COMP3334 Group Project Report

JIANG Ruixiang, WANG Meng, XING Shiji, ZHANG Yubo

April 26, 2022

Contents

Abstract

Advances in block-chain technology has not only elevated the price of GPUs, but also energized the creative community. First seen in 2014 (1), the Non-fungible Token (NFT) bring to life the uniqueness of masterpiece created by artists. With the aids of block-chain technology, it is finally possible to create digital artworks that is distinguishable from the others.

The ideas adopted in NFT has greatly pushed the progression of art digitization, while it does not necessarily adopt block-chain based solution. Our system, DAP[1], is a platform representing representing, storing and exchanging digital artwork. Besides, it is also designed to protect and exchange ownership of digital artwork. Note that, the platform aims to protect the ownership of the artwork, instead of the artwork itself. The art work is very difficult to protect since it is in digital format, and you can simply take an screenshot and it's gone. In this report, we aim to analyze general-sense digital art platform in terms of security. We will discuss the security requirements of the platform, and we will provide case studies to similar platforms on the market. In second half of this report, we will specify the design considerations of the platform, and we will also provide a detailed deploy guide in the last section.

1 Security Requirement Analysis with Justifications

We analyze the security requirements for a digital art platform in this section. C.I.A (Confidentiality, Integrity, Availability) model will be applied with specific justifications, along with several suggestions of security mechanism to be adopted in practice.

1.1 Confidentiality

Before diving deep into discussions about the details of security mechanism on confidentiality, it is necessary to check what should be kept secure in an artwork platform. For an artwork platform that aims to provide functions including representing, storing and exchanging digital artwork, it is important to keep the following information secure:

1. The artwork's metadata (esp. ownership), the artwork's digital signature (if any).
2. The transaction record.
3. User's account information.
4. Communication between the platform and the user.

[1]Our code are publicly available in: `https://github.com/Yb-Z/COMP3334-Group-Project`

It should be noted that the content of the artwork itself is not considered as confidential information. This is because it would be impossible to keep the content of the artwork confidential, if the platform is designed to be used by the public. Such principle is seen in NFT-based digital assets (2), where the artwork content is stored off-chain. However, for non-NFT platforms, this idea is also applicable.

What really matters in digital art system is to keep the metadata secure. The metadata includes, but is not limited to: 1: the artwork's ownership, 2: the pointer to the artwork's content. Amongst these, the ownership information is the most important one. The system should provide a mechanism to uniquely identify the ownership of an art-work, such that every user could easily check the ownership of that art work. The ownership field should also be editable, such that user could exchange artworks. Such ownership identification functions are pivotal in digital art platform and must be kept secure in a specific way. For block-chain based application, it would not be a big issue. But for system that adopt a centralized server, they must come up with a way to securely store the ownership information. In practice, it could be achieved by using a digital signature. The owner could use a private key to sign the metadata, and the platform could use a public key to verify the signature. The digital signature is a mechanism to ensure that the metadata is not tampered with.

Beside ownership information, the pointer to the artwork also need additional security protection. In practice, the pointer could be a link to the artwork, which does not necessarily to be kept secrete. However, they must not be modified by third-parties. Details would be covered in the next section on integrity.

1.2 Integrity

Integrity in the context of digital art platform could be defined as the following:

1. The artwork's content is not modified, either partially or re-directed to another one.
2. The ownership information is not modified by third-parties.
3. The communication between the platform and the user is not modified.
4. The communication between the platform and other service server is not modified.
5. The communication is from the original source.
6. The data stored in the platform is not modified by third-parties.

For system that store artworks locally, they ought to both prevent from third-party modification and perform integrity checking. The first one usually means the server should not be accessible by any third-parties. By contrast, when the artwork content is stored in the cloud (include block-chain based), it would be impractical to only rely on the security of the cloud service provider. In such case, a better solution is to assume that the artwork content could be modified, and provide a way to check the integrity of the artwork. In practice, the platform could pre-compute the hash of the artwork content, and store it in the metadata local server. The platform could then check the hash of the artwork content when it is requested.

For the metadata, it must be stored on a trust-worthy server (which again depends on the security assumption and architecture). The reason is that they carry sensitive information such as ownership. Even if the server is trustable, they still should be encrypted using either symmetric or asymmetric encryption. The ownership should not be modified by third-parties, but it should be made public, so that the user could easily check the ownership, which suggests that asymmetric encryption is a good solution. As for the pointer to the artwork content, it should not be modified by third-parties, and ideally not accessible by the public. For other data, the encryption mechanism should be chosen based on the security assumption and the size of the metadata. In general, an AES encryption is sufficient for the metadata. Although encryption does not prevent modification, it does provide a way to check if the data has been modified or not.

Apart from integrity of the content, the integrity in communication between each part of the system is also important. For example, the content of communication between the platform and the user should be kept original. This both means the data send to and from the server. For web-based system, there exist various solutions to achieve this security requirements. For example, a lot of Web frameworks support authentication mechanism, and integrity of communication content could be achieved by SSL and TLS. For other means of communication, there are also established standards/protocols to achieve this security requirement. For e-mail, the integrity checking and encryption of could be done by S/MIME (3) or PGP(4).

When the data is no-longer trust-worthy, the system should take actions. For example, the system could simply rollback the whole transaction to avoid any inconsistencies. However, such naive approach may impose a limit on performance of the system if there are frequent inconsistency of data, which is common in transaction-based system. Better practices are possible, for example to temporarily accept questionable data to form different branches of the system, and choice the one that are most trust-worthy (e.g., *blockchain*).

1.3 Availability

Availability of an artwork platform generally impose the following overlapping security requirements:

1. The artwork's content is available to the user.
2. The communication between the platform and the user, if any, is available to the user.
3. The communication between the platform and other service server , if any, is available to the user.

The availability of the artwork content means that user can access the artwork content without any problem. We only discuss the situation where the content is stored in the cloud. In such case, the availability requirement could be decomposed into two parts: 1: the pointer to the content is correct, and 2: the transfer of the content is available. The first part is already covered in the previous section on integrity. The second part, however, is more complicated as it involves external server. To avoid the DoS attack, the system should be designed to limit the number of connections to the external server. For example, the system could use a proxy server to limit the number of connections to the external server. This also means the system should detect abnormal connections. However, it would be difficult to differentiate the normal connections from the abnormal connections, and the corresponding actions further depends on the architecture of the system. Apart from detection, the system may also consider encrypt the original pointer to the content, and only expose a 2nd-level pointer to the user.

2 A literature review of similar systems in the market

In this section, we review the similar system in the market. We will first classify the similar system into three categories based on the server architecture, and then we will give a case study to a specific similar system.

2.1 Overview of the similar system in the market

An digital artwork platform in a larger sense is a Web application. Therefore, there mainly exists three possible server architectures: point-to-point (P2P), Centralized and Distributed.

1. P2P: point-to-point architecture generally means clients could directly communicate with each other. In such systems, each client is also a server, and each server is also a client. The

strength of P2P is that there is no centralized server, reducing the likelihood of single-point information leak. Also, since the data is not processed by an intermediate server, it could be easier to implement the security mechanism. However, it would be a challenge to view others' artwork without a server. To the best of our knowledge, there is no such digital art system in the market.

2. Centralized: centralized architecture generally means that there is a centralized server (logically), which is responsible for processing data and communicate with all clients. This is the most common architecture for Web applications. The benefits of centralized architecture are that it is easy to implement and maintain the system. However, it also means that the system is more vulnerable to attack and single-point failure. Moreover, it also means the system admin has full control of the system. Examples of such digital art systems include DeviantArt (5), Pixiv (6).

3. Distributed: distributed architecture generally means that there are multiple servers, and there is usually redundancy between the servers. Note that a centralized server could have distributed server physically, and we do not consider this case. While there are a lot of ways to achieve de-centralization, we focus on the blockchain-based (i.e., NFT). Basically, NFTs store the metadata on the chain and store the art content off-the chain. The advantage is that blockchain is a fully decentralized system, which intrinsically provides a secure and reliable system against third-party modification. The disadvantage include the off-chain storage, which suffers from the cost of storage and the difficulty of finding the data. In addition, the blockchain itself is not a scalable system, which means that the system is not able to handle the large amount of data. Examples of such digital art systems include Quantum (1), Crypto(6).

2.2 Case study of a similar system

Although it is not compulsory to use NFT / Blockchain as the means for building up our system, NFT marketplace is a kind of system that is most similar to our system, so we look closely into one of the largest marketplaces in the world: OpenSea[2].

OpenSea is the world's first and largest NFT marketplace. Besides the functionalities of a online shop, the website also provides mechanisms to store and protect the artworks uploaded, which is related to the security. The underlying security mechanisms (or contracts) of NFTs is implemented with OpenZippelin, which provides security products to build, automate, and operate

[2] `https://opensea.io/`

decentralized applications, and is he standard for secure blockchain applications. OpenZippelin is written in solidity, which is a library in Node JS.

The basic of the contracts is the block chain. By its nature, since all transcations(editions) will be recorded, block chain can prevent secret tampering. Therefore, the important factor is to build a contract that guarantee that only legit editions can be made to the block chain, which is authentication. OpenZeppelin provides a comprehensive standard template of contract, namely ERC721 for users to implement. By properly implementing this template, user can fulfill basic functionalities security requirements of an NFT.

Besides, there are few points stressed when implementing an contract:

- All information is public in a contract, even when marked private.
- Any interaction from a contract (A) with another contract (B) and any transfer of Ether hands over control to that contract (B).
- The call stack size has a limit of 1024. Recursive methods should be called with caution.

3 System Design Specification

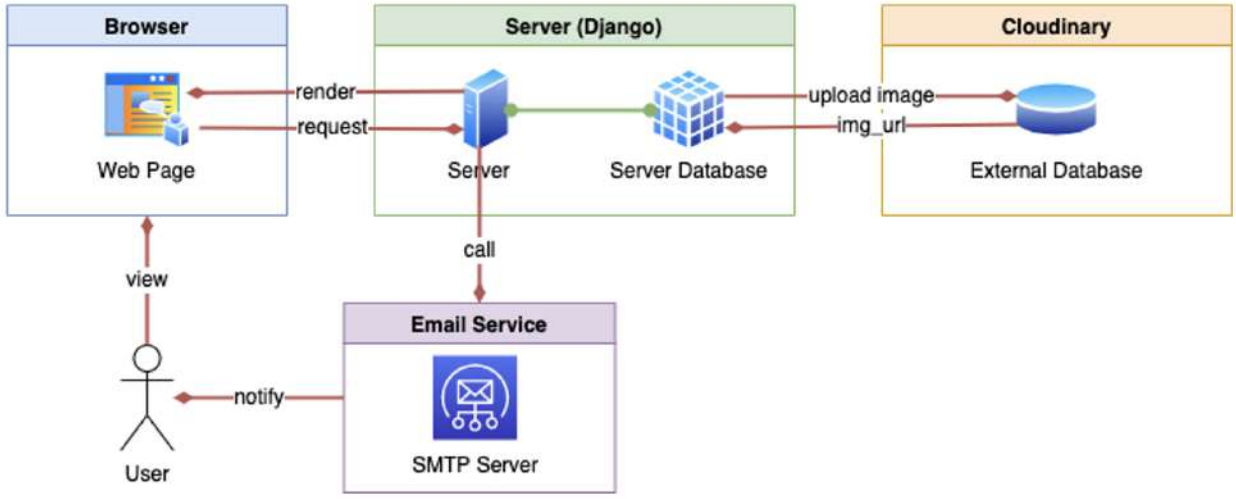

Figure 1: Architecture Overview of DAP

The system is designed with specific security consideration which will be introduced in this section. Overall, DAP is a client-server system and there is no direct communication between clients. The server will get involved in all activities. Therefore, we do not need to consider about P2P protocol and security. As a client-server system, DAP has a back-end and a front-end.

Front-end code provides interaction entries and get/post data to the server, whereas back-end stores information of artwork and handles encryption/decryption. The architecture is depicted in Fig. 1. We implement DAP based on Python Django Web framework[3].

3.1 Security assumptions

Similar as any system, our discussion on security of DAP is base on several assumptions:

1. The front-end of the system could be accessed by the public.
2. The private key and user data in server cannot be hacked
3. Third party services like cloudinary is trustworthy
4. The communication between the platform and the user could be insecure.

3.2 Authentication and authorization

As a online platform, it is important to have an authentication mechanism. We use the authentication system provided by Django. Django authentication system handles both authentication and authorization. In the context of Django, authentication verifies that a user is who they claim they are, whereas authorization determines what permissions does an authenticated person have.

A model in Django is the single, definitive source of information about data. Each model maps to a single database table. To be more specific,

- Each model is a Python class that subclasses django.db.models.Model.
- Each attribute of the model represents a database field.

User models are the core of the authentication system. Clients and administrators are the two types of users that interact with DAP. For administrators, they are allowed to access the admin page and view/modifiy our database. DAP does not store raw (plain text) passwords on the user model, but only a hash. Therefore direct manipulation of the user's password properties is not allowed, even for administrators. By default, Django uses the PBKDF2 (7) algorithm with a SHA256 hash to store passwords, which is a password stretching mechanism from RSA Laboratories' Public-Key Cryptography Standards series recommended by NIST. This should be sufficient for DAP: it's quite secure, requiring massive amounts of computing time to break. Keys of PBKDF2 are encrypted based on RSA and stored in Django admin database.

[3]https://www.djangoproject.com/

```
To: polyu@fake.email
Date: Sun, 17 Apr 2022 13:52:01 -0000
Message-ID:
 <165020352157.4080.17795930481174777233@1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.
.0.0.0.0.0.0.0.0.0.0.0.0.0.ip6.arpa>

-----BEGIN PGP MESSAGE-----

yMJZATvCy8zAwci1in+b57VUUYbTB/hLGZJipBtdUhOLFArycypLdbi4IvNLixQc
i0rK84uyFUoyUnNTrRQySkoKiq309YtSi/WSc/JLUzLzEosq9ZLzc/Vz81PzSgsS
9TNzE9NT9UsLcvITU/TLDM1MDQwtzYwtjfUrKxIzSyoTs3Ly0/JTc3JKSgtyU/Wy
CtIVMosVcjKzU1MUkioVkkorU4sUElNyM/OsFKBGOuSX1uTk52eDrNEDOUshOTFP
ITkjNTlbIbNEASirkFiikJJYAFHA5eIYcOhgMQsDIweDLCsTyF9iiiz+n/w1jl1c
```

Figure 2: An example of PGP encrypted email (partly)

3.3 Communication

We use HTTPS for communication. The main underlying protocol under HTTPS is TLS, which implements an mechanism similar to strong password protocol to ensure security. The difference is that, TLS requires the server to have an certification, which will help prove the identity of server. As a consequence, only server needs to have a private key in TLS. Without the use of HTTPS/SSL, malicious web users could sniff authentication credentials or any other information transmitted between client and server, and in some cases, active web attackers could change data sent in either direction.

Besides HTTPS, since DAP use email to deliver messages, we also need to concern the security of email as a data transfer method.

By default, SMTP does not include encryption of data. However, the email data should be protected in our application. Pretty good privacy (PGP)(4) is one of the possible solution to add security level. Basically, PGP use a combination of cryptographic functions to sign and encrypt the text. We use PGP to protect the email content, so that the content could only be decrypted by the receiver. Example see Fig 2.

Apart from PGP still provide some encryption level to the email. When we are communicating with the SMTP server, we use TLS (Transport Layer Security) provided by Django[4] to encrypt the data.

[4]https://docs.djangoproject.com/en/4.0/ref/settings/email-use-tls

3.4 User data

When user uploads a artwork, we calculate and store locally the hash of the file. Such hash is used to check the integrity of the file when we further fetch it from cloudinary. This design follows our discussion about integrity in section 1.2.

For the content of artwork, as mentioned in Sec 1.1, does not necessarily to be encrypted. We store artworks in cloudinary, which additionally provides some safety mechanism (they do not release the detail).

3.5 Digital artwork

As a digital artwork platform, digital artwork themselves are stored in a third-party platform called cloudinary[5], who has their own mechanisms to ensure the safe storage. After a user post his/her digital artwork *DA1*, DAP will call cloudinary api to transfer *DA1* securely (via HTTPS). Then, cloudinary will store *DA1* together with a digital signature in their database. Finally, cloudinary will reply a secure link (also HTTPS) of *DA1* to DAP and DAP will store this link as part of user data. Users with successful authentication can view *DA1* via this link.

Due to business and security concerns, cloudinary does not disclose too much specific security mechanisms to developers, but using the services they provide is still one of our safest choices, which also matches the assumption at Sec 3.1.

The ownership information (OI) is important to generate digital signature (DS) on digital artwork. OI is created/modified when the owner post their digital or a transaction is done, and DS will be changed in cloudinary storage simultaneously. Along with other user data stored in user models, OI is also protected in our database.

3.6 Transaction

Transaction in a higher level of abstraction means the change of ownership. It does not necessarily invloves any format of currency. In our DAP, we currently do not include real payment APIs for some commercial issues. In our current setting, users has rights to transfer the ownership of an artwork to another user. Users could also request transfer of artwork, and if the owner agree, the transaction could be done.

[5]https://cloudinary.com/

4 System installation guide

The requirements of the project on its hosting platform is shown in Table 1. Make sure the requirements are met before proceeding to deploy the project.

Level	OS	Software
Minimum	Windows 10; MacOS 10.9; Linux 4.0(kernel)	Python 3.8
Recommended	Windows 11; MacOS 13; Linux 5.17(kernel)	Python 3.9, 3.10

Table 1: Requirements

1. Make sure having python libarary `virtualenv` installed, or install with `pip install virtualenv`.

2. Initialize a new virtualenv and install all dependencies.

```
python -m venv env                          # create virtual environment
source env/bin/activate                     # activate virtual environment.
# On Windows, run env\Scripts\activate instead.
pip install -r requirements.txt             # install all dependencies
```

3. Initialize the Django database schema.

```
python manage.py makemigrations marketplace
python manage.py migrate
```

4. Creating an admin user
   First we'll need to create a user who can login to the admin site. Run the following command:

```
python manage.py createsuperuser
# Enter your desired username and press enter.
Username: admin
# You will then be prompted for your desired email address:
Email address: admin@example.com
# The final step is to enter your password.
# You will be asked to enter your password twice,
# the second time as a confirmation of the first.
Password: **********
Password (again): *********
Superuser created successfully.
```

After step 8, open a Web browser and go to "/admin/" on your local domain – e.g., https://127.0.0.1:8000/admin/. You should see the admin's login screen.

5. Save environmental variables into a `.env` file:

```
# Cloudinary api configurations (available after registration),
# see https://cloudinary.com/
C_NAME = ""
C_KEY = ""
C_SECRET = ""

# email backend configurations
# send to console: "django.core.mail.backends.console.EmailBackend"
# send to mailbox: "django.core.mail.backends.smtp.EmailBackend"
EMAIL_BACKEND = "django.core.mail.backends.console.EmailBackend"

# A mail account as host for sending emails
EMAIL_HOST = ""
EMAIL_HOST_USER = ""
EMAIL_HOST_PASSWORD = ""

# Django Secret key
SECRET_KEY = "<THIS KEY MUST NOT BE EMPTY>"
```

6. Make locally-trusted development certificates for https.
   This tool, mkcert, is available to generate the certificate and the key.

7. Run the server on localhost.

```
python manage.py runserver                # run the server in http
python manage.py runsslserver --certificate cert.pem --key key.pem
                                          # https with local certificate
```

By opening the URL shown in the console after running the server (maybe http://127.0.0.1:8000 or https://127.0.0.1:8000) with browser, a welcome page (e.g., Figure 3) should show up, which indicates completion of deployment.

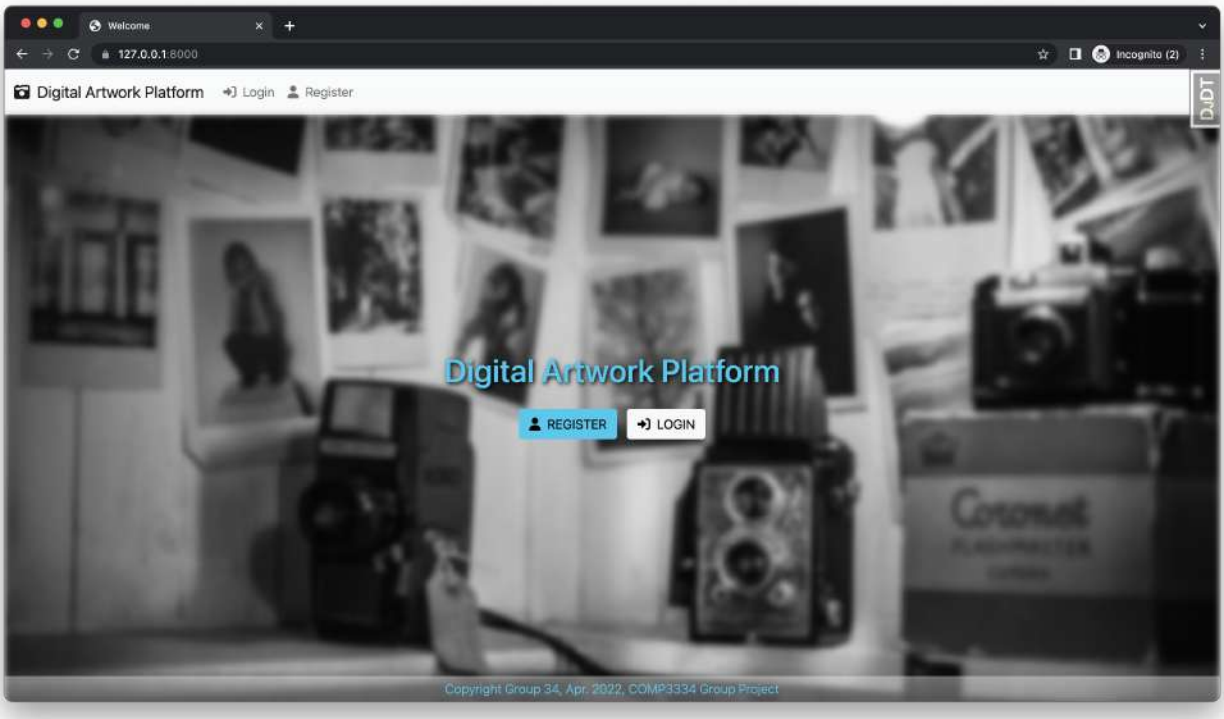

Figure 3: Welcome Page, which indicates success of deployment

5 Use cases

5.1 Use Case 1

A user called testuser1 first sign up DAP market.

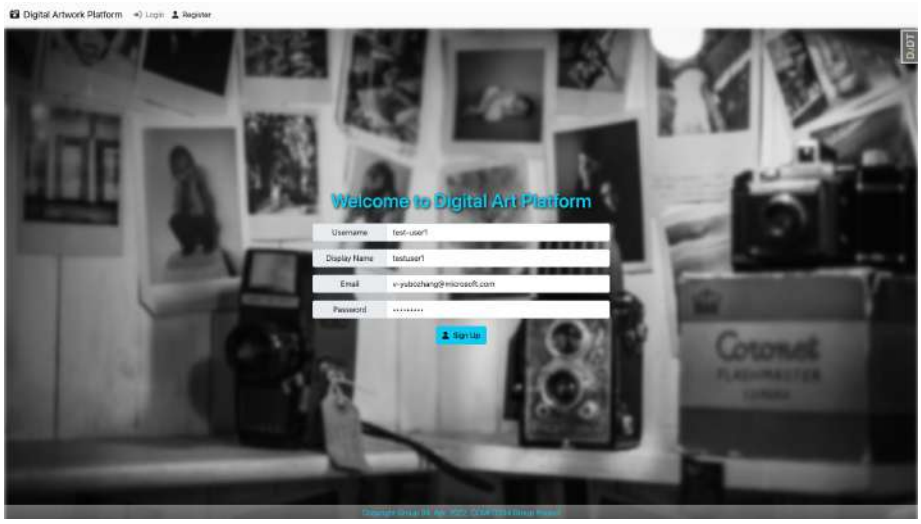

Figure 4: Case1 – Sign up

After sign in, testuser1 can post his/her new digital artwork.

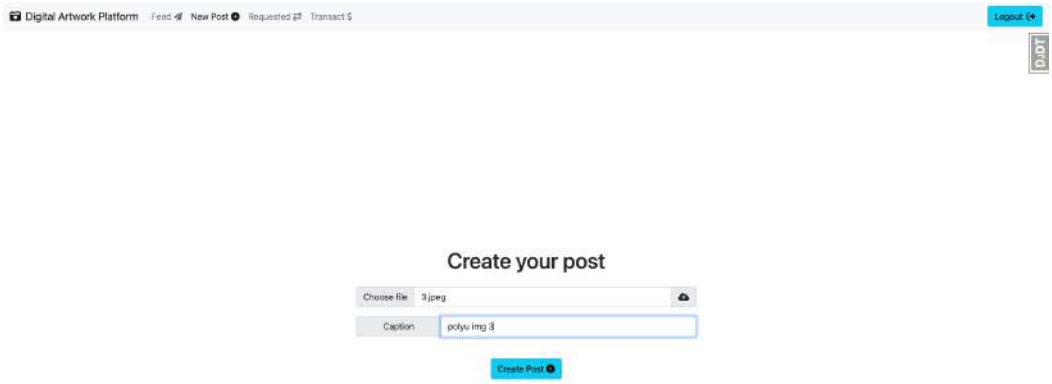

Figure 5: Case1 – Create your post.

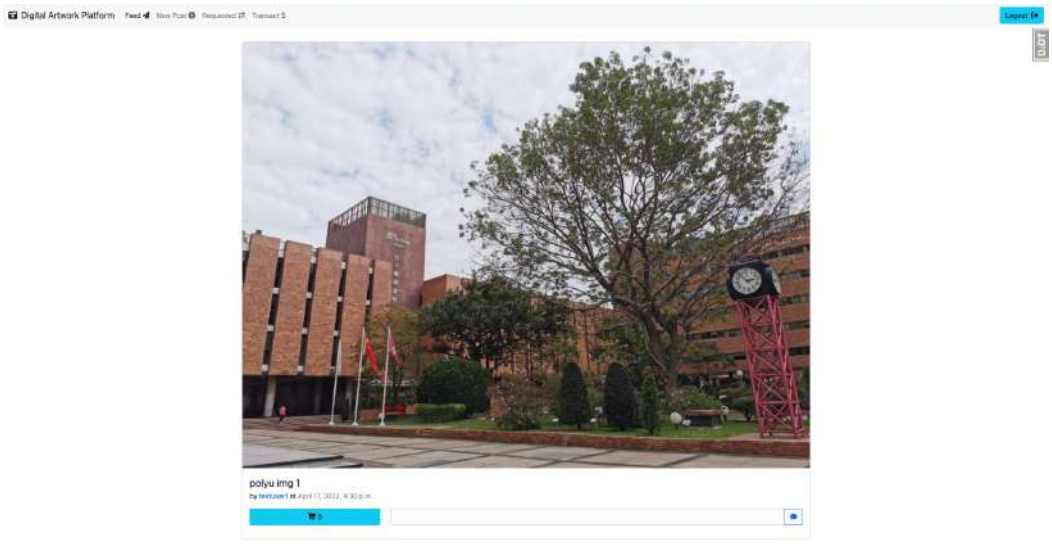

Figure 6: Case1 – View your created post.

Similarly, another user called testuser2 can also sign up and then sign in to new a post. They can view other digital artworks at the "feed" page.

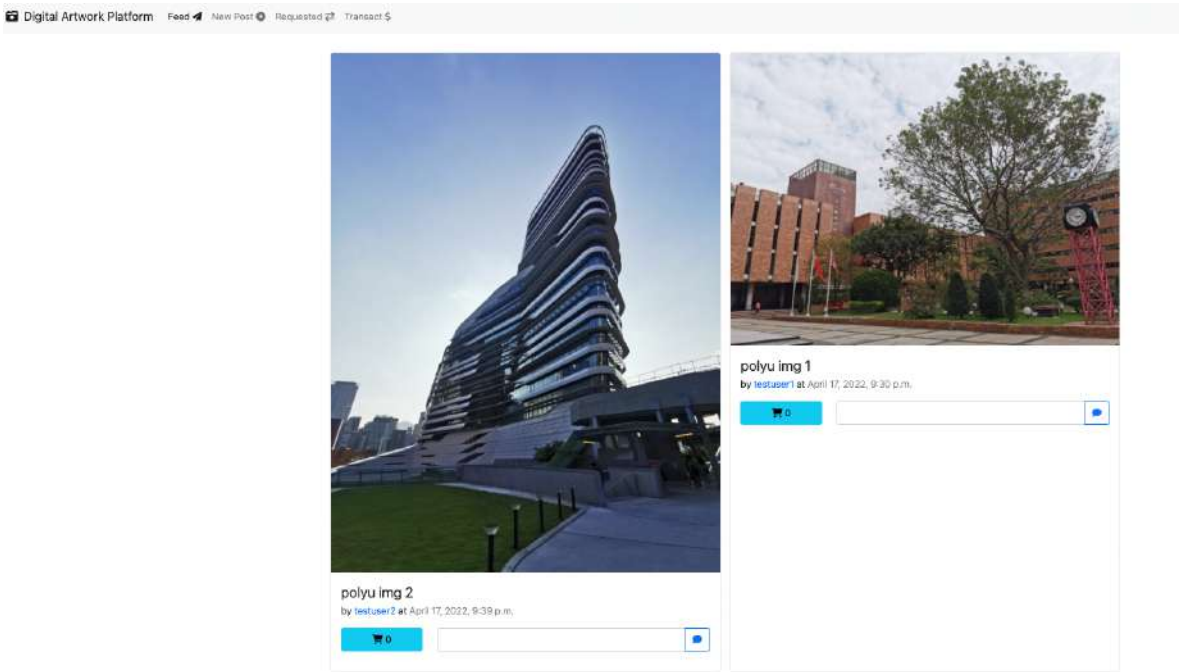

Figure 7: Case1 – Feed page with two digital artworks "polyu img 1" and "polyu img 2" created by testuser1 and testuser2 respectively.

If testuser2 is intersted in testuser1's digital artwork "polyu img 1", he/she can press the "shopping cart" button and then a request will send to testuser1 via a encrypted system email.

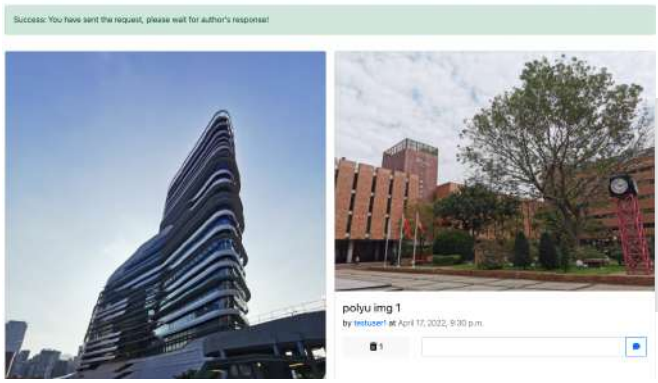

Figure 8: Case1 – testuser2 send a request to testuser1's digital artwork "polyu img 1"

Figure 9: Case1 – the email sent to testuser1

After testuser1 approved the transaction, testuser2 can checkout.

Figure 10: Case1 – testuser1 approves the transaction

Figure 11: Case1 – testuser1 checkout the transaction

Finally, the ownership of digital artwork "polyu img 1" is successfully changed to testuser2.

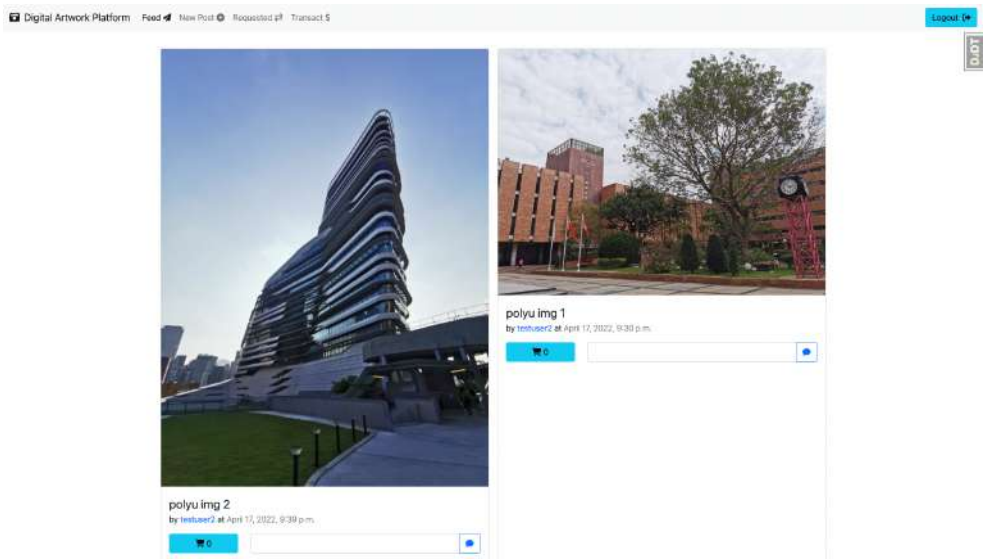

Figure 12: Case1 – Feed page with two digital artworks "polyu img 1" and "polyu img 2" created by testuser1 and testuser2 respectively.

5.2 Use Case 2

In this section, we will present some extra functions of DAP.

testuser1 posts a digital artwork called "polyu img 2". If testuser2 tries to upload a similar "copied" figure without testuser1's signature, DAP will warn him/her that this digital artwork already exists.

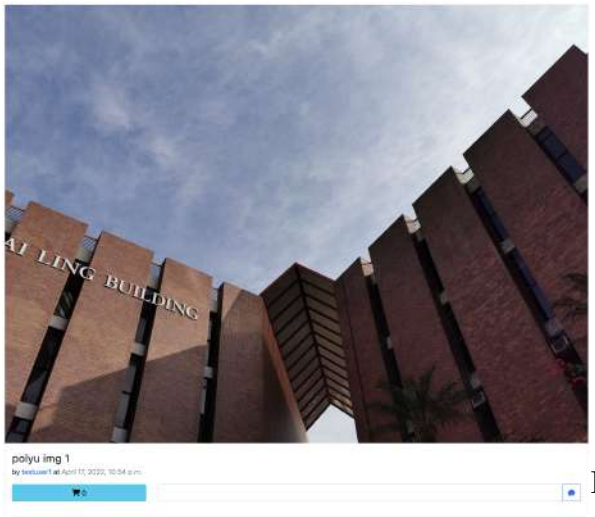

Figure 13: Case2 – testuser1 post a digital artwork

Figure 14: Case2 – a warnning to testuser2

Besides sending "request" to the owner, testuser1 and testuser2 can also comment to show

their interest. These comments are private and only visible to them.

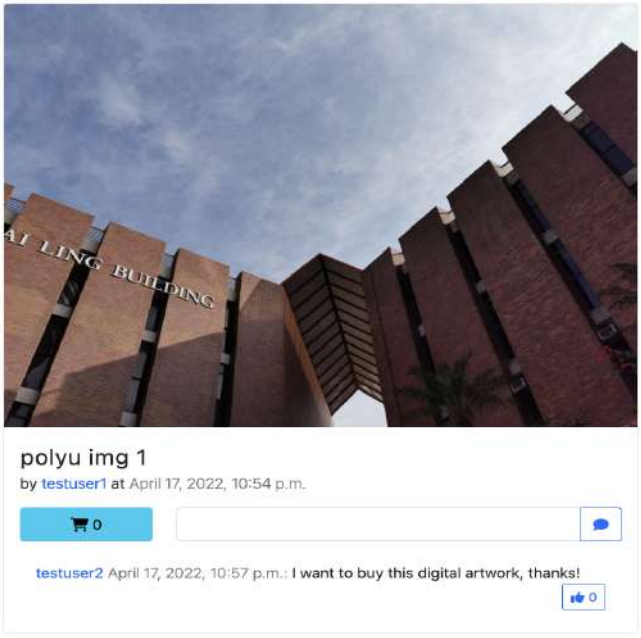

Figure 15: Case2 – testuser2 comment on "polyu img 1"

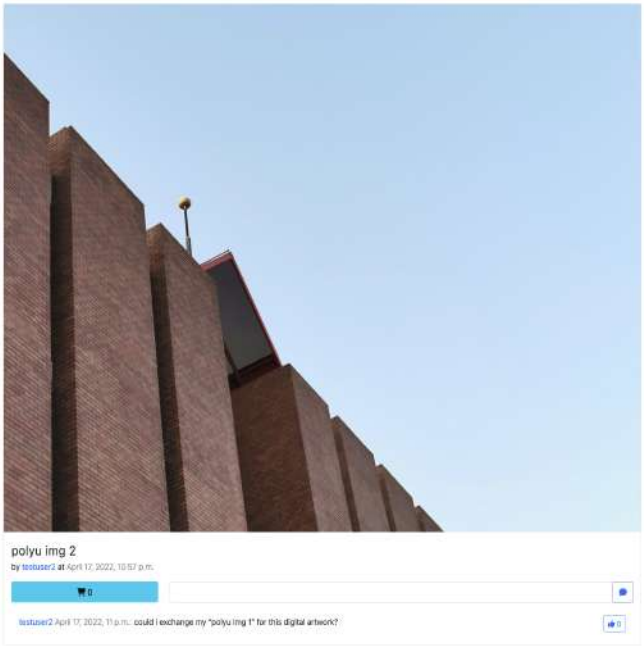

Figure 16: Case2 – testuser1 reply via comment to testuser2

If they achieve agreement, they can complete the transaction procedure the same as case 1. After they both complete the transaction procedure, system check the transaction validity, then rewrite the ownership of their exchanged digital artworks.

References

[1] "Quantum.art https://quantum.art/," Apr 2022.

[2] D. Daniele, "Nfts' nifty copyright issues - intellectual property - canada," May 2021.

[3] J. Peterson *et al.*, "S/mime advanced encryption standard (aes) requirement for the session initiation protocol (sip)," tech. rep., RFC 3853, July, 2004.

[4] N. W. Group *et al.*, "Openpgp message format," *RFC 4880*, 2007.

[5] "Deviantart - the largest online art gallery and community."

[6] "The best place to buy, sell, and pay with cryptocurrency."

[7] B. Kaliski, "Pkcs# 5: Password-based cryptography specification version 2.0," 2000.