The Hong Kong Polytechnic University

Department of Computing

Security Analysis and Application For NFT Market

COMP3334 Computer Security

Group 11

Name:	Student ID:
JIANG Yiyang	21095707d
WANG Lili	20093979d
WANG Zihan	20074893d
XIONG Yifan	20078998d

April 2023

Contents

Abstract. With the development of blockchain technology in recent years, people gradually have a clearer understanding of the concept of "virtual assets". In this context, NFT (Non-Fungible Token) was born, which gives digital artworks and other virtual assets unique properties through cryptography, making them scarce and collectible, satisfying the pursuit of uniqueness and ownership in the digital world. On the NFT trading platform, people can securely buy, own, and trade unique digital artworks and virtual items, enhancing the protection of creators' rights and interests in the digital realm, while also providing investors with a new investment channel. the rise of NFT has driven the rapid development of digital art, games, collectibles, and other fields, changing traditional trading and collecting methods.

In this group project for COMP3334 Computer Systems Security course, we will apply the cryptographic knowledge we learned in class (e.g., RSA, AES, etc.) to develop a platform that simulates the trading of virtual digital artworks. In this report, we will explain the security analysis of the platform we developed, the encryption algorithms used in the process, the difficulties encountered during the development process, and the solutions.

1 Security requirement analysis with justifications

1.1 C.I.A Model Analysis

We analyze in this section the security requirements for NFT trading platforms. the C.I.A model will be applied, with specific justifications, and several recommendations for security mechanisms to be used in practice.

1.1.1 Confidentiality

As a trading platform for virtual artworks, it is necessary to keep some personal information of users. In order to provide our services, we need to keep the following information:

- User data: Protects users' personal information, such as usernames, passwords, email addresses, phone numbers, etc., from unauthorized access and leakage.
- Transaction data: Ensure that transaction details are visible only to transaction participants and authorized persons, e.g., by using encryption technology to protect the data.
- Artworks: Use digital copyright protection measures to prevent unauthorized copying and distribution.

1.1.2 Integrity

For artworks stored on the platform, it is crucial to ensure that the artwork is not tampered with to ensure its uniqueness. In addition to this, we need to ensure the integrity of many pieces of information:

- Digital artwork. The core of a digital artwork trading platform is to ensure the inerrancy and integrity of a digital artwork, which safeguards the value of that digital artwork. In our practice, we can use distributed storage, sophisticated encrypted means of transportation and storage to guarantee its integrity and tamper-evident nature.

- Transaction records: Transaction records are closely related to the movement of user assets, so we need to safeguard the integrity of transaction records. Blockchain technology can be used to ensure the integrity of transaction records using a distributed ledger.

1.1.3 Availability

The availability of the platform is critical to NFT's virtual digital artwork trading platform. High usability ensures that users can access the platform to trade and browse at any time, enhancing user experience and satisfaction. At the same time, availability is critical to maintaining the platform's reputation and trust. If the platform is frequently down or inaccessible, users may lose confidence and switch to other competitors. In addition, high availability helps protect against potential security threats, such as distributed denial-of-service attacks, ensuring that the platform operates properly and safeguarding user assets. Therefore, guaranteeing platform availability is important to ensure business continuity, improve customer satisfaction and protect corporate reputation.

- System availability: In order to ensure that the platform has enough bounce to handle peak access, the platform needs to guarantee its system availability. In our practice, we can use means such as traffic control to ensure the resilience of the system. At the same time, we need to keep an eye out for malicious attacks. Therefore, it is necessary to deploy anti-DDoS attacks, turn on access blacklist, and other measures for the platform.

- Data availability: Data is exceptionally important to our platform, and there is a high probability that unavailability of data will prevent us from serving our users properly. We need to take regular backups, distributed storage and other means to ensure the availability of data.

1.2 Overall security objectives

- Ensure the privacy and data security of platform users

- Guarantee the integrity, reliability, and traceability of the transaction process

- Prevent unauthorized access and operation

- Ensure the stability and availability of the platform

1.3 Security requirements breakdown

1. Communication encryption:
    - Use RSA+AES encryption technology to encrypt the communication between the client and the server to ensure the security of the data in the transmission process.
    - Use SSL/TLS protocol to ensure the integrity and confidentiality of the communication process.
2. Transaction process security:
    - Digital signature of transactions to ensure non-repudiation and traceability of transactions.
    - Transactions use virtual currency and transaction information is packaged and stored using blockchain technology, ensuring that transactions are not tampered.
3. Data Security
    - Encrypted storage of sensitive user data (e.g., password).
    - Regularly backup and update data to prevent data loss or corruption

2 A literature review of similar systems in the market

2.1 OpenSea

OpenSea is the world's largest decentralized non-homogenized token (NFT) marketplace, providing a platform for users to trade digital artwork, virtual collectibles, gaming items, and various other non-homogenized tokens. Since its inception in 2018, OpenSea has continued to grow and attract many artists, collectors, and gaming enthusiasts.

Decentralized Trading: OpenSea uses a decentralized trading approach to ensure that buyers and sellers can trade directly, reducing the cost and risk of intermediaries.

Smart Contracts: By using smart contract technology, OpenSea ensures that the transaction process is automated, safe and secure. This technology can effectively prevent fraud and increase the transparency of transactions.

Cross-chain support: OpenSea supports multiple blockchain networks, such as Ether, Polygon, etc., allowing users to easily trade NFT between different networks.

Digital wallet integration: Integration with popular digital wallets (such as MetaMask, Trust Wallet, etc.) makes it easy for users to manage and trade cryptocurrencies and NFT.

2.2 Axie Infinity

Axie Infinity is an ethereum blockchain-based combat and farming game. The core gameplay of the game is to collect, train, fight and trade virtual creatures "Axie".

Non-Homogenized Token (NFT): Each Axie in Axie Infinity is a unique Non-Homogenized Token (NFT) with unique genetics and appearance. NFT is a token on the blockchain that represents a unique and rare digital asset. NFT technology ensures ownership, rarity, and tradability of Axie. In addition, in-game lands, props, etc. are also NFT, making these in-game assets have real value.

Decentralized Marketplace: Axie Endless has a decentralized marketplace that allows users to trade Axie, lands, and props directly on the blockchain. The decentralized marketplace eliminates intermediaries, reducing transaction costs and increasing transaction efficiency. In addition, users can trade Axie Infinity's game assets through third-party NFT marketplaces such as OpenSea.

These two sites inspired our project:

- Intuitive Interface Architecture: Pertaining to OpenSea's interactional schema, the organization proffers a cognitively accessible and facile modus operandi, enabling users to expeditiously acclimate and pinpoint the virtual artistic creations that pique their interest.
- Disintermediated Exchange: Disintermediated commercial interactions attenuate pecuniary expenditures and potential hazards while concurrently corroborating the indelible nature of financial transactions.
- Asymmetric Cryptographic Techniques: Throughout the TLS/SSL protocol initiation, the consumer and provider engage in asymmetric encryption methodologies (e.g., RSA, ECDSA, etc.) to facilitate authentication and key distribution. Asymmetric encryption paradigms utilize a duo of public and private cryptographic keys for enciphering and deciphering, thus fortifying the secure identification of the respective communication entities.
- Symmetric Cryptographic Procedures: Upon the consummation of the key transfer, data is encoded and decoded reciprocally between the user and the service provider via symmetric encryption algorithms (e.g., AES, ChaCha20, etc.). Symmetric encryption practices employ an identical key for both encipherment and decipherment, engendering efficacious and secure performance.

3 System design specification

3.1 Encryption of communication processes

In our project practice, we adopt high specification and high security AES encryption for the communication between users and servers and encrypt the AES

public key with RSA asymmetric encryption algorithm.

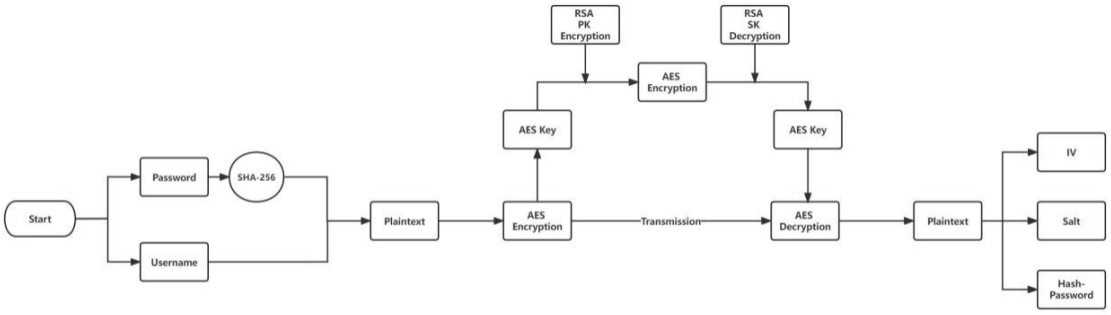

Figure 1: Communication encryption process

As illustrated in the diagram, when a user registers a new account on our website, they are prompted to provide their account username and password. In reality, other essential information such as their name and email address are also entered during this process. However, for the purpose of this explanation, I will focus on the encryption process for the account username and password.

3.2 Security assumptions

Foundational assumptions regarding the security paradigms implemented within non-fungible token (NFT) trading platforms encompass fundamental principles and beliefs aimed at safeguarding the platform, its participants, and their assets. These suppositions form the bedrock for establishing and preserving a secure milieu conducive to NFT transactions. A selection of security presuppositions for an NFT trading platform may encompass:

1. Dependable Infrastructure: The platform postulates that the foundational blockchain technology, exemplified by Ethereum or Binance Smart Chain, is impervious and resistant to tampering, thereby establishing a trustworthy substrate for NFT commerce.

2. Impenetrable Smart Contracts: The platform posits that the smart contracts employed for NFT generation, exchange, and proprietorship administration are devoid of exploitable vulnerabilities, having undergone comprehensive audits conducted by external cybersecurity specialists.

3. Rigorous Authentication: The platform presupposes that users disclose their authentic identities and fortify their accounts with robust, distinctive passwords and/or multi-factor authentication (MFA) to preclude unsanctioned access.

4. Secure Digital Wallets: The platform presumes that users' cryptocurrency wallets, irrespective of being custodial or non-custodial, are safeguarded with appropriate cryptographic techniques and private key administration, mitigating the likelihood of illicit access or misappropriation.

5. Robust Encryption: The platform assumes that data exchange between users and the platform is encrypted utilizing potent, industry-standard cryptographic methodologies to shield confidential information from interception by nefarious entities.

6. Unceasing Surveillance: The platform envisions that real-time monitoring and intrusion detection mechanisms are operational to detect and counteract any dubious activities or security infringements.

7. User Enlightenment: The platform presumes that furnishing users with informative resources on security best practices will facilitate risk reduction associated with NFT trading activities.

3.3 Authentication and authorization

Authentication and authorization are critical components of a secure NFT trading platform. These processes ensure that users can securely access their accounts and that they are granted appropriate permissions for actions within the platform. Here's an explanation of each aspect:

- Authentication:
  - Our NFT website authentication method involves encrypting the user's username before sending it to the server, where it is subsequently decrypted. To ensure information security, you utilize Advanced Encryption Standard (AES) and Rivest-Shamir-Adleman (RSA) cryptographic algorithms.[1]
  - AES is a symmetric encryption algorithm, meaning it uses the same key for encryption and decryption. It is widely recognized for its speed and efficiency, making it suitable for handling sensitive information in various applications.
  - On the other hand, RSA is an asymmetric encryption algorithm that relies on two distinct keys, a public key for encryption and a private key for decryption. This approach is beneficial when securely transmitting data over public networks, as the public key can be openly shared while the private key remains secret to the recipient.
  - By combining AES and RSA encryption techniques, our website's authentication process provides an additional layer of security for sensitive user information. This approach helps protect user data from being intercepted or tampered with during transmission and ensuring that only the intended recipient can decrypt and process the information.

- Authorization:
  - Once a user has been authenticated, the authorization process determines the level of access and permissions they have within the platform. In an NFT trading platform, this may include:

- Viewing and browsing available NFTs. Creating, minting, or listing new NFTs for sale.
- Purchasing or bidding on NFTs.
- Transferring NFTs between users or wallets. Managing account settings, such as payment methods and personal information.

3.4 One-way encryption

When the user inputs their password and clicks the "submit" button, the front-end employs a one-way Hash function to encrypt the user's password. On the server side, we store the hashed value of the user's password rather than the actual password. This approach ensures that even if the server is compromised and the database is breached, the user's real password remains secure. It is worth noting that during this process, we introduce a "salt" which effectively safeguards against rainbow table and dictionary attacks while increasing the difficulty of decryption. We utilize the SHA-256 hashing algorithm[2], making it virtually impossible for an attacker to reverse-engineer the hashed value and discover the user's actual password. The code for the front-end hash encryption of the password entered by the user is as follows:

Code 1: Front-end hash encryption of the password

```
function hashPassword(password, salt) {
    const passwordWithSalt = password + salt;
    const hashedPassword = CryptoJS.SHA256(passwordWithSalt).
        toString();
    return hashedPassword;
}
```

During the login process, the salt value is sent from the back-end to the front-end. The front-end then combines the user's inputted password with the salt value and hashes it before sending it to the back-end for comparison. This step determines whether the login attempt is successful or not.

3.5 Encrypted transmission

AES (Advanced Encryption Standard) is used to encrypt the data in the process of message encryption. The following is the code implementation:

Code 2: Front-end hash encryption of the password

```
function aesEncrypt(str, aesKey, iv) {
    var key = CryptoJS.enc.Utf8.parse(aesKey);
    var srcs = CryptoJS.enc.Utf8.parse(str);
    var iv = CryptoJS.enc.Utf8.parse(iv);
    var encrypted = CryptoJS.AES.encrypt(srcs, key, {mode: CryptoJS
        .mode.CBC, padding: CryptoJS.pad.Pkcs7, iv: iv});
    return encrypted.ciphertext.toString(CryptoJS.enc.Base64);
}
```

In AES encryption, we have adopted the CBC encryption mode. Although CBC mode requires IV (initial vector), which makes the encryption process more complicated, it has the following advantages compared to other AES encryption modes:

- In CBC mode, the encryption of each data block depends on the encryption result of its previous data block. This means that the ciphertext blocks produced by the same plaintext block at different locations can be very different, which means that it is difficult for a hacker to crack one of the blocks individually.
- In CBC mode, if an error occurs in a block during decryption, it will only affect the decryption result of that block and the subsequent block. This helps to localize the decryption error and prevent the error from spreading to other data blocks.

In this case, both IV and aesKey are 16-bit hexadecimal numbers. Each time we communicate, the IV and aesKey will be generated randomly, which ensures the high randomness of our encryption. The function of randomly generated 16-bit hexadecimal is demonstrated as follows:

Code 3: Front-end hash encryption of the password

```
function randomHex(n) {
    let result = "";
    const characters = "0123456789abcdef";
    const charactersLength = characters.length;
    for (let i = 0; i < n; i++) {
    result += characters.charAt(Math.floor(Math.random() *
        charactersLength));
    }
    return result;
}
```

3.6 RSA encryption

In AES encryption, the aesKey used for encryption and decryption are the same due to the symmetry of the encryption. Therefore, once the message is intercepted during transmission and the hacker learns the AES key, he or she can decrypt the message. Therefore, we also need to RSA the aesKey for asymmetric encryption, so that even if the hacker intercepts the encrypted message and the encrypted aesKey, without the private key of the server, the hacker cannot decrypt the aesKey and get the plain text. In our project, we took 2048-bit RSA key length to ensure the security of encryption.

In the front-end, the code for RSA encryption of aesKey is as follows:

Code 4: Front-end hash encryption of the password

```
const jsEncrypt = new JSEncrypt();
jsEncrypt.setPublicKey(publicKey);
const encryptedAesKey = jsEncrypt.encrypt(aesKey);
```

In the process of user registration, we put the server's public key out in the code of the front-end. The user directly uses the server public key obtained from

the front-end to encrypt the aesKey, and after completing the transmission the server accesses the private key stored locally on the server at the back-end and completes the decryption, and after getting the data entered by the user, it is stored in the database to complete the whole registration process.

4 Back-end decryption

In the back-end, after receiving the encrypted data segment and the aesKey encrypted with RSA public key delivered by the server, the encrypted aesKey should be decrypted first using the RSA private key stored locally in the server, and the core code is as follows:

Code 5: Front-end hash encryption of the password

```
def decryp_aes(aesFileName):
    data = request.get_json()
    encrypted_aes_key = data.get(aesFileName)
    with open("../MyKey/private_key.pem", "rb") as key_file:
        private_key = serialization.load_pem_private_key(
            key_file.read(),
            password=None,
            backend=default_backend()
        )
    encrypted_aes_key = base64.b64decode(encrypted_aes_key)

    # Decrypt the AES key
    # Using PKCS1v15 as padding standard. Actually OAEP is better.
    decrypted_aes_key = private_key.decrypt(
        encrypted_aes_key,
        padding.PKCS1v15()
    ).decode()
    return decrypted_aes_key
```

In this process, it is important to note that the padding protocol used in the front and back ends is the same (PKCS1v15), otherwise, it will lead to decryption failure. In practical use, OAEP padding protocol will be more secure, but for the project of this course, PKCS1v15 padding protocol is also sufficient.

Finally, the server backend decrypts the ciphertext using the decrypted aesKey to get the data entered by the user (note that the password is a hash and not the real value). The core code is as follows:

Code 6: Front-end hash encryption of the password

```
def decrypt(key, encrypted_data, IV):
    unpad = lambda s: s[:-ord(s[len(s) - 1:])]
    key = key.encode('utf-8')
    IV = IV.encode('utf-8')
    data = base64.b64decode(encrypted_data)
    cipher = AES.new(key, AES.MODE_CBC, IV)

    # unpad
    text_decrypted = unpad(cipher.decrypt(data))
    #print(text_decrypted)
```

```
    text_decrypted = text_decrypted.decode('utf8')
    # print(text_decrypted)
    return text_decrypted
```

4.1 Transaction Signature

In the metaverse, signing transactions is one of its core mechanisms. When a user wants to initiate a transaction, he must sign the transaction with his private key, and then transmit the signature and the original transaction information to the server for verification. The server decrypts the signature with that user's public key and compares it with the hash of the original transaction information to verify that the transaction was really initiated by the private key holder. In the context of this mechanism, even if a user's account is stolen, the person who stole the account has no way to initiate a transaction with his private key.

The private key is given to the user at the time of registration and the user is reminded to keep it. In practical applications, users often use hardware wallets or software wallets to save their private keys to avoid the overly tedious process of entering keys.

4.2 Blockchain simulations

The security of data in the metaverse is largely due to blockchain technology. In our group project, we wrote our own code to simulate this process:

1. First, the user initiates a transaction, signs it with his private key and sends it to the server. (Simulating a user broadcasting a transaction)

2. The server verifies the transaction and if the verification is successful, it proves that the transaction was indeed initiated by the user and passes the change transaction to the miner.

3. The miner calculates the random number and hash function until a value that meets the requirements is calculated, and then packages the transaction into the blockchain. (Simulates the miner's Pow process).

Note that in this process, we use different tables in the database to simulate a decentralized blockchain. Each table has the block header of the previous table, containing information such as hash and nonce.

4.3 Cryptocurrency Attempts

Since our wallet is written by ourselves and simulates a blockchain environment, we did not implement a real virtual currency, but at the same time, we built another website for testing cryptocurrencies and we used the Metamask API. It can currently be accessed through nft.imjyy.com.

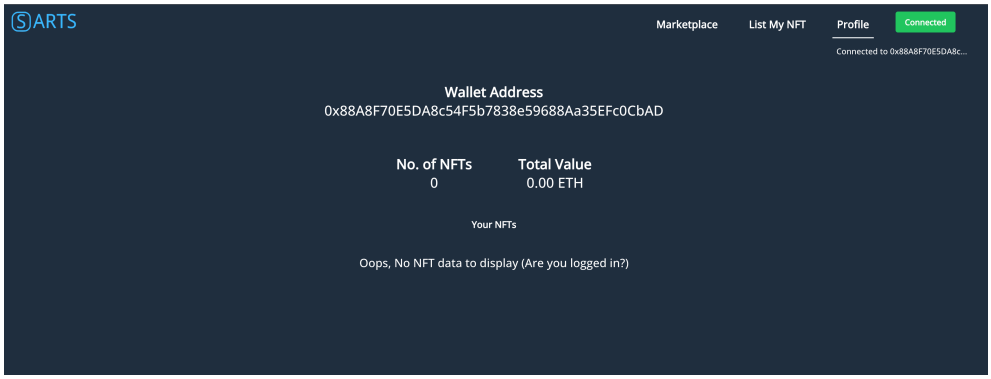

Figure 2: Cryptocurrency Testing

5 System installation guide

5.1 For Users

Visit the website https://3334.imjyy.com/ directly and use it without installation.

5.2 For Developers

The minimum environment and system requirements for installation are as follows:

- linux 4.0, Centos 7.0, Windows 10.
- Python 3.7.

1. Download the original code file to the server
   `git clone https://github.com/fletcherjiang/NFT_Marketplace`
   `cd NFT_Marketplace`

2. Install the appropriate environment
   `pip install requirement.txt`

3. Go to the app directory and modify the content of app.py. If you run locally and on the server, you don't need to modify it, but you need to modify your own SSL certificate to replace your own.
   `vim app.py`

```
config.app.run(host='0.0.0.0', debug=True, port=8080,
    ssl_context=('fullchain.pem', 'privkey.key'))
```

4. Change the database settings in `config.py`
   `vim config.py`

```
app.config['DATABASE'] = {
    'engine': 'peewee.MySQLDatabase',
    'name': 'Your_database_Name',
```

```
    'user': 'Database_user_name',
    'password': '',
    'host': 'localhost',
    'port': 3306
}
```

5. Run `app.py` and open https://127.0.0.1:8000 or Your own URL with browser.

6 Use cases

6.1 Case 1: Sign up and Sign in

- First, register a user called test. (Figure 3)

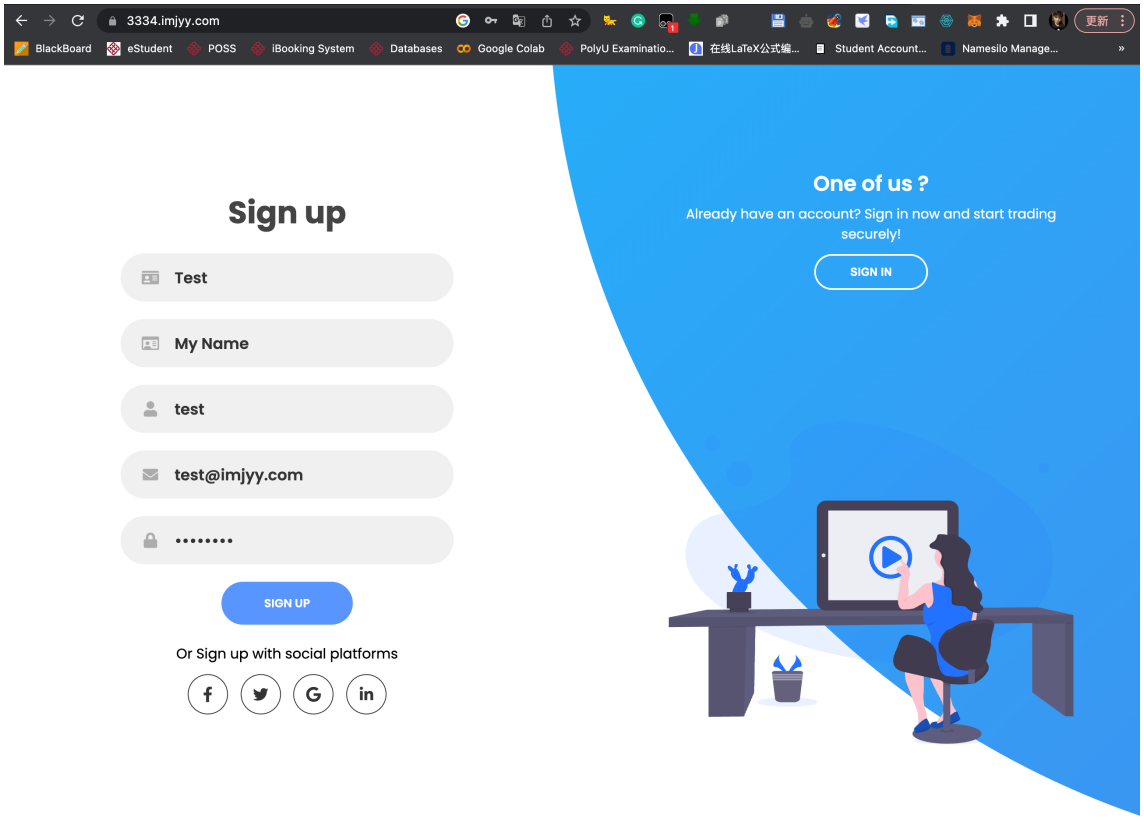

Figure 3: Sign up interface

- After successful registration, the login page is displayed. Users can log in.
- And the user will be prompted to download the PK, and the user will need to use this PK for subsequent transactions. (Figure 4)
- You cannot register duplicate users of the system, such as the same username as well as email address. (Figure 5)
- A successful login will automatically bring you to the main screen. (Figure 6)

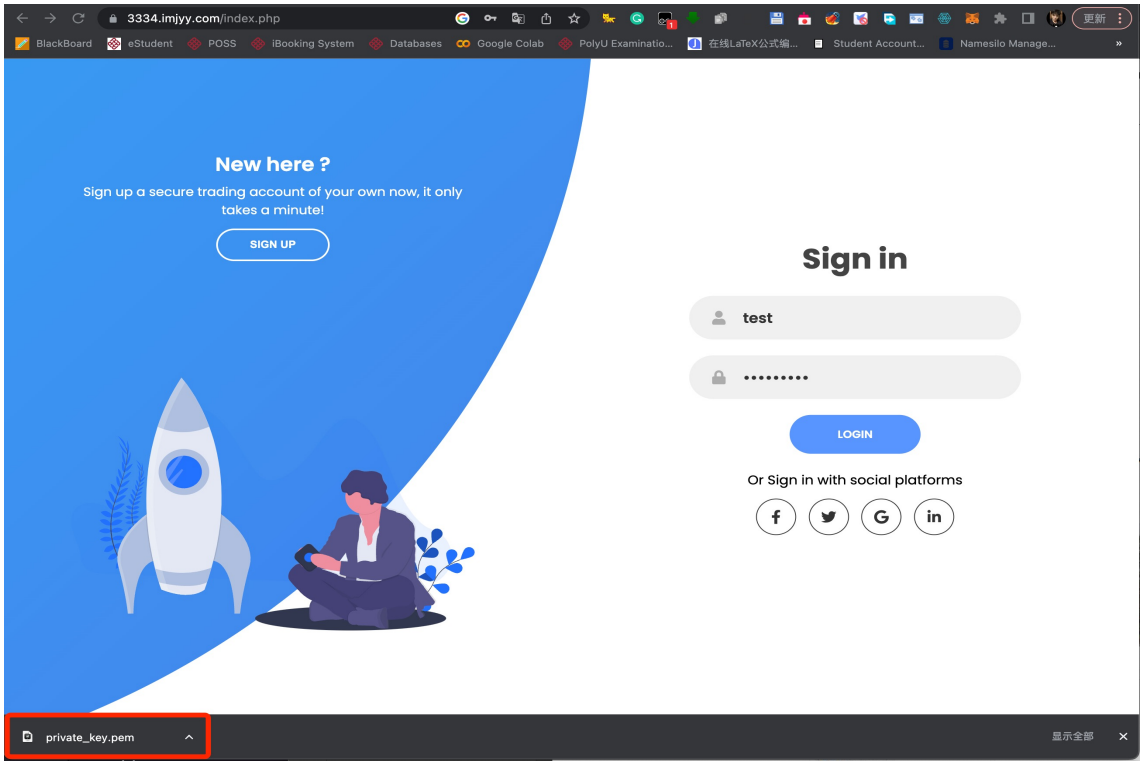

Figure 4: SK download

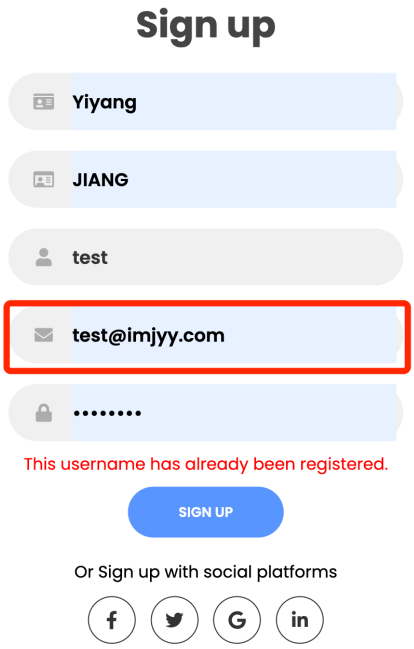

Figure 5: Repeat Registration

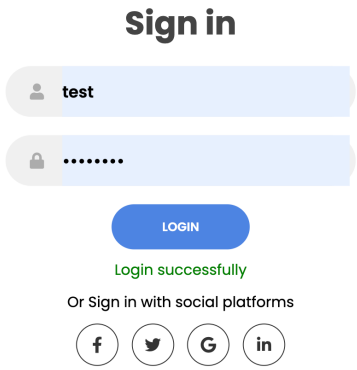

Figure 6: Success interface

- You can log out of your account by clicking log out. Note that if you log out of your account, you need to log in again and the system will delete the user session.

6.2 Case 2: View, buy, upload NFT products

- The main screen provides access to personal information, and product viewing. (Figure 7)

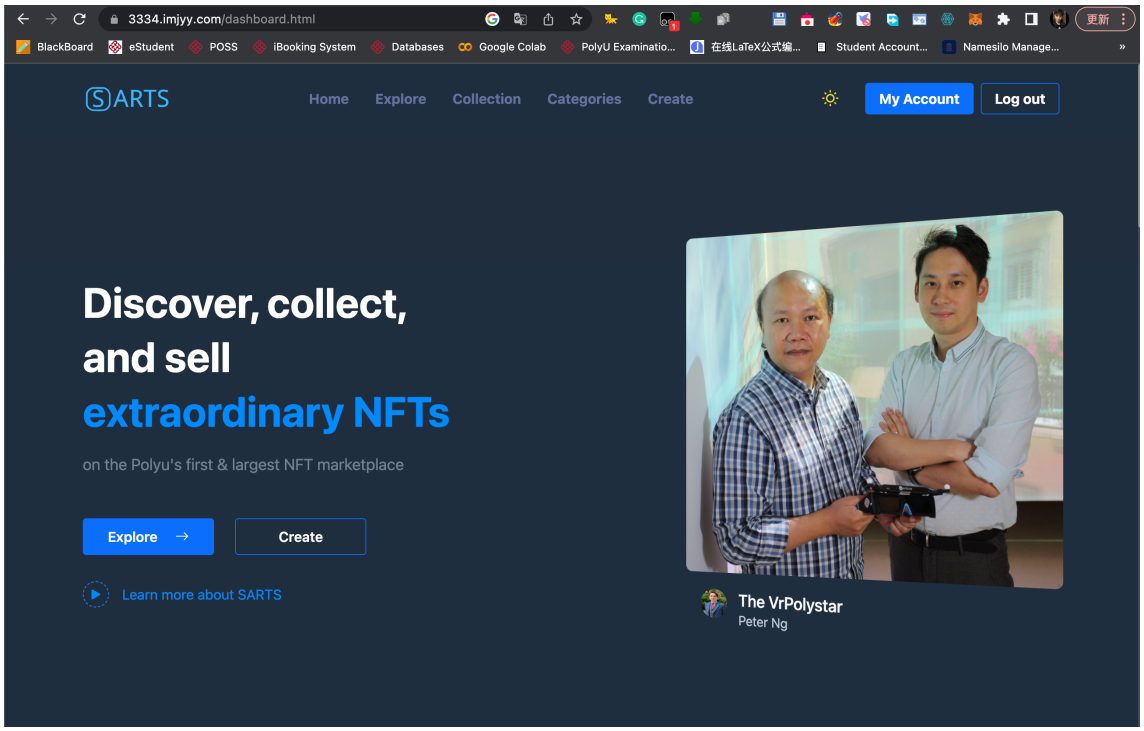

Figure 7: Main User Interface

- In the personal account screen, you can see your balance in the upper right corner, and also in this screen you can see the NFT items you own, as well as the ability to click the upload button to upload and the explore button to view the items. (Figure 8)
- You can choose to view any item and click through to see the details. (Figure 9 and Figure 10)
- The purchase screen requires you to use the previous PK and wait for the purchase. An error message will pop up if the password is wrong. (Figure 11 and Figure 13)
- When we enter our personal screen after successful purchase, we can see that the items we own have been updated.(Figure 13)
- To upload an NFT item, we need to enter all the information and select the image to upload. (Figure 14)

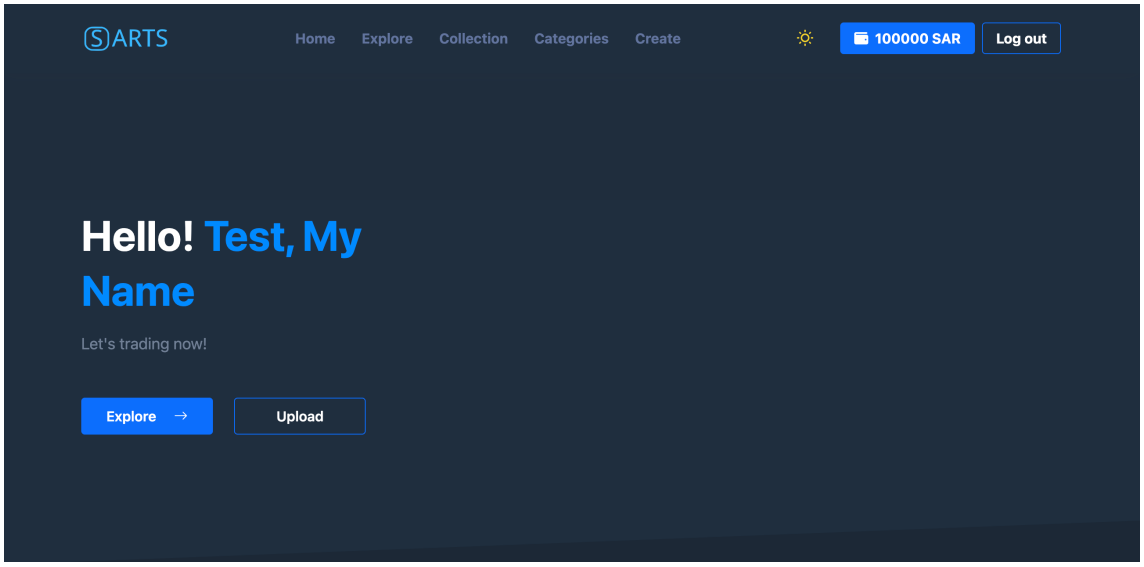

Figure 8: Personal Account Interface

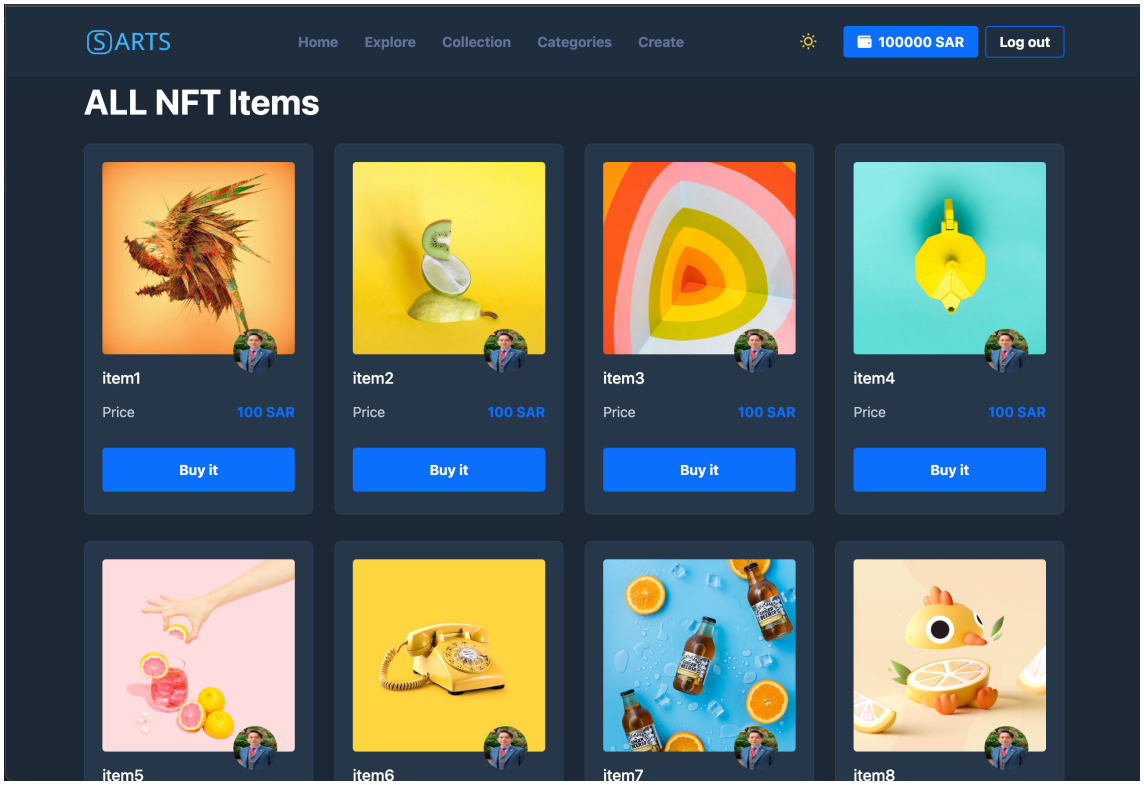

Figure 9: Item Interface

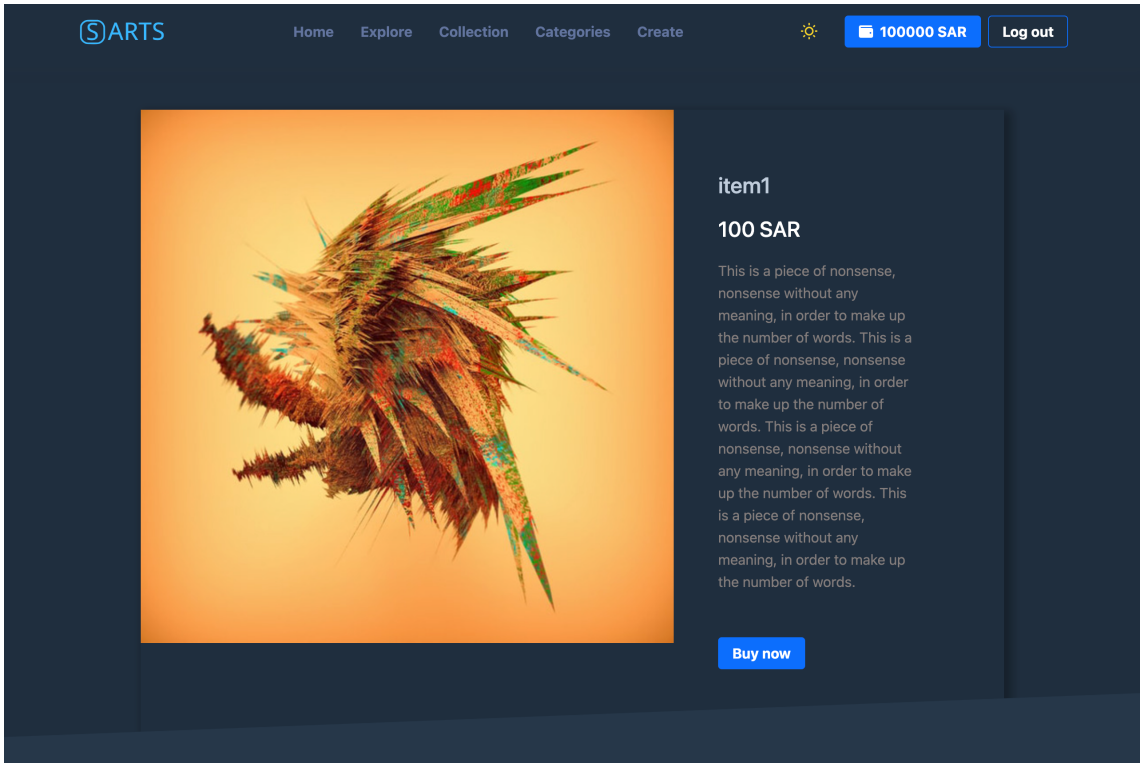

Figure 10: Item Description Interface

- When the upload is successful, we return to the product page and see that our product has been uploaded successfully. (Figure 15)

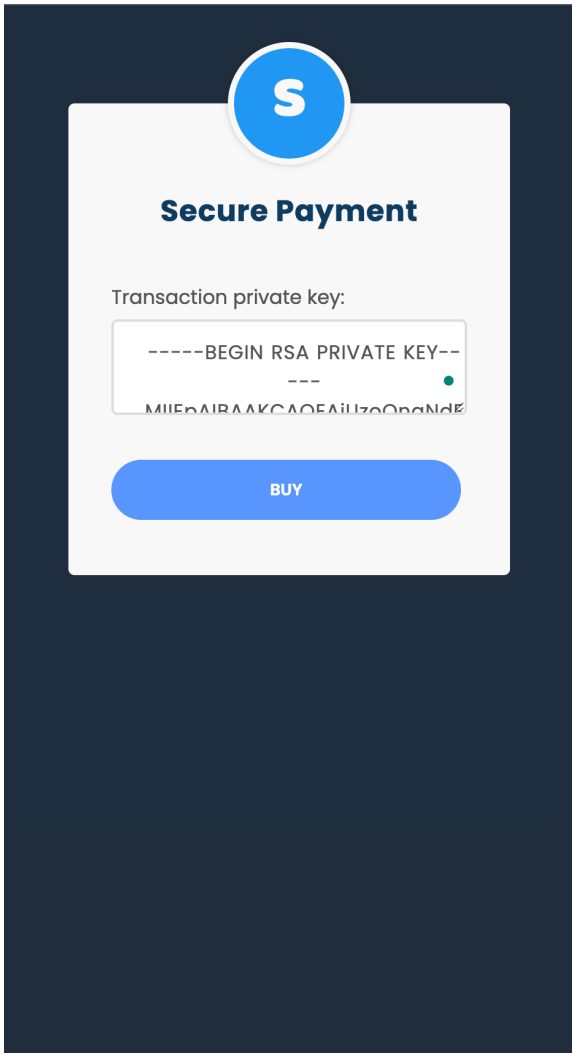

Figure 11: Trading Process

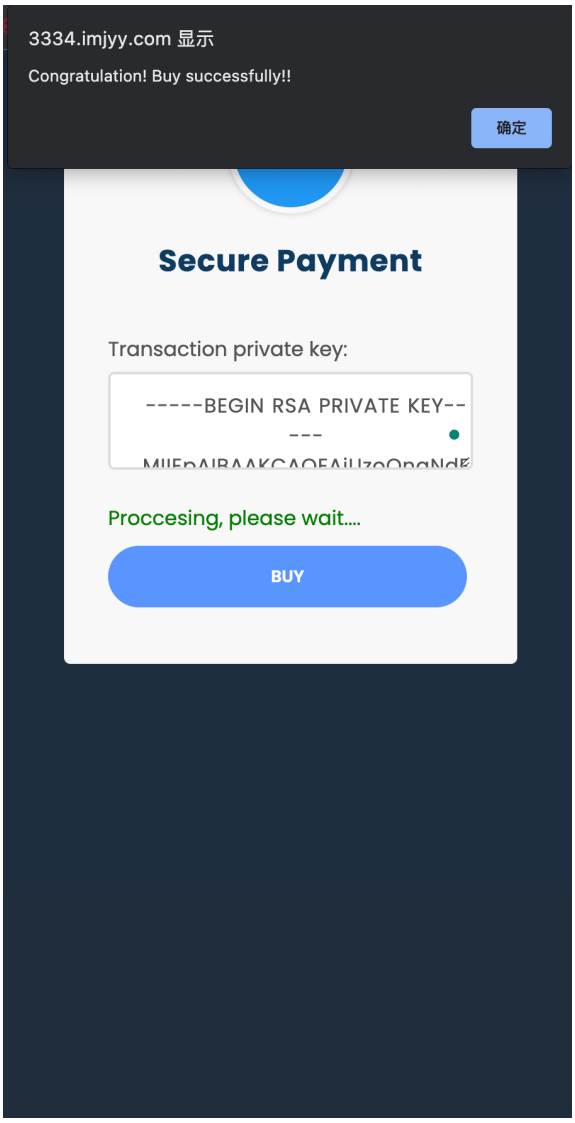

Figure 12: Trading Process

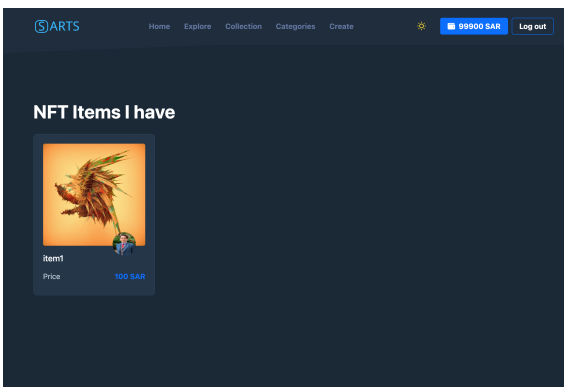

Figure 13: Owned NFT product interface

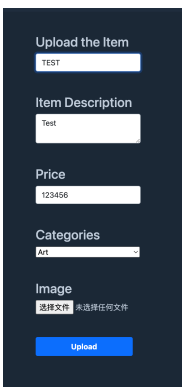

Figure 14: Upload product interface

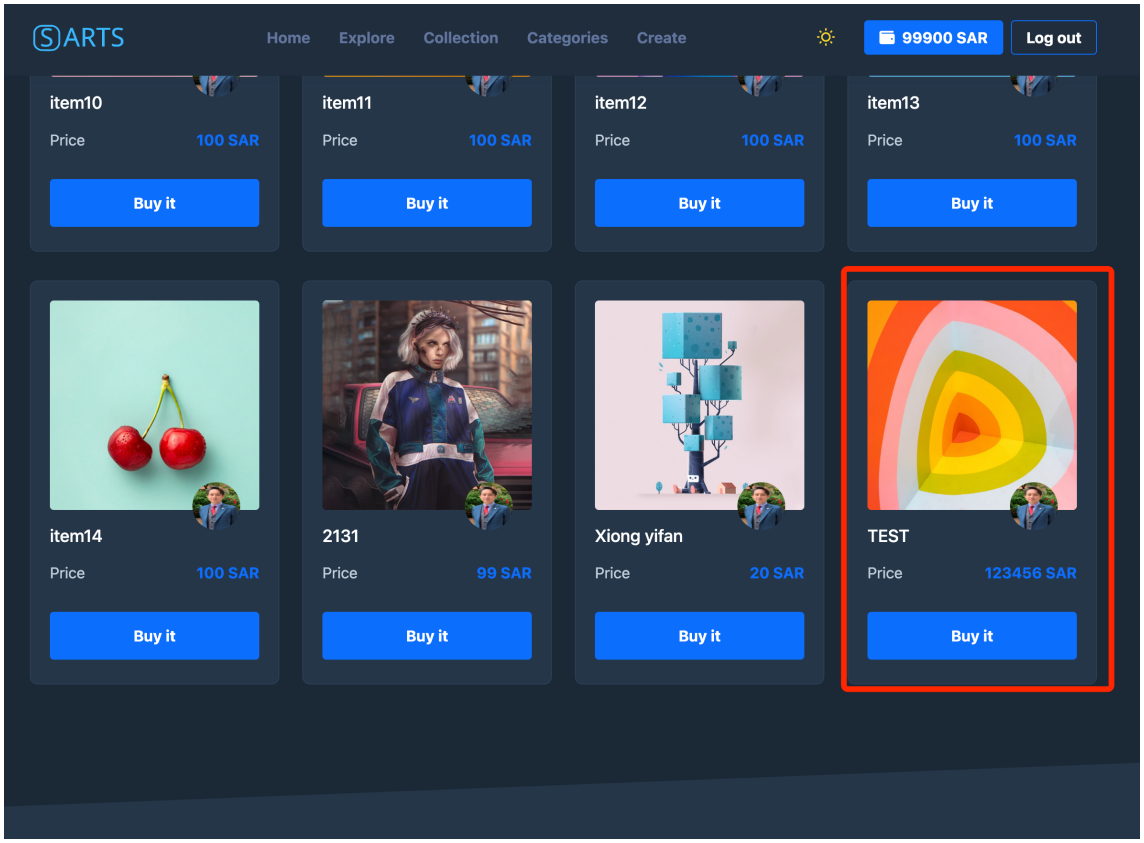

Figure 15: Update the interface of the product

References

[1] Kalyani Ganesh Kadam and Vaishali Khairnar. Hybrid rsa-aes encryption for web services. *International Journal of Technical Research and Applications", Special*, (31):51–56, 2015.

[2] Obrina Candra Briliyant and Ahmad Baihaqi. Implementation of rsa 2048-bit and aes 128-bit for secure e-learning web-based application. In *2017 11th International Conference on Telecommunication Systems Services and Applications (TSSA)*, pages 1–5. IEEE, 2017.

[3] Ratnadewi Ratnadewi, R.P. Adhie, Y. Hutama, J. Christian, and Dini Wijaya. Implementation and performance analysis of aes-128 cryptography method in an nfc-based communication system. *World Transactions on Engineering and Technology Education*, 15:178–183, 01 2017.