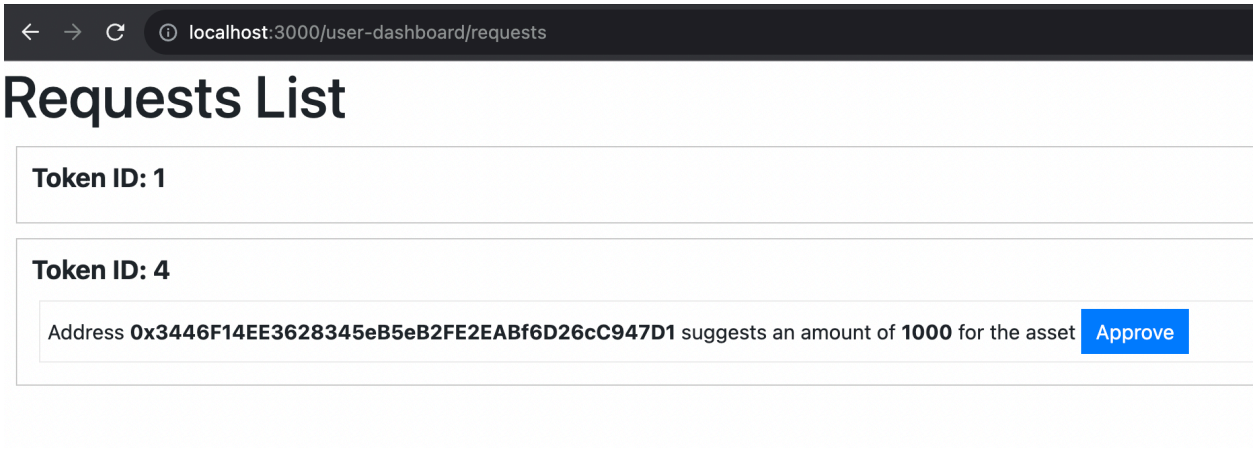

Fig 21. My Asset List and Request List

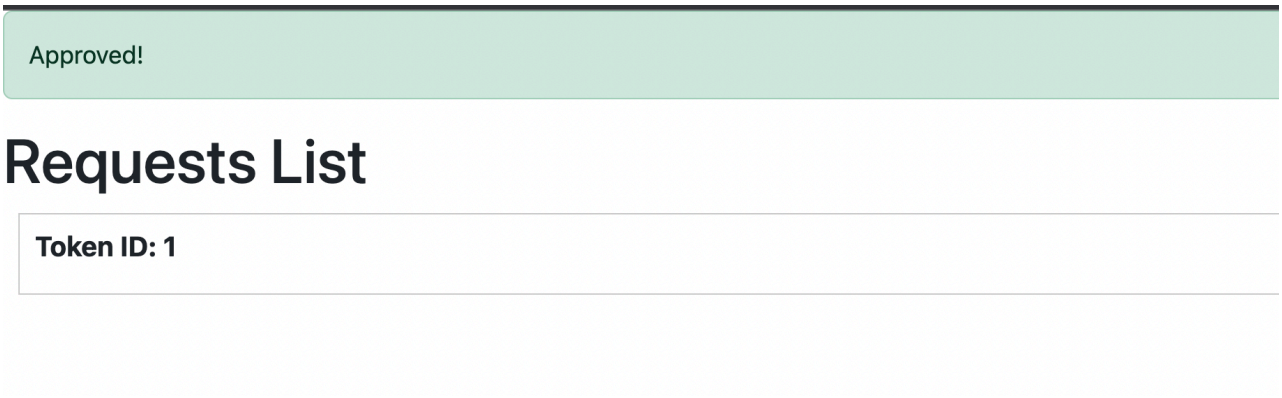

Fig 22. Approved message

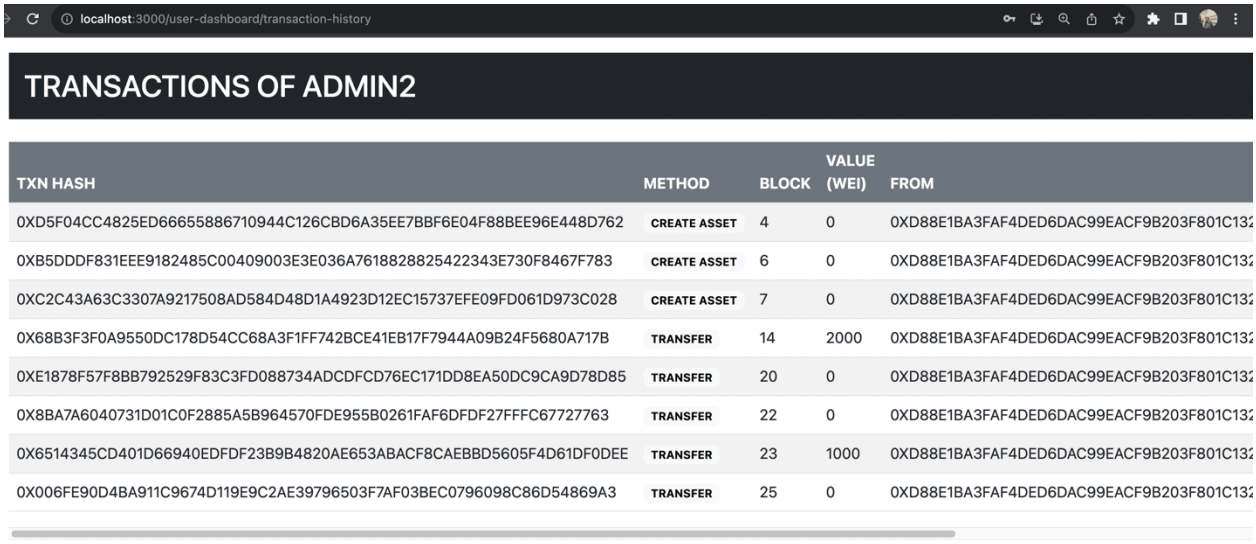

Fig 23. Transaction History

Our website incorporates thoughtfully crafted messages to ensure a positive user experience across various situations, including requesting sign-in, requesting higher bidding amounts, preventing purchase self-item, notifying log-in failures, approving and requesting successfully. The following screenshots illustrate the messages:

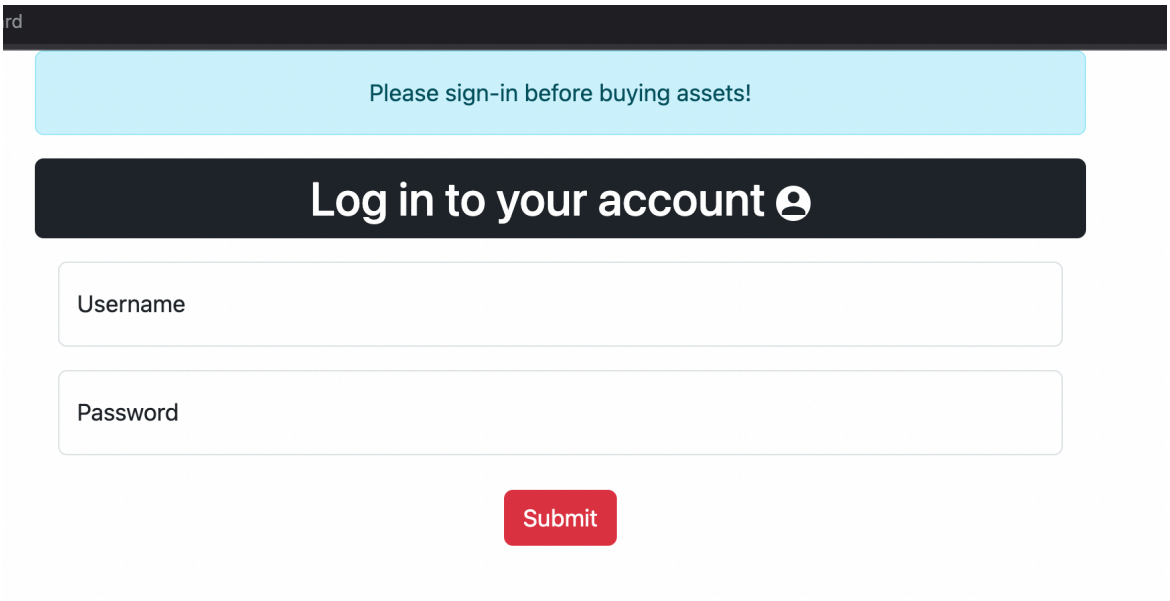

Fig 24. Request Log In

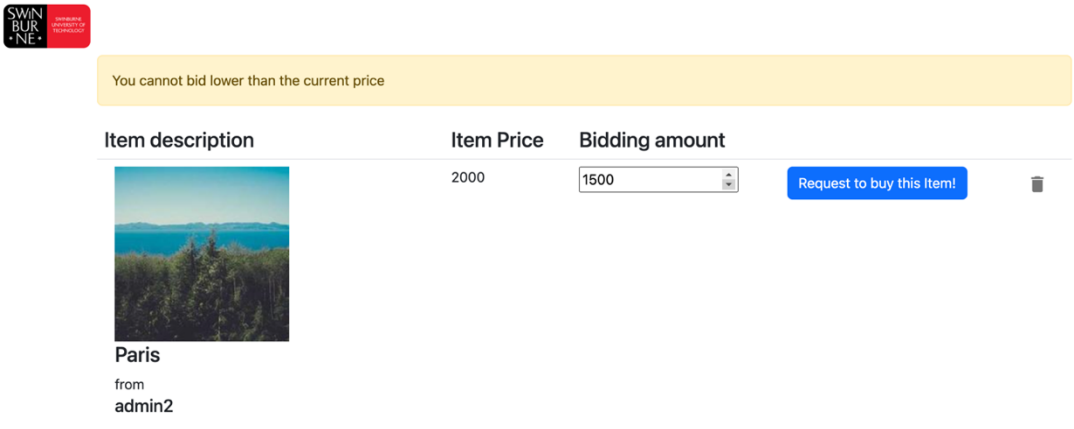

Fig 25. Request bid with higher price message

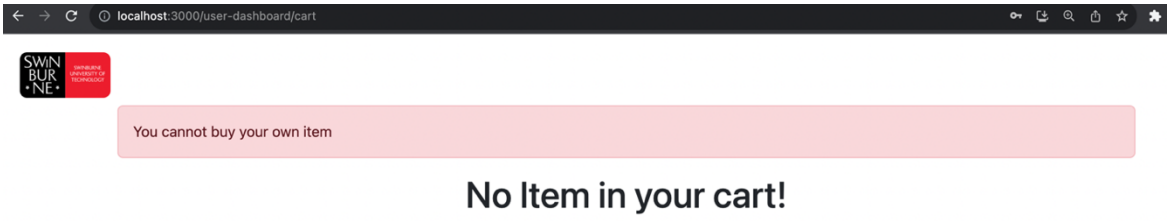

Fig 26. Cannot buy your own asset message

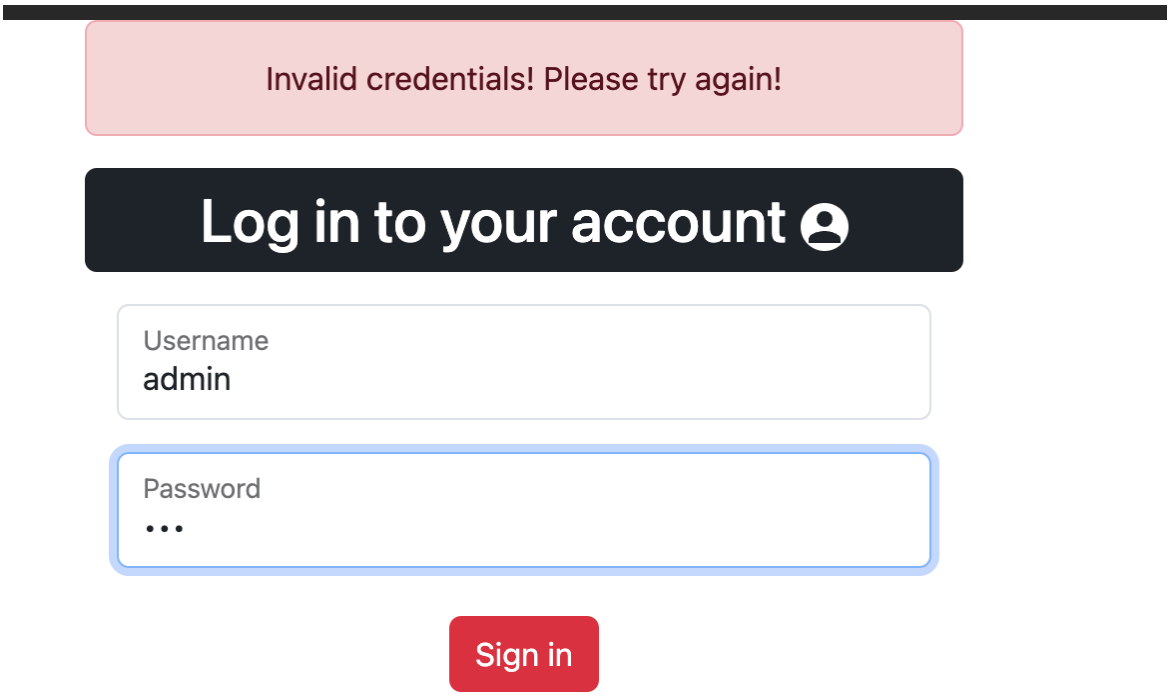

Fig 27. Log in failed message

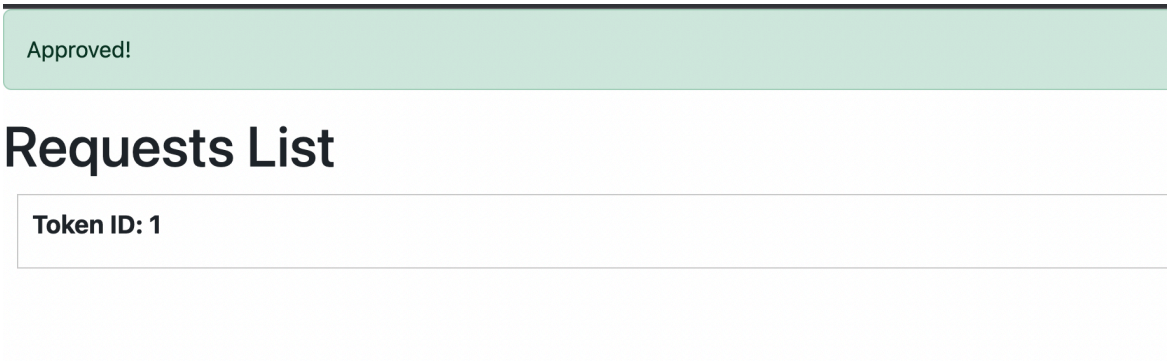

Fig 28. Approve successfully message

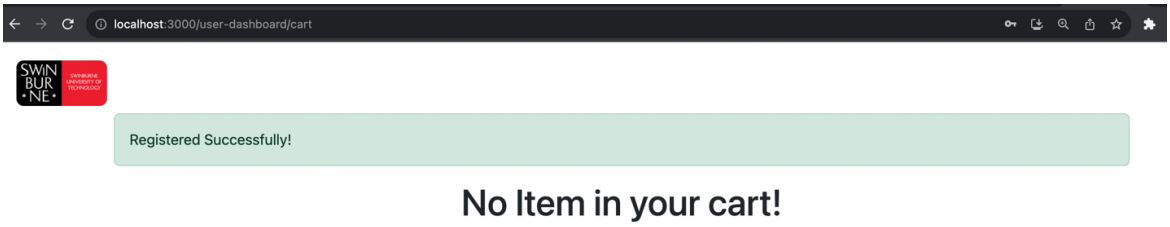

Fig 29. Register successfully message

Details of functionalities

1. **Function Name: View Available Assets**

Purpose: This function allows users to browse all available assets to purchase, which enables users to compare options, making the website more efficient and user-friendly.

Use cases:

Step 1: Open the website

2. **Function Name: Apply filtering/sorting/searching**

Purpose: The purpose of filtering, sorting, and searching features is to enhance users' shopping experience by enabling them to efficiently refine their asset searches based on specific criteria and easily locate assets within their desired categories, which helps match their preferences and needs.

Use cases:

Step 1: Open the website

Step 2:

- For filtering by categories: Click the Categories button
- For sorting by price: Click the Price menu
- For searching by keywords: Click the search bar

Step 3:

- For filtering by Categories: Click one of the category tabs -> display filtered assets
- For sorting by price: Click one of the criteria -> display sorted assets
- For searching by keywords: Input the keywords and click on the search button -> display matched assets

3. **Function Name: Request to buy**

Purpose: This function aims to allow users to participate in the bid by invoking the "registerToBuy()" function of the smart contract and provide the amount they are willing to pay for an asset.

Use cases:

Step 1: Open the website

Step 2: Log in

Step 3: Click the shopping cart icon of an asset

Step 4: Input a bidding value

Step 5: Click 'Request to buy this Item!'

4. **Function Name: Approve a request**

Purpose: the function aims to allow the user to approve a request to purchase an asset owned by this account.

Use Case:

Step 1: Open the website

Step 2: Log in

Step 3: Click the user profile

Step 4: Click the menu

Step 5: Click the item "My Assets" in the menu

Step 6: Click the approve button of the chosen request

5. **Function Name: View account details**

Purpose: This function allows users to access their information like address, private key and balance.

Use Case:

Step 1: Open the website

Step 2: Log in
Step 3: Click the user profile icon
Step 4: Click the menu
Step 5: Click the “Account Details” item in the menu

6. **Function Name: View transaction history**

Purpose: This function allows users to view their transaction history including transaction hash, value, transaction ‘from’ and address that are retrieved from the blockchain.
Use Case:
Step 1: Open the website
Step 2: Log in
Step 3: Click the user profile icon
Step 4: Click the menu
Step 5: Click the “Transaction History” item in the menu

7. **Function Name: View requests to buy assets (view current assets)**

Purpose: This function allows users to view all the assets that they own and view requests for their assets.
Use Case:
Step 1: Open the website
Step 2: Log in
Step 3: Click the user profile
Step 4: Click the menu
Step 5: Click “My Asset” item in the menu

8. **Function Name: Login**

Purpose: This function allows users to log in to their account to view, request to buy assets or approve requests.
Use Case:
Step 1: Open the website
Step 2: Click the user profile icon
Step 3: Input username and password
Step 4: Click in

9. **Function Name: Log out**

Purpose: This function allows users to log out.
Use Case:
Step 1: Open the website
Step 2: Click the user profile icon
Step 3: Click the menu
Step 4: Click “Log Out” item in the menu

4.6 Project Deployment Instructions

1. Download the source code
1. Install dependencies for the front-end code

Using the command line to access the front-end project, then execute **npm install** in the front-end project to install all the node.js dependencies.

2. Install dependencies for the back-end code

Ensure the local machine has installed Python and pip

- Install python: https://www.python.org/downloads/
- Install pip: https://pip.pypa.io/en/stable/installation/

Use **pip install** or **pip3 install** to install the essential packages. Some notable ones can be named as:

- Fastapi & Uvicorn: to build fastapi server
- Mysql-connector-python: to connect and query the database
- Py-solc-x: to compile smart contracts
- Web3: to interact with the Blockchain server

Recommended method: Using PyCharm as the IDE for the backend.

Create a Python interpreter (Virtualenv Environment recommended)

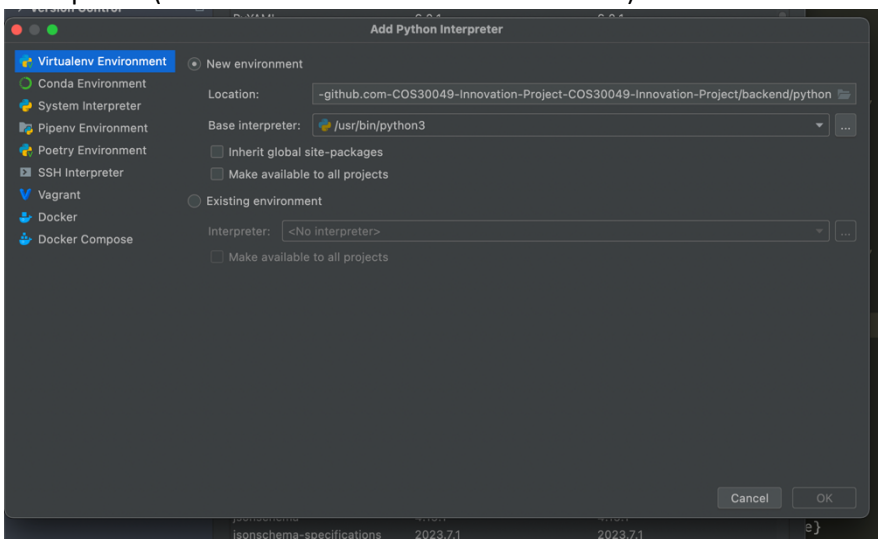

Fig 30. Create a Python interpreter

Create a fast API configuration and attach the Python interpreter to it. The packages installed to run the back end are as follows:

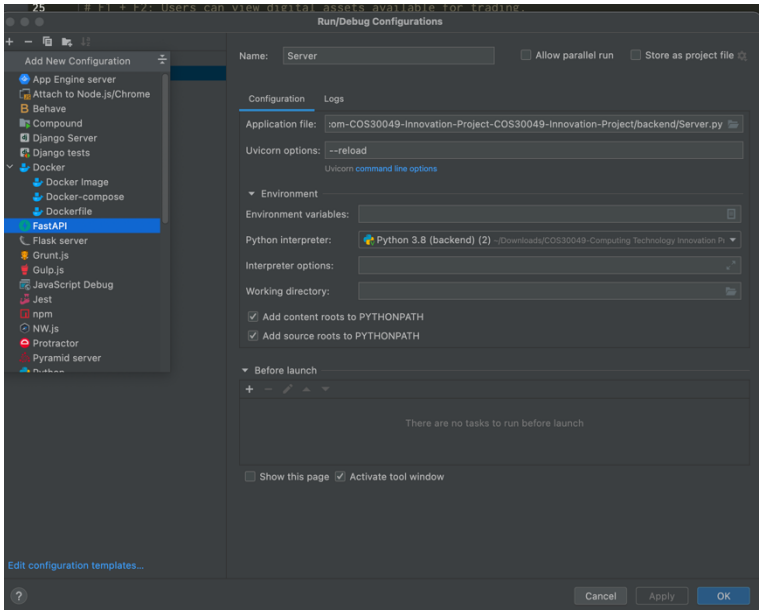

Fig 31. Create fast API configurations

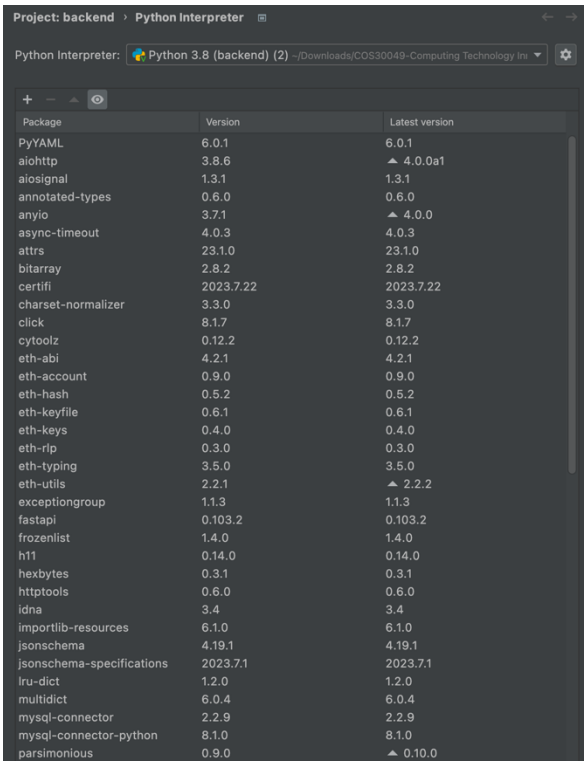

Fig 32. required packages

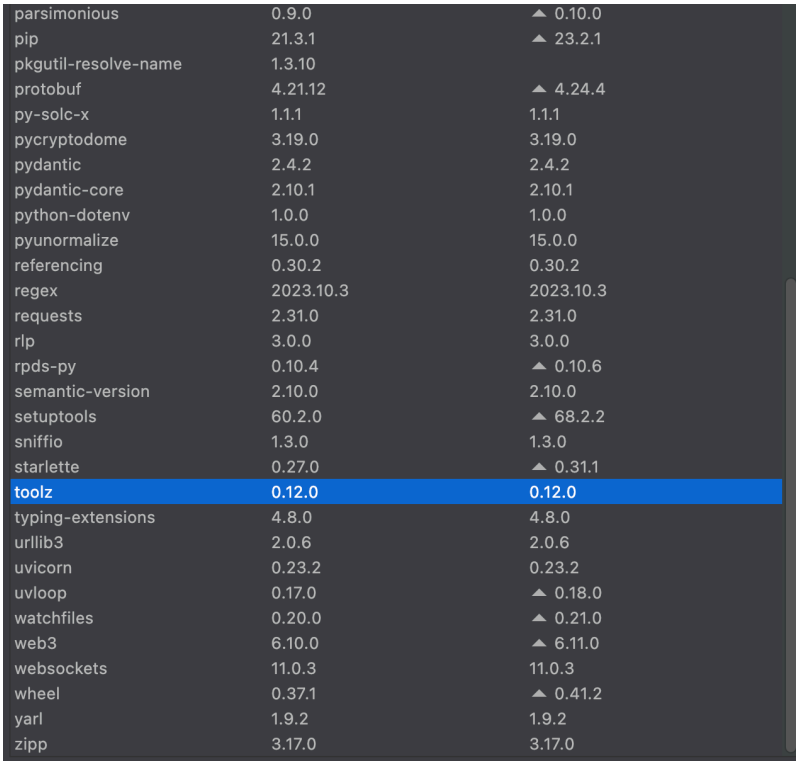

Fig 33. Required packages (2)

3. Install a MySQL database (MAMP on Mac)

Install MAMP for Mac (also Windows) to use its database the services or use Swinburne database on the Mercury server

Link: https://www.mamp.info/en/downloads/

Start MAMP to start the database server. A database schema export is provided in the .zip file.

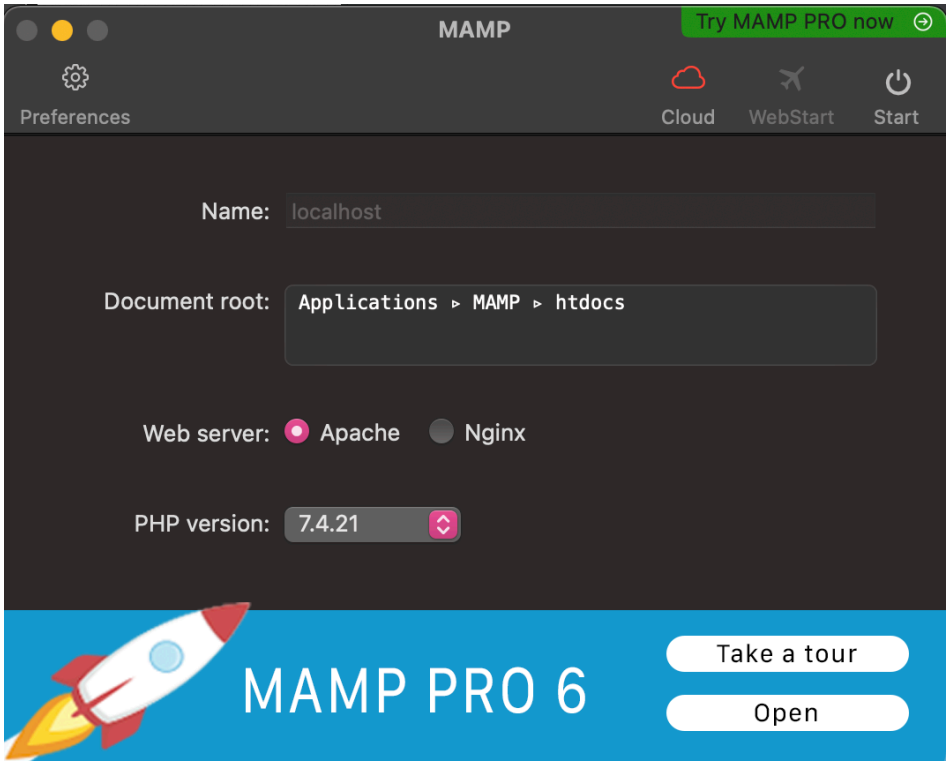

Fig 34. MAMP interface (click on the button start to start the server)

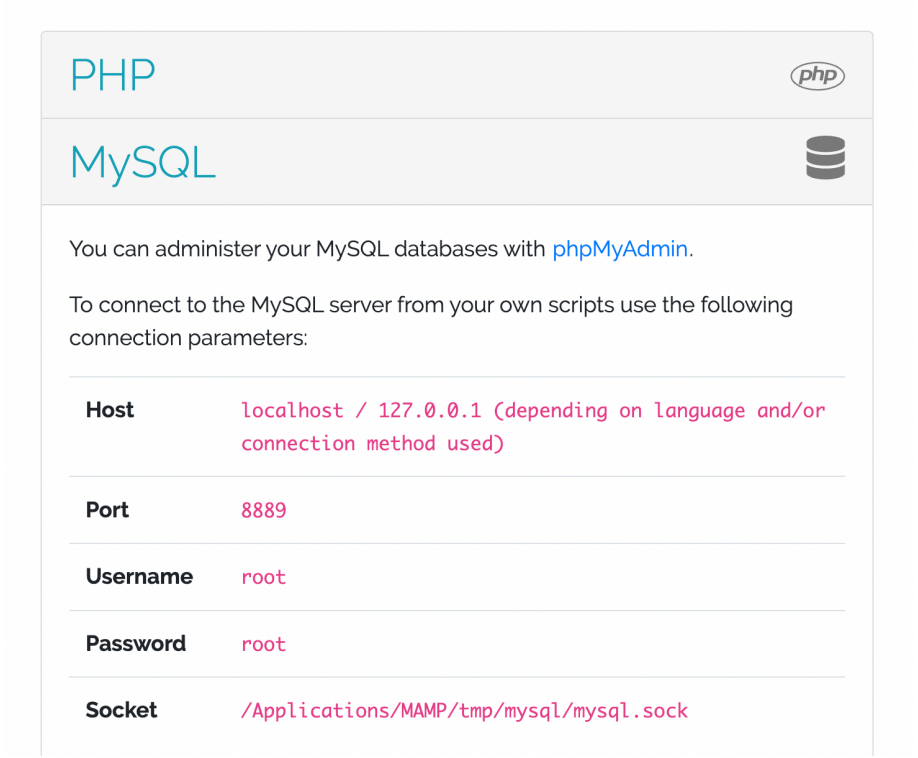

Fig 35. MAMP database server config

4. Install a blockchain server

Install Ganache (recommended): https://trufflesuite.com/ganache/

When finished and opened, click on Quick Start and save the server info

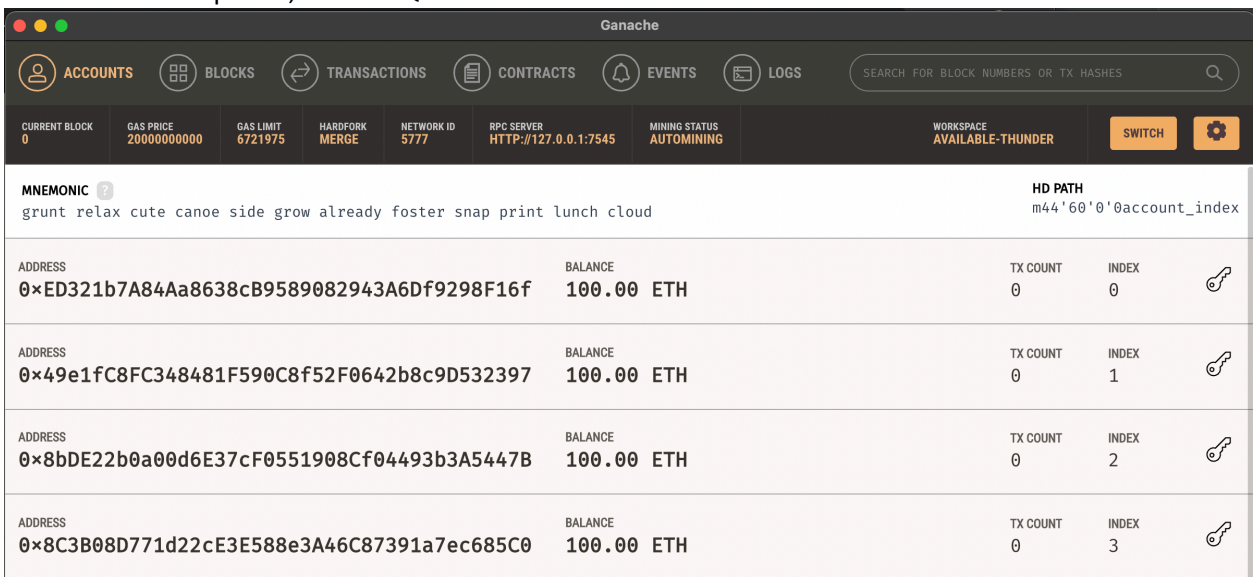

Fig 36. Ganache server

5. Generate sample data

Remember to open Ganache and MAMP server

In the backend project, navigate to Database.py, and look for the generateSampleUser() function. Replace the existing user records with your Ganache address and private key (for example, what has been shown in step 5).

```
def generateSampleUsers():
    dtb = Database("localhost", "root", "root", "COS30049")
    user = User("admin", "admin",
                "0x3446F14EE3628345eB5eB2FE2EABf6D26cC947D1", #address
                "0x1d9e6d130d08f435bf3eda23bfefd8d47ca386a2dbaa49d4a5f05a58a956e788") #private key
    user1 = User("admin1", "admin1",
                 "0xcAC3D4DeB1B462C1D8873cc09Fd7648339A68371",
                 "0x3f5067fc8daefe53ee65c80329e2459ae5b504b28c8e210c68f04d247262ab68")
    user2 = User("admin2", "admin2",
                 "0xd88e1bA3fAF4DEd6dAC99EaCf9b203F801c13291",
                 "0x5565aa59492f3603c4fb22abcf0eb6895f09acbad328a258fcd738ca71780b30")
    dtb.addUser(user)
    dtb.addUser(user1)
    dtb.addUser(user2)

# generateSampleUsers()
```

Fig 37. generateSampleUsers() method

Uncomment or write the function & run the Database.py file

In the backend project, navigate to IntegrationW3AndDatabase.py, and look for the generateSampleData() function (ensure you have finished with the previous step).

Uncomment or write the function & run the IntegrationW3AndDatabase.py file

```
def generateSampleData():
    # Define the list of random owners, categories, and quotes for description
    owners = ["admin", "admin1", "admin2"]
    random_location_names = [
        "Paris",
        "Tokyo",
        "Venice",
        "New York",
        "Sydney",
        "Rio de Janeiro",
        "Cairo",
        "Barcelona",
        "Dubai",
    ]
    categories = ["Art",
                  "Gaming",
                  "Membership",
                  "PFPs",
                  "Photography",
                  "Music"]
    quotes = [
        "The only way to do great work is to love what you do. - Steve Jobs",
        "In the middle of every difficulty lies opportunity. - Albert Einstein",
        "Life is what happens when you're busy making other plans. - John Lennon",
        "Success is not final, failure is not fatal: It is the courage to continue that counts. - Winston Churchill",
        "The best way to predict the future is to create it. - Peter Drucker",
```

Fig 38. generateSampleData() method

6. Start the backend server

Click on the play button near the fast API configurations in PyCharm IDE or uvicorn Server:app -–reload using the command line

7. Look for differences in the URL to communicate with the frontend server and start the frontend server

Be aware of the port running on the backend server. The default is 8000. However, if that is not the case, replace the port number with the correct one (when starting the backend server). The API calls exist in those .jsx files: AccountDetails.jsx, FilterComponent.jsx, Main.jsx, RequestList.jsx, SearchBar.jsx, ShoppingItem.jsx, SignIn.jsx, TransactionHistory.jsx.

Navigate to the frontend project, run npm start to start the frontend server

8. Open the web browser and log in with the credentials you created to run the website

Link: http://localhost:3000

5 Reference

MAMP. MAMP Downloads. [Online] Available at: https://www.mamp.info/en/downloads/ (Accessed: 15 October 2023).

Python Packaging Authority. pip Installation Guide. [Online] Available at: https://pip.pypa.io/en/stable/installation/ (Accessed: 15 October 2023).

Python Software Foundation. Python Downloads. [Online] Available at: https://www.python.org/downloads/ (Accessed: 15 October 2023).

Truffle Suite. Ganache with Truffle. [Online] Available at: https://trufflesuite.com/ganache/ (Accessed: 15 October 2023).