

Decentralized Trading Platform – Final Project Document

COS30049 – Computing Technology Innovation Project

Group 1.5

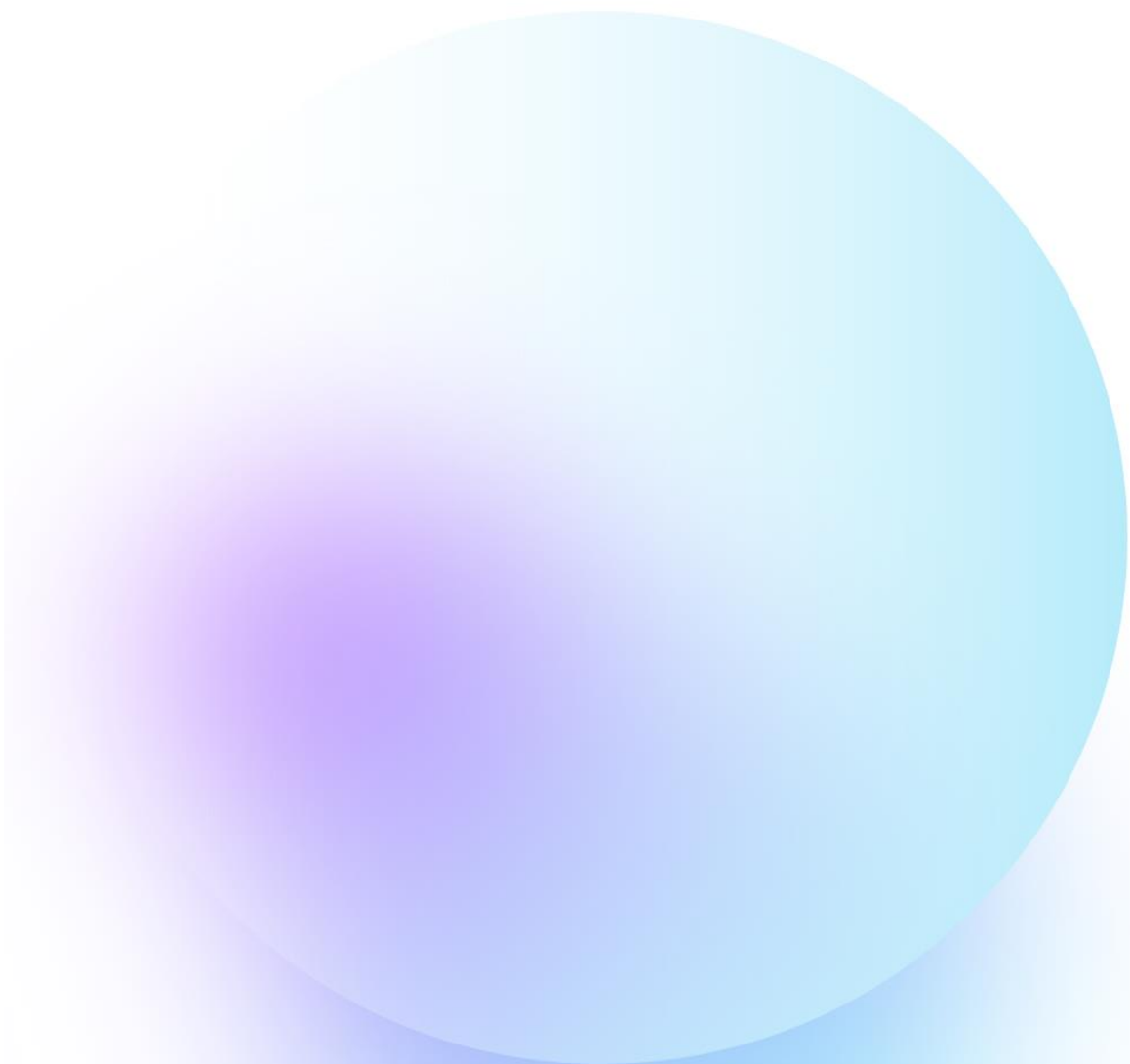


Table of Contents

| | |
|---|----|
| Project background and introduction | 2 |
| Team introduction | 3 |
| Project requirement list and description | 4 |
| Project design | 5 |
| A. Overall system architecture design | 5 |
| B. Front-end prototype | 6 |
| C. Backend database design | 10 |
| D. API design | 13 |
| POST /api/signup | 14 |
| POST /api/login | 15 |
| POST /api/logout | 16 |
| GET /api/login | 17 |
| GET /api/profile | 18 |
| PATCH /api/profile | 19 |
| GET /api/collected | 21 |
| GET /api/collected/:tokenId | 22 |
| GET /api/traded | 23 |
| GET /api/transactions | 24 |
| POST /api/sendCoins | 25 |
| POST /api/buyCoins | 26 |
| POST /api/mint | 27 |
| GET /api/marketplaceCount | 29 |
| GET /api/marketplace | 30 |
| GET /api/marketplace/:postingId | 32 |
| POST /api/marketplace | 34 |
| POST /api/purchase | 35 |
| E. Function description | 36 |
| User authentication | 36 |
| Profile viewing and management | 37 |
| Sending and purchasing coins | 39 |
| Minting and uploading NFTs | 40 |
| Browsing the marketplace and purchasing items | 41 |
| F. Project deployment instructions | 43 |
| Conclusion | 45 |
| References | 45 |

Project background and introduction

Blockchain technology has ushered in a new era of digital ownership and decentralized ecosystems. Non-fungible tokens (NFTs), a type of cryptographic asset, have become very popular in recent years. These tokens reflect ownership of digital or physical assets, offering a safe and transparent method of determining validity and provenance. The increased interest in NFTs has resulted in the establishment of several platforms and markets, each catering to a unique set of makers and collectors.

This project aims to enrich the burgeoning NFT ecosystem by creating **a cutting-edge virtual fashion trading website**. Our platform guarantees users a smooth and visually appealing experience by using the capabilities of React, a prominent JavaScript toolkit for creating user interfaces, and Tailwind CSS, a utility-first CSS framework, for the frontend. The backend is built with Express.js, MySQL, Web3.js, and Geth, industry-standard tools that ensure a secure and efficient business logic layer. The combination of these technologies enables us to develop a responsive and dynamic web application that meets current industry requirements.

Team introduction

Our team is called Maverick Mates (a legacy of teamwork efforts in previous units) and consists of Ta Quang Tung, Nguyen Quang Huy, Tran Hoang Hai Anh, Phan Sy Tuan, Vu Xuan Sang, and Nguyen My Hanh. The team has been working together to create a decentralized trading platform with a blockchain system. Below we describe in more detail the contributions made by our team members:

Tung, the team's leader, is the sole web designer and lead programmer of the project. The reason he takes on the designing task alone is to ensure stylistic consistency throughout the website. This does not mean other members have no say in the process; their feedback is also considered and incorporated into the design. Using Figma as the design tool, Tung has created a UX-focused and futuristic-looking website for trading virtual fashion NFTs. Tung was also responsible for choosing the tech stack and the front-end architecture, programming some pages, and reviewing the code written by the other members. He is also responsible for the development of the smart contract, which is one of the cores of the application. Tung oversees the development of the backend and makes sure every component works smoothly together.

Huy is responsible for coding perhaps the most important page of the website - the Marketplace page. This page is the highlight of our decentralized platform and acts as the entry point to trading where users can search for assets of interest. Huy is responsible for the logic of the marketplace page, which enables users to search for, filter, and sort NFTs on sale.

Hai Anh programs the Item information page, which displays detailed information about a specific asset. This page is where the user makes one of the most important interactions on the platform: purchasing an item. To increase the likelihood that the user decides to buy an item, the page has to be visually pleasing and present the item's information logically. Hai Anh also works on purchase logic to enable transactions to happen securely.

Hanh and Sang are responsible for programming the user profile page, based on Tung's Figma design. The profile page is meant to be a hub where the user can see their personal information and trading history on the platform. To this end, Hanh and Sang have to ensure all the expected information and user interactions are presented intuitively on the page. They contribute to the backend logic that helps users see their collection and transaction history.

Tuan handles the header and footer, seemingly simple components that play a key navigational role on the website. These components link all the pages together and allow the user to quickly navigate the website. Tuan also helps develop the logic to mint and upload NFTs.

Project requirement list and description

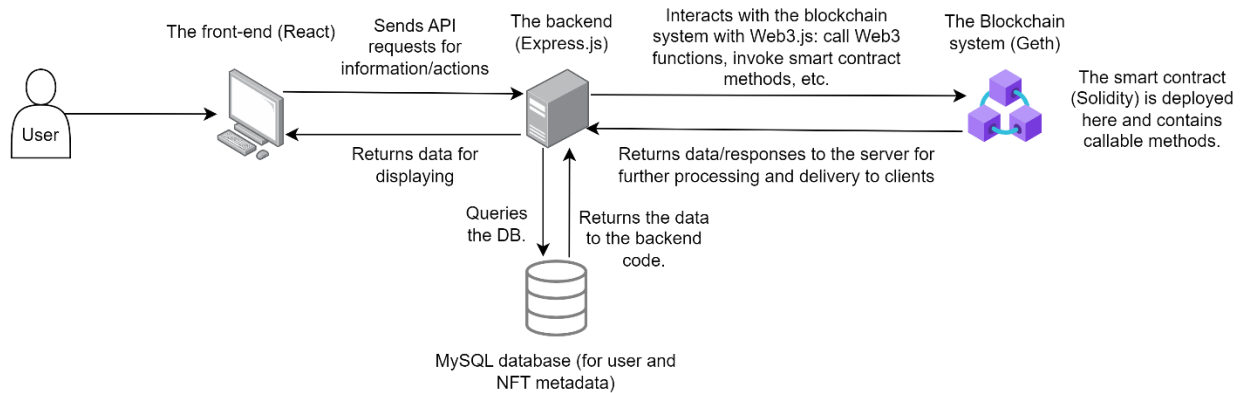
The vision of the project is to build a decentralized trading platform with a fully-featured frontend, backend, and blockchain system. Our design is based closely on the platform's requirements as specified on Canvas. Below we present our analysis of these requirements and map them to the necessary functions.

- **Users can view digital assets available for trading:** This requirement hints at a place where the user can see all the available assets on the platform. We will achieve this by designing a Marketplace page that will list the available assets. The backend logic will enable users to selectively read assets from the MySQL database.
- **All the listed digital assets information should be stored in the database:** To show this information, we have designed an Item information page, which displays all the information (price, description, owner, etc.) related to an item. This page is accessible by clicking on an item in the Marketplace. The asset information will be stored in a MySQL database.
- **The system should provide a search and filter functionality for users to discover specific assets of interest:** To fulfill this requirement, we will add a search bar with search and filter options on the Marketplace page. The backend API will receive requests with options to search, filter, sort, and return the data as appropriate.
- **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:** We will write a smart contract implementing the ERC 721 standard to enable users to mint and transfer ownership of assets securely. The smart contract methods will be written such that any errors during the transaction will revert it and maintain the state of the blockchain.
- **Users should have access to a transaction history to view their past trades:** This requirement suggests a page where users can see their trading activity on the platform. To achieve this, we will design a User Profile page that lists all the items that the user has bought or sold and the crypto transactions they have made. The transactions' metadata (including their hashes) will be stored in a MySQL database and they can be looked up from the blockchain by hash.

Project design

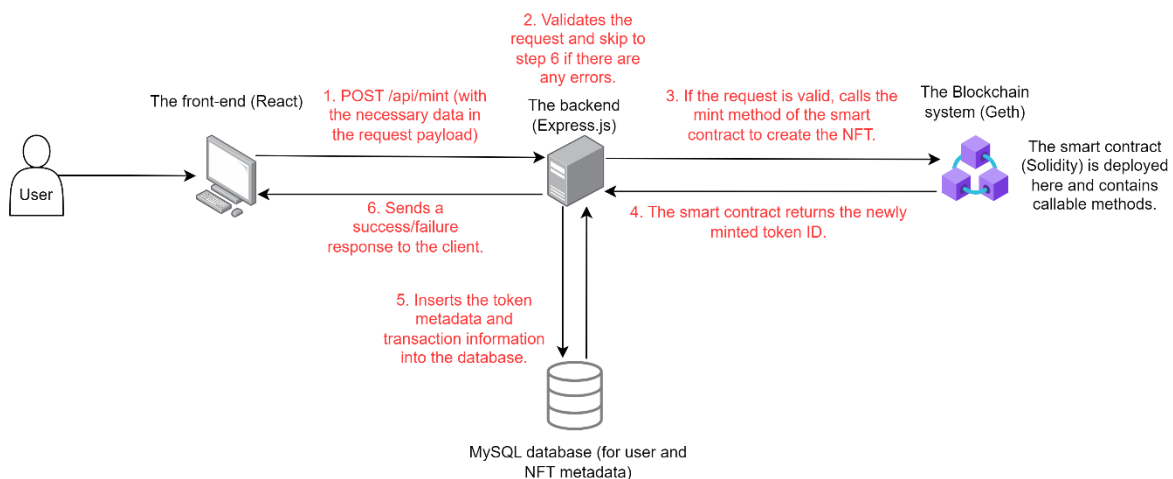
A. Overall system architecture design

The project is a full-stack web application with an integrated blockchain system. The overall system architecture is visualized in the diagram below.



According to the diagram, the frontend consists of a web interface built with React. The frontend will make API requests to the backend server, which is written in Express.js, to fetch data or perform actions such as minting or purchasing NFTs. The backend server communicates with a MySQL database that stores user, NFT, and transaction metadata and a blockchain node launched by Geth which contains the user addresses, NFTs, transactions, and smart contract code. Communication between the backend server and the Geth node is possible thanks to the Web3.js library. The smart contract is written in Solidity and then compiled and deployed to the blockchain. It implements the ERC 721 standard, which is a standard governing the ownership of NFTs (OpenZeppelin, n.d.).

To illustrate how this architecture works, we present a diagram for the use case of minting NFTs, an operation that is central to our platform.



B. Front-end prototype

The architecture of the complete project will include both the frontend and backend. The user will interact with the frontend, which in turn will send the necessary requests to the backend and blockchain system to perform transactions. Transactions will be securely stored in the blockchain system and cannot be tampered with once created.

To build the frontend website, we use the **React.js** library and the **TailwindCSS** framework. React is a popular JavaScript library for building UI components which makes the development process faster than writing traditional HTML. Tailwind is a utility-first CSS framework that is easy to use and enables our designs to be consistent.

Our frontend architecture consists of six pages:

1. The **homepage**, which is the entry point of the website.
2. The **marketplace**, which lists the available trading assets and offers search, sort, and filter functionalities.
3. The **item information** page, which displays all the information related to an item and the option to purchase it.
4. The **user profile** page, which shows the user's information, including their transaction history, wallet balance, and items they have traded.
5. The **signup** page, which enables a new user to create an account on the platform.
6. The **login** page, which enables a user with an existing account to log into the platform.

Since the website is for a virtual fashion NFT trading platform, we have decided to follow a futuristic aesthetic. For this, we have chosen modern-looking fonts and a space-esque color palette consisting of dark purples and bright pinks. We have also incorporated glass morphism design into our website, which makes certain objects appear like glass, to lend it a more futuristic feel (Design Studio, 2024). To enable a smooth user experience, all of our pages have been designed to be responsive.

Before beginning the coding process, we designed all our pages in Figma first. Our homepage was inspired by a Behance design (Jahangir Ali, 2024) and our user profile was inspired by OpenSea's user profile page (OpenSea, n.d.). Below are images of our Figma designs:

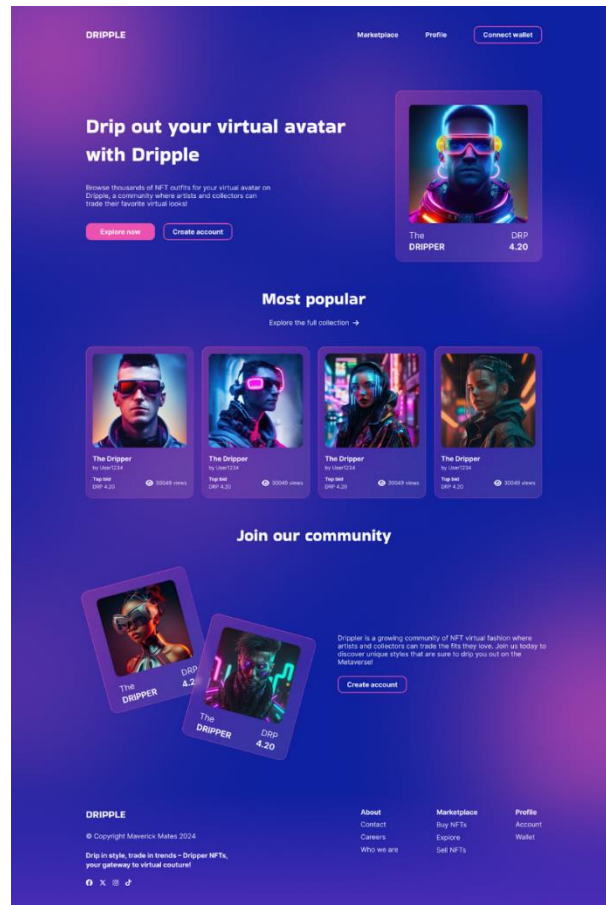


Fig. 1: The website's homepage.

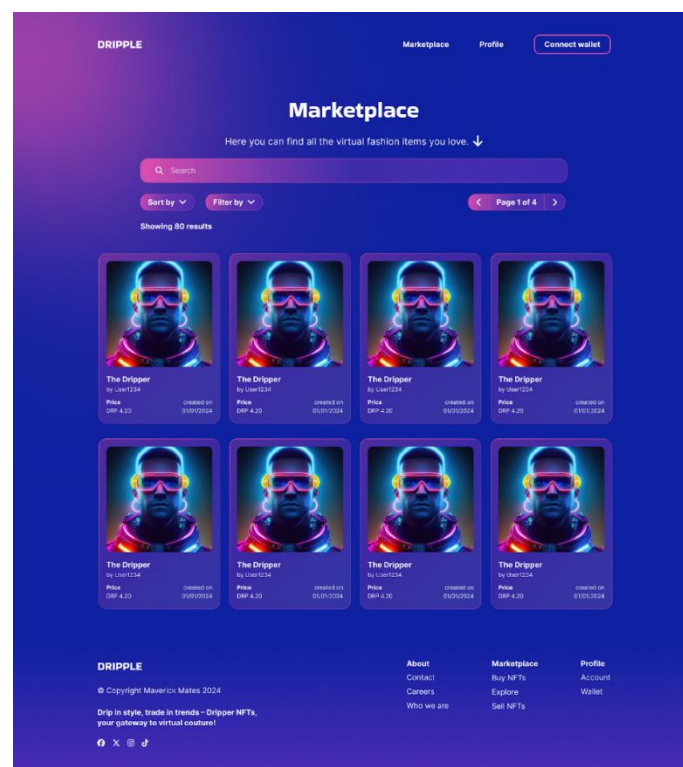


Fig. 2: The website's marketplace page.

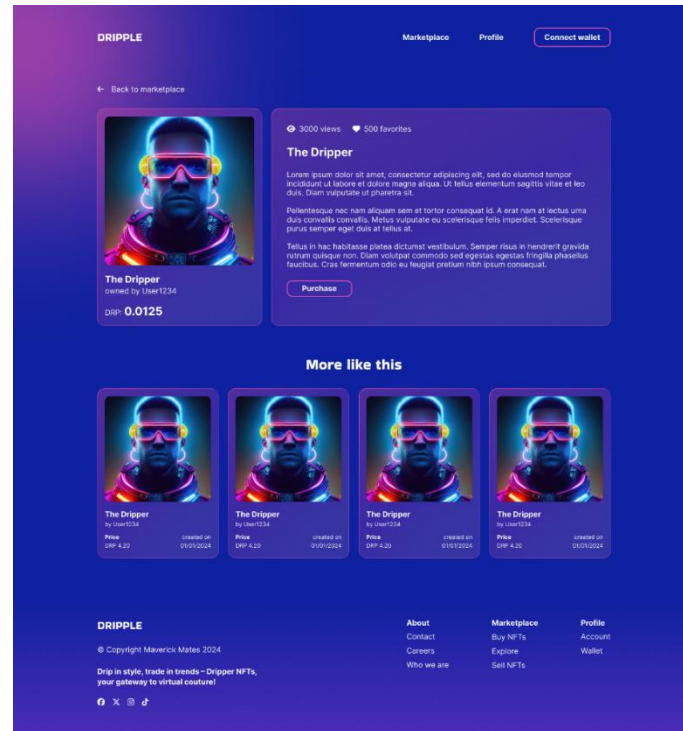


Fig. 3: The item information page.

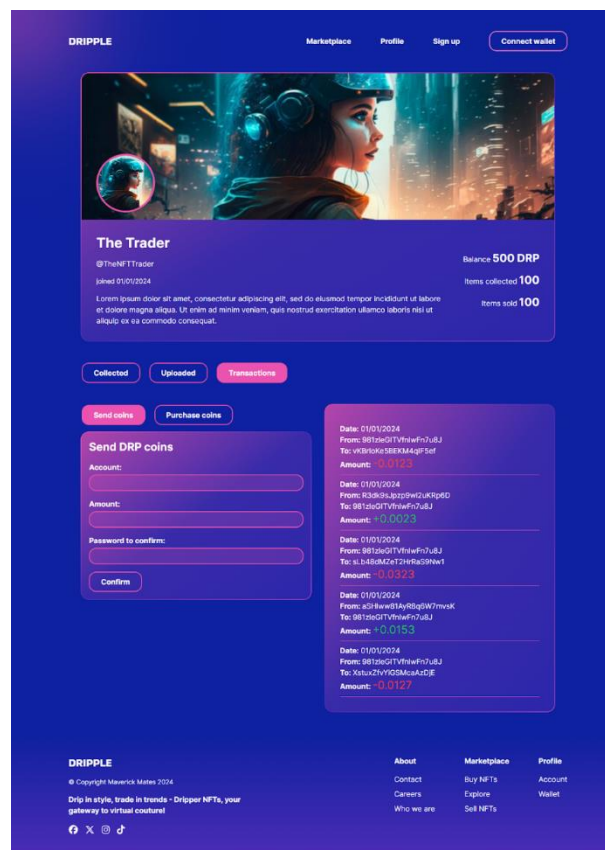


Fig. 4: The user profile page, which shows the user's transaction history.

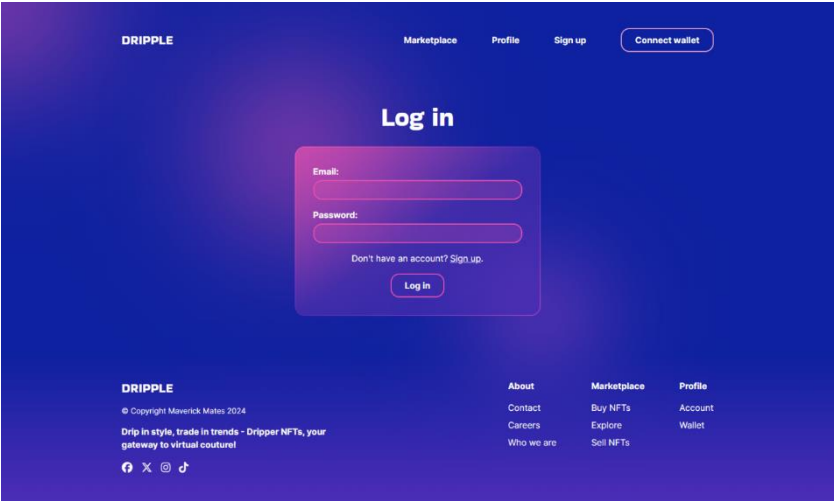


Fig. 5: The login page.

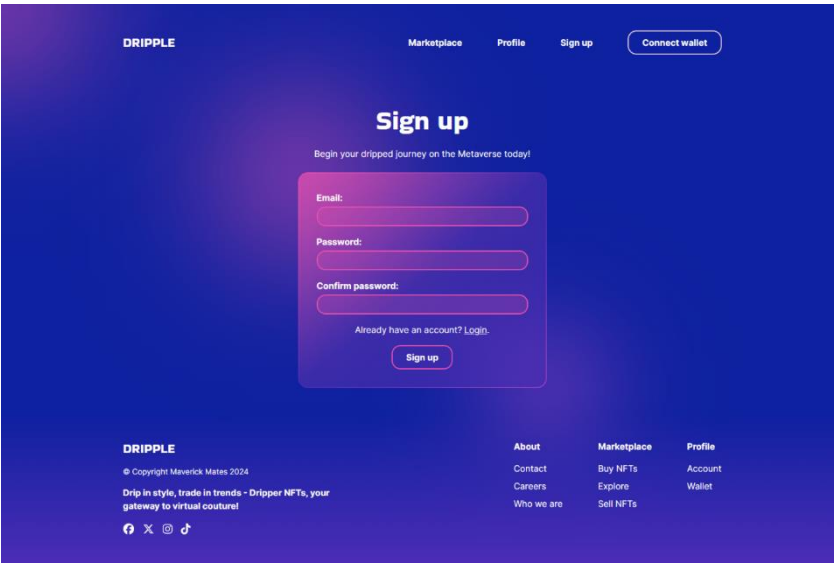


Fig. 6: The signup page.

C.Backend database design

The backend features a MySQL database that is used to store metadata related to users, NFTs, transactions, etc. The database corresponds to the data stored on the blockchain in several aspects. Below is a diagram of the database.

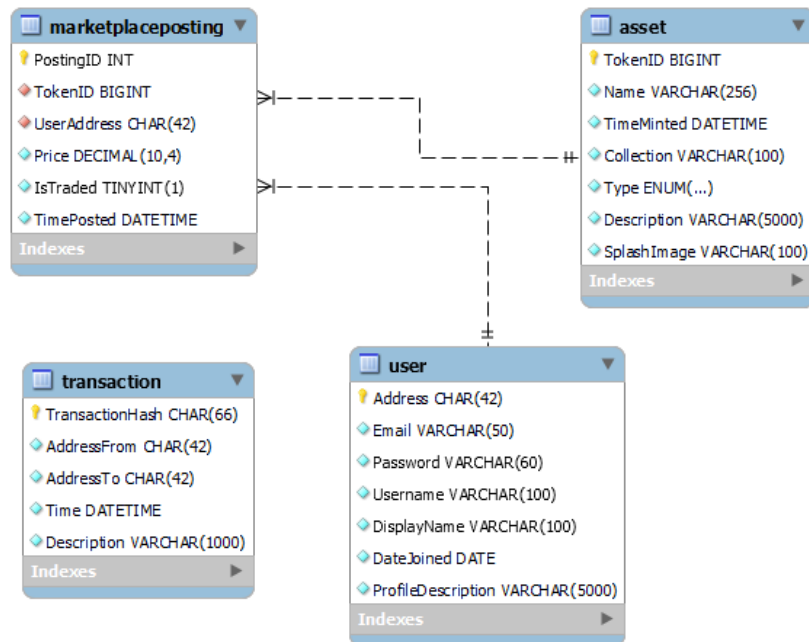


Fig. 7: The relational database schema.

The database contains 4 tables. The following section describes each table in detail and shows how the data corresponds to the blockchain.

User – Each row represents a user account on the platform.

| Column name | Type and properties | Description |
|-------------|-----------------------|---|
| Address | CHAR(42), PRIMARY KEY | Corresponds to a user address on the blockchain. |
| Email | VARCHAR(50) | The user's email. |
| Password | VARCHAR(60) | Corresponds to the password used to create the account on the blockchain. |
| Username | VARCHAR(100) | The account's username. |
| DisplayName | VARCHAR(100) | The account's display name. |

FINAL PROJECT DOCUMENT

| | | |
|--------------------|---------------|---------------------------------------|
| DateJoined | DATE | The day the user created the account. |
| ProfileDescription | VARCHAR(5000) | The user profile's description. |

Asset – Each row represents an NFT minted on the platform.

| Column name | Type and properties | Description |
|-------------|---|--|
| TokenID | BIGINT, PRIMARY KEY | Corresponds to the token ID on the blockchain. |
| Name | VARCHAR(256) | The name of the NFT. |
| TimeMinted | DATETIME | The time at which the NFT is minted. |
| Collection | VARCHAR(100) | The collection of the NFT. |
| Type | ENUM("Clothing", "Accessories", "Footwear", "Hairstyles", "Makeup", "Body Modifications") | The type of the NFT. |
| Description | VARCHAR(5000) | The description of the NFT. |
| SplashImage | VARCHAR(100) | The file name of the NFT's showcase image. |

MarketplacePosting – Each row represents an item posted on the marketplace.

| Column name | Type and properties | Description |
|-------------|--|--|
| PostingID | INT, PRIMARY KEY | The ID of the posting. |
| TokenID | BIGINT, FOREIGN KEY references Asset.TokenID | The ID of the token being sold. References a token ID in the Asset table. Corresponds to a token ID on the blockchain. |

FINAL PROJECT DOCUMENT

| | | |
|-------------|--|--|
| UserAddress | CHAR(42), FOREIGN KEY references User.Address | The address of the seller. References an address in the User table. Corresponds to a user address on the blockchain. |
| Price | DECIMAL(10, 4) | The price of the NFT. |
| IsTraded | TINYINT(1) | A flag indicating whether this posting has been purchased. |
| TimePosted | DATETIME | The time at which the NFT was uploaded to the marketplace. |

Transaction – Each row represents a transaction on the platform.

| Column name | Type and properties | Description |
|-----------------|-----------------------|---|
| TransactionHash | CHAR(66), PRIMARY KEY | Corresponds to a transaction hash on the blockchain. |
| AddressFrom | CHAR(42) | Corresponds to the “from” address in the transaction on the blockchain. |
| AddressTo | CHAR(42) | Corresponds to the “to” address in the transaction on the blockchain. |
| Time | DATETIME | The time when the transaction took place. |
| Description | VARCHAR(1000) | The description of the transaction. |

D. API design

This section describes the endpoints exposed by the Dripple API. This API makes up the backend of our website and allows the frontend to fetch data and request certain actions from the server such as authentication, minting, purchasing, etc.

The base URL of the API is <http://localhost:8000>. It should be noted that all responses returned are in JSON format.

(continued on the next page)

POST /api/signup

Creates an account for the user with the provided email and password. The user is automatically logged in afterward.

Request body

| Field name | Required | Description |
|------------|----------|--|
| email | Yes | The email associated with the account. The email must not already exist on the platform. |
| password | Yes | The account's password. |

Example request

```
POST http://localhost:8000/api/signup
{
  "email": "taquangtung2004@gmail.com",
  "password": "1234567890"
}
```

Response

| Status code | Description | Body | Example |
|---------------------------|---|---|--|
| 200 OK | The user has successfully signed up and is logged in. | The status code and a data object containing the logged-in state and the account address on the blockchain. | <pre>{ "status": 200, "data": { "isSuccessful": true, "isLoggedIn": true, "accountAddress": "0xBf521d6BcA3bdf27F251505546fFC6cd123d4148" } }</pre> |
| 400 BAD REQUEST | The email provided already exists or the user is already logged in. | The status code and a message describing the issue. | <pre>{ "status": 400, "message": "Email already exists." }</pre> |
| 500 INTERNAL SERVER ERROR | An error happened on the server. | | <pre>{ "status": 500, "message": "Something went wrong." }</pre> |

POST /api/login

Logs the user in using the provided email and password.

Request body

| Field name | Required | Description |
|------------|----------|--|
| email | Yes | The email associated with the account. |
| password | Yes | The account's password. |

Example request

```
POST http://localhost:8000/api/login
{
  "email": "taquangtung2004@gmail.com",
  "password": "1234567890"
}
```

Response

| Status code | Description | Body | Example |
|---------------------------|--|---|--|
| 200 OK | The user has successfully logged in. | The status code and a data object containing the logged-in state and the account address on the blockchain. | <pre>{ "status": 200, "data": { "isLoggedIn": true, "accountAddress": "0xBf521d6BcA3bdf27F251505546fFC6cd123d4148" } }</pre> |
| 400 BAD REQUEST | The email provided does not exist, the password is wrong, or the user has already logged in. | The status code and a message describing the issue | <pre>{ "status": 400, "message": "Email already exists." }</pre> |
| 500 INTERNAL SERVER ERROR | An error happened on the server. | | <pre>{ "status": 500, "message": "Something went wrong." }</pre> |

POST /api/logout

Logs the user out.

Example request

POST http://localhost:8000/api/logout

Response

| Status code | Description | Body | Example |
|---------------------------|---------------------------------------|---|---|
| 200 OK | The user has successfully logged out. | The status code and a data object containing the logged-in state. | <pre>{ "status": 200, "data": { "isLoggedIn": false } }</pre> |
| 500 INTERNAL SERVER ERROR | An error happened on the server. | The status code and a message describing the issue. | <pre>{ "status": 500, "message": "Something went wrong." }</pre> |

FINAL PROJECT DOCUMENT

GET /api/login

Determines if the user is logged in or not.

Example request

GET http://localhost:8000/api/login

Response

| Status code | Description | Body | Example |
|---------------------------|----------------------------------|---|--|
| 200 OK | Success. | The status code and a data object containing the logged-in state and the account address on the blockchain. | <pre>{ "status": 200, "data": { "isLoggedIn": true, "accountAddress": "0xBf521d6BcA3bdf27F251505546fFC6cd123d4148" } }</pre> |
| 500 INTERNAL SERVER ERROR | An error happened on the server. | The status code and a message describing the issue. | <pre>{ "status": 500, "message": "Something went wrong." }</pre> |

FINAL PROJECT DOCUMENT

GET /api/profile

Retrieves information on the user's profile. The user must be logged in first.

Example request

GET `http://localhost:8000/api/profile`

Response

| Status code | Description | Body | Example |
|------------------------------|----------------------------------|---|---|
| 200 OK | Success. | The status code and an object containing details of the user profile. | <pre>{ "status": 200, "accountInformation": { "address": "0xBf521d6BcA3bdf27F251505546fFC6cd123d4148", "email": "taquangtung2004@gmail.com", "username": "taquangtung2004@gmail.com", "displayName": "taquangtung2004@gmail.com", "dateJoined": "2024-03- 22T17:00:00.000Z", "profileDescription": "", "balance": "50", "itemsOwned": 0, "itemsSold": 0 } }</pre> |
| 401 UNAUTHORIZED | The user has not logged in. | The status code and a message describing the issue. | <pre>{ "status": 401, "message": "Not logged in." }</pre> |
| 500 INTERNAL SERVER ERROR | An error happened on the server. | | <pre>{ "status": 500, "message": "Something went wrong." }</pre> |

PATCH /api/profile

Updates the user's account information (username, display name, and description.) The user must be logged in first.

Request body

| Field name | Required | Description |
|--------------------|----------|---|
| username | No | The account's username. It can be omitted but must not be empty if specified. |
| displayName | No | The account's display name. It can be omitted but must not be empty if specified. |
| profileDescription | No | The account's description. |

Example request

```
PATCH http://localhost:8000/api/profile
{
  "username": "Quang Tung Ta",
  "displayName": "qtung2004",
  "profileDescription": "Virtual fashion NFT trader."
}
```

Response

| Status code | Description | Body | Example |
|---------------------------|---|---------------------------------------|--|
| 200 OK | The user profile has been updated successfully. | The status code and a status message. | { "status": 200, "message": "Updated profile successfully." } |
| 400 BAD REQUEST | The username and/or display name are empty. | | { "status": 400, "message": "Username cannot be empty." } |
| 401 UNAUTHORIZED | The user has not logged in. | | { "status": 401, "message": "Not logged in." } |
| 500 INTERNAL SERVER ERROR | An error happened on the server. | | { "status": 500, "message": "Something went wrong." } |

FINAL PROJECT DOCUMENT

GET /api/collected

Retrieves a list of NFTs owned by the user. The user must be logged in first.

Example request

GET http://localhost:8000/api/collected

Response

| Status code | Description | Body | Example |
|---------------------------|----------------------------------|---|--|
| 200 OK | Success. | The status code and an array of owned NFTs. | <pre>{ "status": 200, "assets": [{ "tokenId": 1, "name": "Ornate Flamingo Handfan", "timeMinted": "2024-03-23T12:16:59.000Z", "collection": "CyberDynasty", "type": "Accessories", "description": "A beautifully crafted hand fan which blends oriental and cyber aesthetics.", "splashImage": "splash_1711196215446.jpg" }] }</pre> |
| 401 UNAUTHORIZED | The user has not logged in. | The status code and a message describing the issue. | <pre>{ "status": 401, "message": "Not logged in." }</pre> |
| 500 INTERNAL SERVER ERROR | An error happened on the server. | | <pre>{ "status": 500, "message": "Something went wrong." }</pre> |

GET /api/collected/:tokenId

Retrieves information of an NFT owned by the user. The user must be logged in first. The NFT must exist.

Request path parameters

| Field name | Required | Description |
|------------|----------|--|
| tokenId | Yes | The token ID of the NFT. The token must exist and the user must own the token. |

Example request

GET http://localhost:8000/api/collected/1

Response

| Status code | Description | Body | Example |
|---------------------------|---|--|---|
| 200 OK | Success. | The status code and an object containing information on the requested NFT. | <pre>{ "status": 200, "assetInfo": { "tokenId": 1, "name": "Ornate Flamingo Handfan", "timeMinted": "2024-03-23T12:16:59.000Z", "collection": "CyberDynasty", "type": "Accessories", "description": "A beautifully crafted hand fan which blends oriental and cyber aesthetics.", "splashImage": "splash_1711196215446.jpg" } }</pre> |
| 401 UNAUTHORIZED | The user has not logged in or does not own the NFT. | The status code and a message describing the issue. | <pre>{ "status": 401, "message": "Not logged in." }</pre> |
| 404 NOT FOUND | The NFT token ID does not exist | | <pre>{ "status": 404, "message": "Token ID does not exist." }</pre> |
| 500 INTERNAL SERVER ERROR | An error happened on the server. | | <pre>{ "status": 500, "message": "Something went wrong." }</pre> |

GET /api/traded

Retrieves a list of NFTs that the user has uploaded to the marketplace. The user must be logged in first.

Example request

GET <http://localhost:8000/api/traded>

Response

| Status code | Description | Body | Example |
|------------------------------|----------------------------------|---|---|
| 200 OK | Success. | The status code and an array of uploaded NFTs. | <pre>{ "status": 200, "tradedItems": [{ "postingId": 2, "tokenId": 2, "name": "Neon Sunglasses", "collection": "Cyberdrip Collection", "type": "Accessories", "description": "Glasses that have a neon glow.", "splashImage": "splash_1711201354083.jpg", "price": "20.0000", "isTraded": 0, "timePosted": "2024-03- 23T13:46:34.000Z" }] }</pre> |
| 401 UNAUTHORIZED | The user has not logged in. | The status code and a message describing the issue. | <pre>{ "status": 401, "message": "Not logged in." }</pre> |
| 500 INTERNAL SERVER ERROR | An error happened on the server. | | <pre>{ "status": 500, "message": "Something went wrong." }</pre> |

GET /api/transactions

Retrieves a list of transactions sent from and to the user. The user must be logged in first.

Example request

GET http://localhost:8000/api/transactions

Response

| Status code | Description | Body | Example |
|---------------------------|----------------------------------|---|--|
| 200 OK | Success. | The status code and an array of transactions. | <pre>{ "status": 200, "transactions": [{ "hash": "0x7bae777d9f20f7d60b96cd884294ea be09b4f17d4848a1f7e43b33af769c0f46", "from": "0xBf521d6BcA3bdf27F251505546fFC6 cd123d4148", "to": "0xd395C86E63d756fbf67854539a8CcaEc d1F274AD", "time": "2024-03-23T13:42:39.000Z", "description": "Minting NFT with token 2", "value": "0.5", "gasFee": "0.00000002" }] }</pre> |
| 401 UNAUTHORIZED | The user has not logged in. | The status code and a message describing the issue. | <pre>{ "status": 401, "message": "Not logged in." }</pre> |
| 500 INTERNAL SERVER ERROR | An error happened on the server. | | <pre>{ "status": 500, "message": "Something went wrong." }</pre> |

POST /api/sendCoins

Sends the specified number of coins to an account address. The user must be logged in first.

Request body

| Field name | Required | Description |
|------------|----------|--|
| to | Yes | The recipient's address. It must be the address of a user. |
| amount | Yes | The amount to send. It must be a positive number greater than the user's balance and at most 1000. |
| password | Yes | The user's password. |

Example request

```
POST http://localhost:8000/api/sendCoins
{
  "to": "0x0BB6Aa404E165C68485Eb010bdF0Cd336c187960",
  "amount": "200",
  "password": "1234567890"
}
```

Response

| Status code | Description | Body | Example |
|---------------------------|---|---|--|
| 200 OK | The coins were successfully sent. | The status code and a message describing the issue. | <pre>{ "status": 200, "message": "Coins sent successfully." }</pre> |
| 400 BAD REQUEST | One or more of the request fields are missing or invalid. | | <pre>{ "status": 400, "message": "Amount must be a positive number." }</pre> |
| 500 INTERNAL SERVER ERROR | An error happened on the server. | | <pre>{ "status": 500, "message": "Something went wrong." }</pre> |

POST /api/buyCoins

Adds the specified number of coins to the user's balance. The user must be logged in first.

Request body

| Field name | Required | Description |
|------------|----------|----------------------------------|
| amount | Yes | The amount to buy. At most 1000. |
| password | Yes | The user's password. |

Example request

```
POST http://localhost:8000/api/buyCoins
{
  "amount": "200",
  "password": "1234567890"
}
```

Response

| Status code | Description | Body | Example |
|---------------------------|---|---|--|
| 200 OK | The coins were successfully bought. | The status code and a message describing the issue. | <pre>{ "status": 200, "message": "Coins bought successfully." }</pre> |
| 400 BAD REQUEST | One or more of the request fields are missing or invalid. | | <pre>{ "status": 400, "message": "Amount must be a positive number." }</pre> |
| 500 INTERNAL SERVER ERROR | An error happened on the server. | | <pre>{ "status": 500, "message": "Something went wrong." }</pre> |

FINAL PROJECT DOCUMENT

POST /api/mint

Mints an NFT and assigns the user as its owner. The user must be logged in first and have at least 0.5 DRP in their wallet to mint.


Request body

Note: The request body should be encoded with the `multipart/form-data` content type as it requires a file submission.

| Field name | Required | Description |
|-------------|----------|---|
| splash | Yes | An image file. The allowed file types are .jpg, .jpeg, .png, or .webp. |
| name | Yes | The name of the NFT. |
| collection | Yes | The collection to which the NFT belongs. |
| type | Yes | The type of the NFT. Must be one of 'Clothing', 'Accessories', 'Footwear', 'Hairstyles', 'Makeup', or 'Body Modifications'. |
| description | Yes | The description of the NFT. |
| password | Yes | The user's password. |

Example request

POST `http://localhost:8000/api/mint`

| Key | | Value |
|-------------|--------|---|
| splash | File ▾ |  600x600bb (8).jpg |
| name | Text ▾ | Neon Sunglasses |
| collection | Text ▾ | Cyberdrip Collection |
| type | Text ▾ | Accessories |
| description | Text ▾ | Glasses that have a neon glow. |
| password | Text ▾ | 1234567890 |

Response

| Status code | Description | Body | Example |
|-------------|---------------------------------------|---------------------------------------|---|
| 200 OK | The NFT has been minted successfully. | The status code and a status message. | <pre>{ "status": 200, "message": "Your asset has been minted successfully." }</pre> |

FINAL PROJECT DOCUMENT

| | | | |
|---------------------------|---|--|--|
| 400 BAD REQUEST | One or more of the request body fields are missing or invalid or the user does not have at least 0.5 DRP to mint. | | { "status": 400, "message": "Name cannot be empty." } |
| 401 UNAUTHORIZED | The user has not logged in. | | { "status": 401, "message": "Not logged in." } |
| 500 INTERNAL SERVER ERROR | An error happened on the server. | | { "status": 500, "message": "Something went wrong." } |

GET /api/marketplaceCount

Gets the number of marketplace postings satisfying the given set of criteria (if any). Otherwise returns the number of postings on the marketplace.

Request query parameters

| Field name | Required | Description |
|-------------|----------|--|
| searchQuery | No | The text that the names of the returned postings should contain. |
| type | No | The type of NFT to search for. If specified, must be one of 'Clothing', 'Accessories', 'Footwear', 'Hairstyles', 'Makeup', 'Body Modifications'. |

Example request

GET `http://localhost:8000/api/marketplaceCount?searchQuery=Fan&type=Clothing`

Response

| Status code | Description | Body | Example |
|---------------------------|----------------------------------|---|--|
| 200 OK | Success. | The status code and the number of postings matching the specified criteria. | <pre>{ "status": 200, "count": 2 }</pre> |
| 400 BAD REQUEST | The type field is invalid. | The status code and a message describing the issue. | <pre>{ "status": 400, "message": "Type is invalid. Must be one of: 'Clothing', 'Accessories', 'Footwear', 'Hairstyles', 'Makeup', 'Body Modifications'." }</pre> |
| 500 INTERNAL SERVER ERROR | An error happened on the server. | | <pre>{ "status": 500, "message": "Something went wrong." }</pre> |

GET /api/marketplace

Gets a list of marketplace postings satisfying the given set of criteria (if any). The maximum number of returned postings per request is 20.

Request query parameters

| Field name | Required | Description |
|-------------|----------|--|
| searchQuery | No | The text that the names of the returned postings should contain. |
| type | No | The type of NFT to search for. If specified, must be one of 'Clothing', 'Accessories', 'Footwear', 'Hairstyles', 'Makeup', 'Body Modifications'. |
| page | No | The page number. Used for pagination. Each page has at most 20 items. It must be a positive integer. Defaults to 1 if not specified. |
| sortBy | No | The attribute to sort the items by. Must be either 'TimePosted' or 'Price'. Defaults to TimePosted if not specified. |
| sortOrder | No | The order by which the items are sorted. Must be either 'asc' or 'desc'. Defaults to DESC (descending) if not specified. |

Example request

GET `http://localhost:8000/api/marketplace?searchQuery=neon&type=accessories&page=1&sortBy=price&sortOrder=desc`

Response

| Status code | Description | Body | Example |
|-------------|-------------|---|---|
| 200 OK | Success. | The status code and an array of postings matching the specified criteria. | <pre>{ "status": 200, "items": [{ "postingId": 2, "tokenId": 2, "name": "Neon Sunglasses", "timeMinted": "2024-03-23T13:42:39.000Z", "collection": "Cyberdrip Collection", "type": "Accessories", "splashImage": "splash_1711201354083.jpg", "price": "20.0000", "timePosted": "2024-03-23T13:46:34.000Z",</pre> |

FINAL PROJECT DOCUMENT

| | | | |
|---------------------------|---|---|--|
| | | | <pre> "username": "Quang Tung Ta", "displayName": "qtung2004" }] } </pre> |
| 400 BAD REQUEST | One or more request query parameters are invalid. | The status code and a message describing the issue. | <pre> { "status": 400, "message": "Type is invalid. Must be one of: 'Clothing', 'Accessories', 'Footwear', 'Hairstyles', 'Makeup', 'Body Modifications'." } </pre> |
| 500 INTERNAL SERVER ERROR | An error happened on the server. | | <pre> { "status": 500, "message": "Something went wrong." } </pre> |

GET /api/marketplace/:postingId

Gets the details of the posting with the given posting ID.

Request path parameters

| Field name | Required | Description |
|------------|----------|----------------------------------|
| postingId | Yes | The ID of the requested posting. |

Example request

GET http://localhost:8000/api/marketplace/1

Response

| Status code | Description | Body | Example |
|---------------|-----------------------------|---|---|
| 200 OK | Success. | The status code and an object containing the posting details. | <pre>{ "status": 200, "item": { "postingId": 1, "tokenId": 0, "name": "Flamingo Hand Fan", "timeMinted": "2024-03-23T08:03:13.000Z", "collection": "Kacey Productions", "type": "Accessories", "splashImage": "splash_1711180989568.jpg", "description": "A very beautiful handfan.", "price": "10.0000", "isTraded": 0, "timePosted": "2024-03-23T08:03:27.000Z", "owner": "0x66c4EEe9B83d1455DC11f4f7fa5B80D9A5fa25F6", "username": "taquangtung@gmail.com", "displayName": "taquangtung@gmail.com" } }</pre> |
| 404 NOT FOUND | The posting does not exist. | The status code and a message | <pre>{ "status": 404, "message": "Item not found." }</pre> |

FINAL PROJECT DOCUMENT

| | | | |
|------------------------------|--|--------------------------|--|
| 500 INTERNAL SERVER ERROR | An error happened on the server. | describing the issue. | { "status": 500, "message": "Something went wrong." } |
|------------------------------|--|--------------------------|--|

POST /api/marketplace

Uploads an NFT to the marketplace for sale. The user must be logged in first and own the asset.

The asset must not be already on sale.

Request body

| Field name | Required | Description |
|------------|----------|---|
| tokenId | Yes | The NFT token ID. The NFT must exist, be owned by the user, and not already be on sale. |
| price | Yes | The price of the NFT. It must be a positive number. |

Example request

POST http://localhost:8000/api/marketplace

```
{
  "tokenId": "2",
  "price": "20"
}
```

Response

| Status code | Description | Body | Example |
|---------------------------|--|---------------------------------------|---|
| 200 OK | The NFT has been successfully posted to the marketplace. | The status code and a status message. | { "status": 200, "message": "Your NFT has been posted to the marketplace." } |
| 400 BAD REQUEST | One or more of the required fields are invalid. | | { "status": 400, "message": "Token ID cannot be empty." } |
| 401 UNAUTHORIZED | The user has not logged in. | | { "status": 401, "message": "Not logged in." } |
| 500 INTERNAL SERVER ERROR | An error happened on the server. | | { "status": 500, "message": "Something went wrong." } |

POST /api/purchase

Purchases an NFT posted on the marketplace. The user must be logged in first and have enough balance to pay for the purchase. The posting must be available for purchase and not belong to the requesting user.

Request body

| Field name | Required | Description |
|------------|----------|----------------------------------|
| postingId | Yes | The ID of the marketing posting. |
| password | Yes | The user's password. |

Example request

```
POST http://localhost:8000/api/purchase
{
  "postingId": "9",
  "password": "1234567890"
}
```

Response

| Status code | Description | Body | Example |
|---------------------------|--|---------------------------------------|--|
| 200 OK | The NFT has been bought. | The status code and a status message. | { "status": 200, "message": "NFT bought successfully." } |
| 400 BAD REQUEST | One or more of the required fields are invalid. | | { "status": 400, "message": "Posting ID cannot be empty." } |
| 401 UNAUTHORIZED | The user has not logged in. | | { "status": 401, "message": "Not logged in." } |
| 404 NOT FOUND | The posting does not exist or is no longer available for purchase. | | { "status": 404, "message": "This purchase is no longer available." } |
| 500 INTERNAL SERVER ERROR | An error happened on the server. | | { "status": 500, "message": "Something went wrong." } |

E. Function description

To create a seamless user experience on the platform, our website has developed several functionalities. The following section describes in detail each suite of functionalities, including what they do and how the user can interact with them.

User authentication

To use our website, each user will need to create an account. Having an account enables the user to perform many of the website's actions such as minting NFTs or selling and buying NFTs.

To create an account, the user needs to perform the following steps:

1. Click on "Sign up" in the top navigation bar to go to the sign-up form.
2. Enter their email and password.
3. Click "Sign up" and wait a few moments for the account to be created.

Afterwards, the user will be redirected to their profile page where they can start their journey on the platform.

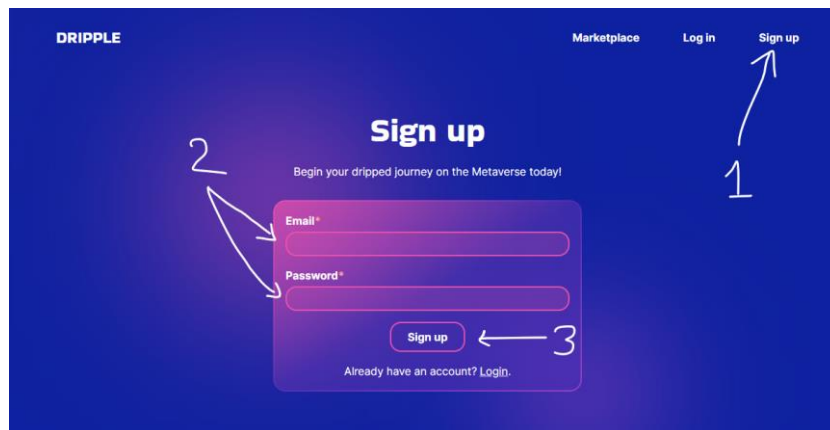


Fig. 8: The signup process.

Alternatively, if the user already has an account, they can log in instead. The steps are similar to signing up – they will go to the login form, enter their email and password, and click "Log in".

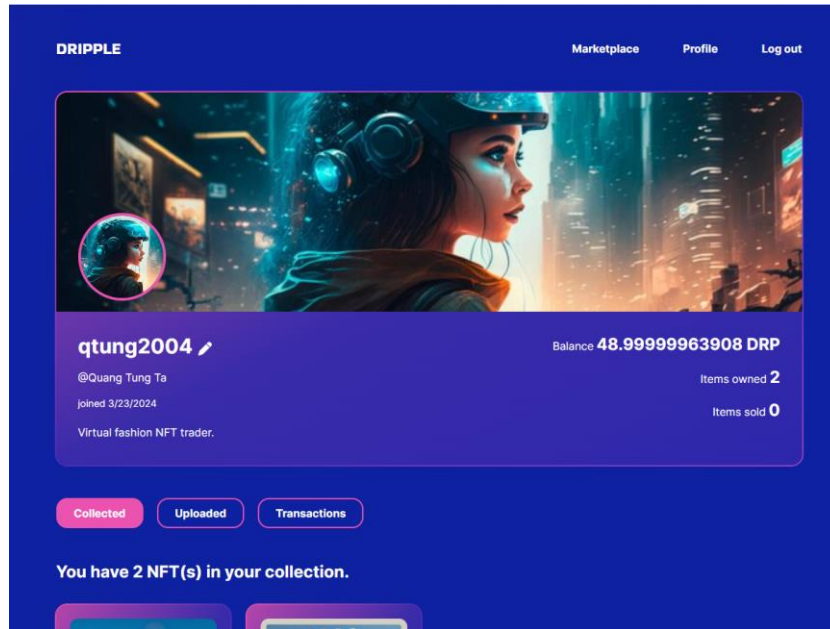


Fig. 9: The user profile page where the user is taken after signing up/logging in.

If the user wishes to switch accounts, they will need to log out first. They can easily do this by clicking “Log out” on the top navigation bar.

Profile viewing and management

Each user on the platform has a unique profile that is created when they sign up to the platform. Users can visit their profile page by clicking “Profile” on the top navigation bar (see Fig. 9). The profile page shows the user’s information such as name, balance, number of NFTs owned, etc., and contains three tabs that the user can switch around. The Collected tab lists the user’s NFTs and their information, which can be seen below.

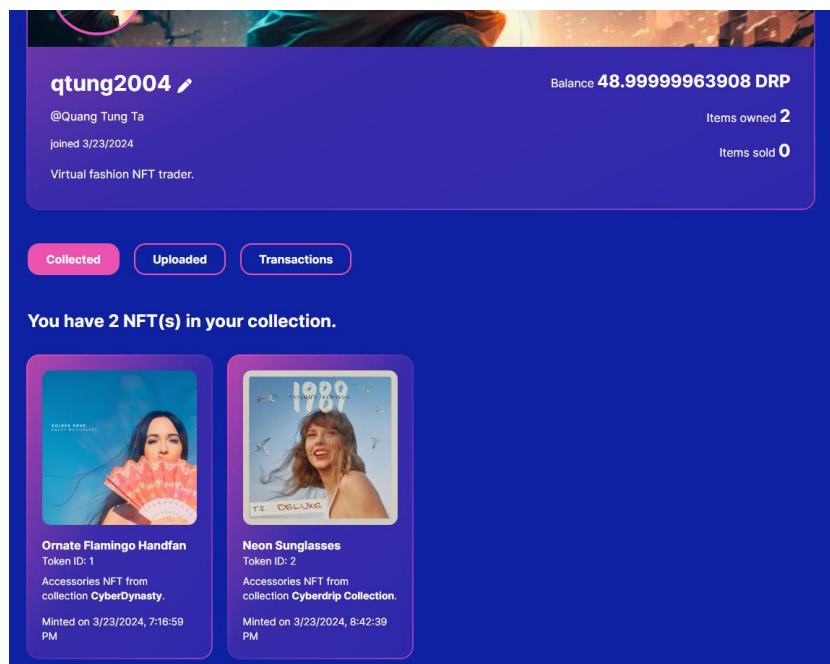


Fig. 10: The Collected tab.

The Uploaded tab lists the NFTs the user has uploaded to the marketplace and contains two forms: one for minting NFTs and another for uploading existing NFTs to the marketplace.

Mint an NFT

NFT name*
E.g. The Dropper

Collection*
E.g. Drip Collection

NFT type*
Clothing

NFT description*
E.g. My awesome NFT fashion item

Image (.png, .jpg, .jpeg, .webp only)*
Choose File No file chosen

Type your password to confirm*

Warning: Minting an asset costs a fixed fee of 0.5 DRP and incurs some negligible additional gas fees.

Confirm

Upload to marketplace

NFT*

Price*

Upload

Neon Sunglasses
Token ID: 2
Accessories NFT from collection Cyberdrip Collection.
Price: DRP 20.0000
NOT SOLD

Fig. 11: The Uploaded tab.

The Transactions tab lists the transactions the user has sent or received and features two forms where the user can buy or send coins.

Collected Uploaded Transactions

Send coins Purchase coins

Send DRP coins

Recipient's address*

Amount (maximum 1000 DRP)*

Type your password to confirm*

Confirm

3/23/2024, 8:42:39 PM
Hash: 0x7bae777d9f20f7d60b96cd884294eabe09b4f17d4848a1f7e43b33a7f69c0f46
From: 0xbf521d68cA3bdf27f251505546fC6cd123d4148
To: 0xd395C86E63d756fbf67854539a8CcaEcd1F274AD
Value: -0.5 DRP
Gas fee: -0.0000002 DRP
Minting NFT with token 2

3/23/2024, 7:16:59 PM
Hash: 0x1fdb0ce896e1cb76cea8bac766d83c8453e081589f0d73dff5e3a83dd4261e7
From: 0xbf521d68cA3bdf27f251505546fC6cd123d4148
To: 0xd395C86E63d756fbf67854539a8CcaEcd1F274AD
Value: -0.5 DRP
Gas fee: -0.0000002 DRP
Minting NFT with token 1

3/23/2024, 6:39:24 PM
Hash: 0xfba7823496d54aa81fe2942a0c39fb72a8ae2d4c6d3147652909bacc77565ed
From: 0xd395C86E63d756fbf67854539a8CcaEcd1F274ad
To: 0xbf521d68cA3bdf27f251505546fC6cd123d4148
Value: +50 DRP
Welcome to Dripple!

Fig. 12: The Transactions tab.

Sending and purchasing coins

The currency of our trading platform is DRP and users need it to make all forms of transactions. When an account is created, users are given 50 DRP for free so that they can start minting assets. However, if at any time the user runs out of coins, they can purchase more coins using the Purchase Coins form in the Transactions tab. They will need to enter the requested amount (up to 1000) and their password before submitting.

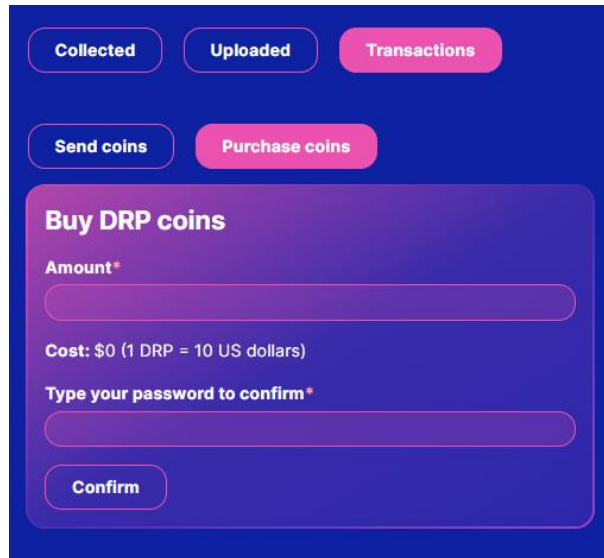
The image shows a mobile application interface with a dark blue background. At the top, there are three tabs: 'Collected', 'Uploaded', and 'Transactions'. The 'Transactions' tab is selected and highlighted in pink. Below the tabs, there are two buttons: 'Send coins' and 'Purchase coins'. The 'Purchase coins' button is selected and highlighted in pink. Below these buttons is a form titled 'Buy DRP coins'. The form has a pink header bar. Inside the form, there is a label 'Amount*' followed by a text input field. Below the input field, there is a text label 'Cost: \$0 (1 DRP = 10 US dollars)'. Below that, there is a label 'Type your password to confirm*' followed by a text input field. At the bottom of the form is a 'Confirm' button.

Fig. 13: The Purchase Coins Form.

Users can also send coins to other users using the Send Coins Form. Users cannot send coins to themselves and can only send a maximum of 1000 coins.

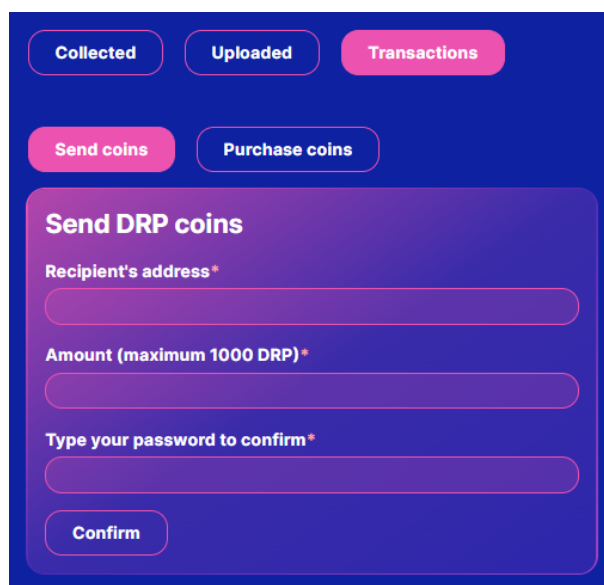
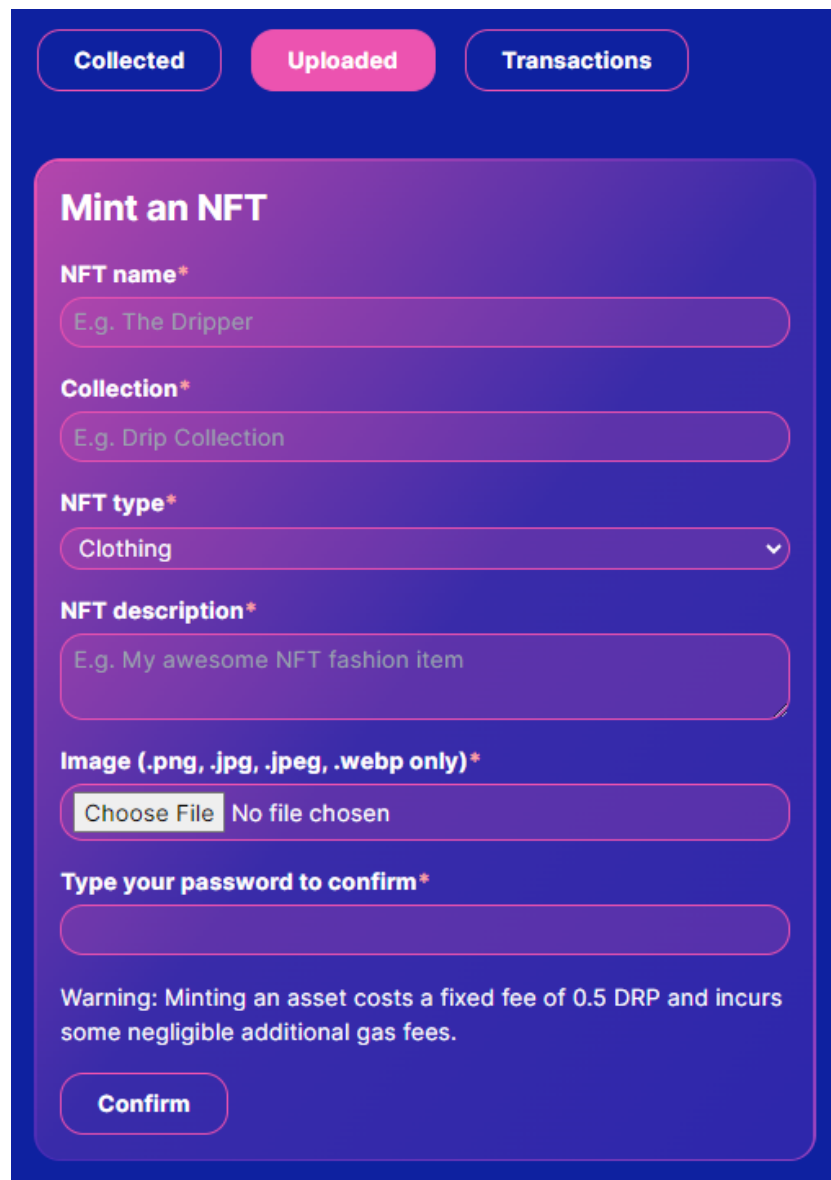
The image shows a mobile application interface with a dark blue background. At the top, there are three tabs: 'Collected', 'Uploaded', and 'Transactions'. The 'Transactions' tab is selected and highlighted in pink. Below the tabs, there are two buttons: 'Send coins' and 'Purchase coins'. The 'Send coins' button is selected and highlighted in pink. Below these buttons is a form titled 'Send DRP coins'. The form has a pink header bar. Inside the form, there is a label 'Recipient's address*' followed by a text input field. Below the input field, there is a label 'Amount (maximum 1000 DRP)*' followed by a text input field. Below that, there is a label 'Type your password to confirm*' followed by a text input field. At the bottom of the form is a 'Confirm' button.

Fig. 14: The Send Coins Form

Minting and uploading NFTs

Users can mint their NFTs using the Mint Form on the profile page. They will need to enter several required fields and have at least 0.5 DRP in their balance for minting.



The image shows a web interface for minting an NFT. At the top, there are three tabs: 'Collected', 'Uploaded' (which is highlighted in pink), and 'Transactions'. Below the tabs is a form titled 'Mint an NFT'. The form contains several input fields, each with a placeholder example: 'NFT name*' with 'E.g. The Dropper', 'Collection*' with 'E.g. Drip Collection', 'NFT type*' with a dropdown menu showing 'Clothing', and 'NFT description*' with 'E.g. My awesome NFT fashion item'. Below these is a file upload section for 'Image (.png, .jpg, .jpeg, .webp only)*' with a 'Choose File' button and the text 'No file chosen'. This is followed by a 'Type your password to confirm*' field. At the bottom of the form is a 'Warning: Minting an asset costs a fixed fee of 0.5 DRP and incurs some negligible additional gas fees.' and a 'Confirm' button.

Fig. 15: The Mint Form.

After minting, the newly created NFT will appear in the user's collected tab and can be uploaded to the marketplace. Uploading an NFT to the marketplace is as easy as specifying the NFT ID and price and can be realized through the Upload Form. The only restriction is that the user cannot re-upload the same NFT to the market if it is on sale.

Fig. 16: The Upload Form

After uploading, the NFT will appear in the Uploaded tab to let the user know that it is on the marketplace.

Browsing the marketplace and purchasing items

Users can enter the marketplace page to look for NFTs of interest. By default, items are ordered by time posted in descending order. However, users can change the way assets are filtered and sorted using the options on the search bar. They can also search for assets by name. Results are displayed in pages of 20 items max.

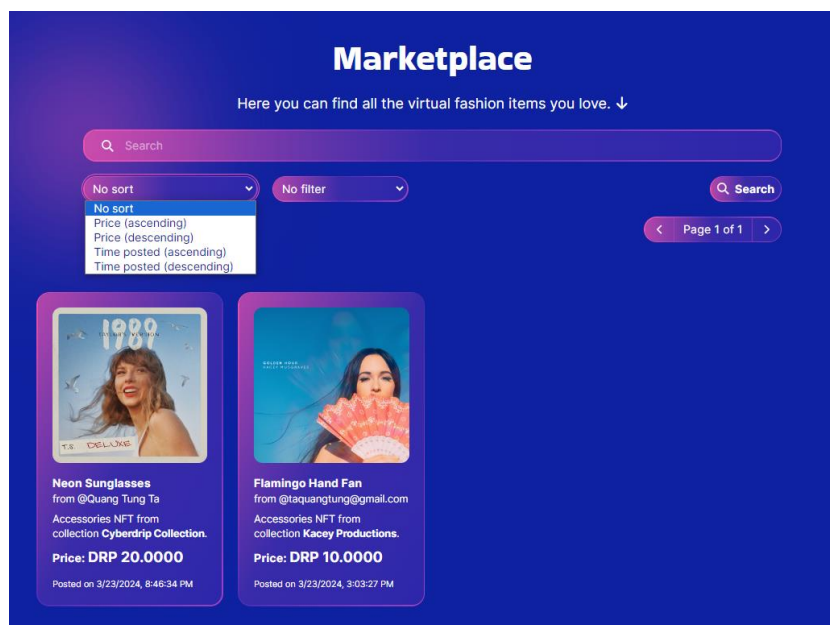


Fig. 17: The Marketplace.

After finding an item of interest, users can click on it, which will take them to the details page. Here, users can read the item's details and decide whether or not to buy it.

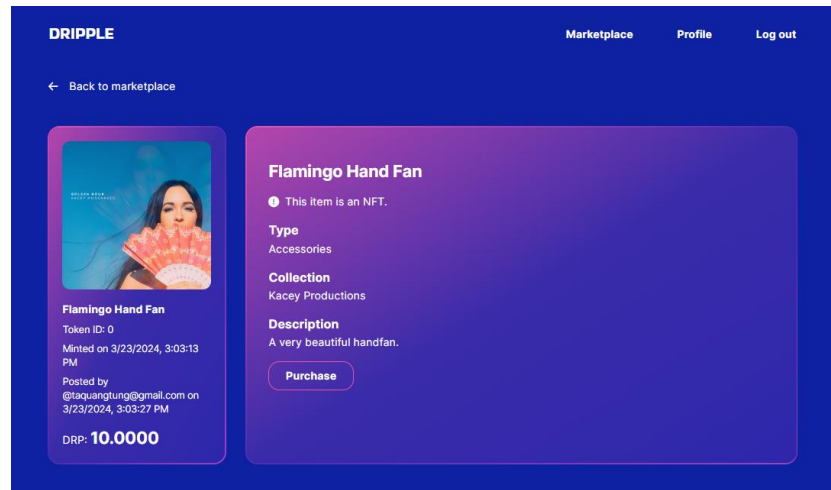


Fig. 18: The item details page.

If the user decides to buy the NFT, they will be taken to the purchase page where they can finalize their purchase and have the NFT in their collection if they have enough DRP. It should be noted that users cannot buy their NFTs, and doing so will show an error message.

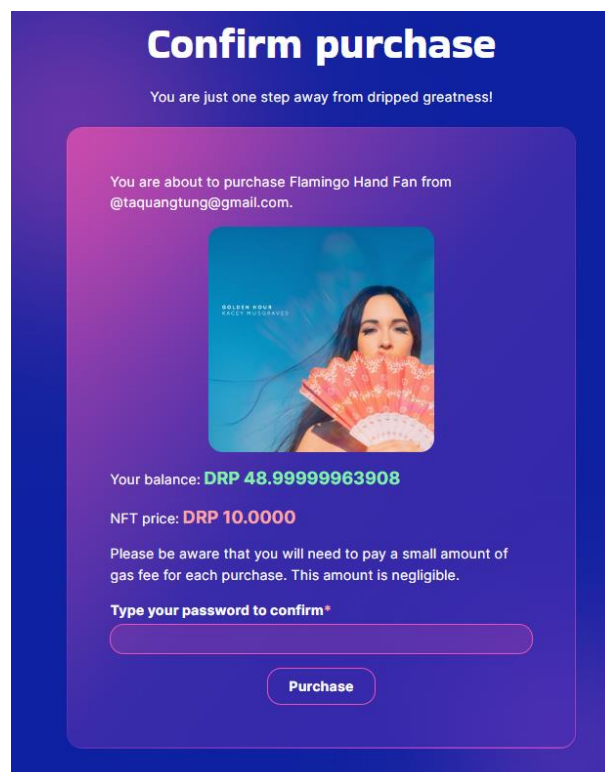


Fig. 19: The purchase page.

F. Project deployment instructions

The code for our project is available on GitHub at <https://github.com/104222196/cos30049-dripple>. You can download the source code there in addition to the Canvas zip file. Below are the instructions on how to set up the project.

This instruction assumes you use **gethnode.zip** instead of creating your own blockchain node. If you decide to create your own node, you will need to change the coinbase/contract addresses in **blockchain/truffle-config.js** and **server/.env**.

Step 1 - Getting the necessary software

To run this project, you will need:

1. **Node.js (version 20 or higher)**, we use v.20.11.0). Please go to [Node's website](#) for download instructions.
2. A package manager like **npm** or **yarn**. **npm** comes included by default with Node, so you should have it after installing Node.
3. **Geth**. Please go to [Geth's website](#) for download instructions. Please make sure to add Geth to your **PATH** environment variable.
4. **MySQL**. Please go to [MySQL's website](#) for download instructions. Alternatively, you can use **phpMyAdmin** or a similar tool to create the MySQL database.

Step 2 - Downloading or cloning this repository

You can either download this repository as a **.zip** file or clone it to your local device. After that, **cd** into the **client** and **server** directories each and run **npm install**. This will install all the necessary modules that the website depends on.

Step 3 - Building the frontend

After installing the dependencies, **cd** into the **client** directory and run **npm run build**. This will build the source code into a **dist** directory for production. Verify the **dist** folder exists after building.

Step 4 - Setting up the MySQL database

With MySQL installed and a server running, run the **schema.sql** file to create the project's database. Verify that the script correctly creates 4 tables: **asset**, **user**, **transaction**, and **marketplaceposting**.

Step 5 - Updating the environment variables

FINAL PROJECT DOCUMENT

Inside the **server** directory, open the **.env** file and change the **DATABASE_PASSWORD** field to your MySQL password.

Step 6 - Starting the Geth node

Unzip the **gethnode.zip** file into a directory of your choice and **cd** into it. If you are on Windows, you can start the Geth node by simply typing in the terminal:

gethstart

This runs a **.bat** file which we have written for your convenience. However, on other operating systems, you will need to copy and paste the following into your terminal:

```
geth --networkid 9999 --datadir ./data --port 30303 --ipcdisable --syncmode
full --http --allow-insecure-unlock --http.corsdomain "" --http.port 8545 --
http.addr localhost --unlock 0xd395C86E63d756fbf67854539a8CcaEcd1F274AD --
password ./password.txt --mine --http.api
personal,admin,eth,net,web3,miner,txpool,debug --ws --ws.addr 0.0.0.0 --
ws.port 8546 --ws.origins '' --ws.api
personal,admin,eth,net,web3,miner,txpool,debug --maxpeers 25 --
miner.etherbase 0xd395C86E63d756fbf67854539a8CcaEcd1F274AD --miner.gasprice
0 --miner.gaslimit 9999999
```

You should see your Geth node running after issuing this command.

Step 7 - Starting the server

At this point, your website is ready to be launched. **cd** into the **server** directory and type the following command in the terminal:

node index.js

This will start the server at **http://localhost:8000**. Please make sure port **8000** is free for the server to run. You can now open this address in the browser to view the full website.

Conclusion

This report has documented the design of our full-stack decentralized trading platform as part of Assignment 2 of unit COS30049 – Computing Technology Innovation Project. This project provides end users with a seamless platform to trade virtual fashion NFTs. To make this possible, the latest industry-standard tools and libraries have been used: React.js for the frontend, and Express.js, MySQL, and Solidity for the backend. Throughout this project, the team has gained valuable insights into the architecture of a decentralized web application and acquired the skills to put together its components. Some of these skills include, but are not limited to: designing an intuitive user interface, designing a logical relational database, writing API endpoints to connect the frontend with the backend, and writing smart contracts to facilitate secure crypto transactions and asset transfer. We hope that our project will make a meaningful contribution to the burgeoning crypto market.

References

Design Studio (2024). *What is Glassmorphism? UI Design Trend 2024*. Available at: <https://www.designstudiouiux.com/blog/what-is-glassmorphism-ui-trend/> (Accessed: 2 Feb. 2024).

ERC721 - OpenZeppelin Docs (n.d.). Available at: <https://docs.openzeppelin.com/contracts/3.x/erc721> (Accessed: 24 March 2024).

Jahangir Ali (2024). *NFT website - landing page design*. Available at: <https://www.behance.net/gallery/187980483/NFT-website-landing-page-design> (Accessed: 2 Feb. 2024).

OpenSea (n.d.). *Your Profile*. Available at: <https://opensea.io/account>. (Accessed: 2 Feb. 2024).