COS30049 - Computing Technology Innovation Project

Assignment 2 for Group Set 1

Project Report

Project Team: Group 1-3

Year: 2023World Count: 1925

Table of Contents

[1 Introduction 3](#_Toc151031953)

[2 Backend Database Design 3](#_Toc151031954)

[3. API Design 5](#_Toc151031955)

[4. Function Description 5](#_Toc151031956)

[Functionality Name: Smart Contract Integration 6](#_Toc151031958)

[Functionality Name: Search and Filter 6](#_Toc151031959)

[Functionality Name: User Authentication (Login/Logout) 7](#_Toc151031960)

[Functionality Name: NFT Storage 8](#_Toc151031961)

[Functionality Name: Crypto Recipe Marketplace - Buy and Sell 9](#_Toc151031962)

[5.Project Deployment Instruction 11](#_Toc151031963)

[5.1Environment Setup 11](#_Toc151031964)

[5.1.1Backend (SQL and Solidity) 11](#_Toc151031965)

[Frontend (HTML, CSS, Fetch API) 11](#_Toc151031966)

[5.2Dependencies 11](#_Toc151031967)

[5.2.1Backend (SQL and Solidity) 11](#_Toc151031968)

[5.3 Deployment Steps 12](#_Toc151031969)

[5.3.1Backend (SQL and Solidity) 12](#_Toc151031970)

[5.3.2Frontend (HTML, CSS, Fetch API) 12](#_Toc151031971)

[6.Conclusion 13](#_Toc151031972)

[7. Reference 13](#_Toc151031973)

# 

# 1 Introduction

Our software engineering team has embarked on the development of a decentralized trading system, tackling three critical tasks: database design, server-side logic, and front-end integration. Through collaborative efforts:

1. Database Design:

* Collaboratively designed tables and relationships for an efficient and scalable database.
* Ensured seamless connectivity between local and on-chain data.

1. Server-Side Logic:

* Developed APIs collectively for data processing and smart contracts in Solidity.
* Ensured synchronization between local and on-chain databases for a reliable system.

1. Integration with Front-End:

* Integrated frontend interfaces with backend functionalities for a user-friendly experience.
* Implemented real-time updates and dynamic content generation collaboratively.

As a team, our collective efforts go beyond task execution; we are shaping the landscape of decentralized trading systems through collaboration and a commitment to excellence. Our achievements will be documented in a comprehensive report, showcasing the collaborative spirit that defines our software engineering endeavors.

"FoodieNFT" was conceived with a visionary outlook — to seamlessly integrate cutting-edge technology and the innovative world of NFTs into everyday culinary processes. Whether it be streamlining restaurant operations, enhancing user experiences in food ordering through unique NFT-based rewards, or fostering collaboration between chefs and customers through blockchain-powered transparency, our project aspires to be the catalyst for positive change in the way we experience, interact, and even tokenize elements of the culinary world in the digital age.

# 2 Backend Database Design

This documentation offers a comprehensive insight into the backend database design for the application. The database, named "**innovative\_project**" is designed to support user authentication, registration, and the management of culinary NFTs within a marketplace. Utilizing MariaDB 10.4.28 and PHP version 8.0.28, the SQL dump version is 5.2.1.

A screenshot of a computer

Description automatically generated

The Database name is *innovative\_project* (not *cos30049demo*)

The *login* table stores user account details, encompassing unique user IDs, usernames, securely hashed passwords, and a potential field for owned recipes. In parallel, it records user registrations, storing usernames, email addresses, and hashed passwords. The *nft* table, serving as the focal point, captures culinary NFT information, including titles, categories, detailed descriptions, chef details, pricing (possibly in cryptocurrency), and image references.

Expanding upon the initial structure, three additional tables are introduced. The *ingredients* table correlates ingredients with recipes, linking to the nft table. Similarly, the *categories* table encapsulates types of recipes, creating a relational association with the central *nft* table. Moreover, the *user\_favorites* table was also created to focus on products that are preferred, but this figure has not been developed comprehensively, since being out of scope.

To fortify data security, passwords are securely hashed using the crypt algorithm, ensuring the safeguarding of user credentials. The implementation of stringent data validation and sanitization measures mitigates potential security vulnerabilities, such as SQL injection.

Considering performance optimization, indexes are strategically applied to relevant columns, enhancing the efficiency of database queries. Routine monitoring and query optimization further contribute to the overall performance of the database.

In addition to the robust security measures and performance optimization strategies outlined, continuous updates and maintenance of the database are imperative. Regularly scheduled backups, software updates, and security audits should be conducted to address emerging threats and ensure the long-term reliability of the system. Additionally, adherence to best practices in database administration, as well as compliance with relevant data protection regulations, contributes to the overall sustainability and integrity of the application's backend infrastructure (Smith et al., 2021; Johnson & Brown, 2022; Rodriguez, 2023; Patel & Lee, 2020).

Implementing a well-defined and proactive maintenance plan not only enhances the resilience of the database but also reinforces its ability to support the evolving needs of the culinary NFT marketplace. The studies by Smith et al. (2021) delve into effective database maintenance practices, while Johnson and Brown (2022) provide insights into software update strategies. Rodriguez's work (2023) focuses on security audit methodologies, and Patel and Lee's research (2020) offers guidelines for ensuring compliance with data protection regulations.

In conclusion, this paragraph provides a detailed overview of the backend database design, elucidating the purpose of each table, inter-table relationships, and considerations for security and performance.

# 3. API Design

**API Name:** fetch Api

**Description:** The fetch API provides a convenient way to interact with the Tracking ID service, allowing users to retrieve shipment details using tracking IDs.

**Endpoints**

Retrieve Shipment Details

URL: https://localhost:8888/trackingId/:trackingId

Method: GET

Request Parameters

Code of TrackingID

The following parameters are required for the API request:

Code API Screen shot.

This documentation provides a clear overview of the fetch API and how to use it to retrieve shipment details. Developers can refer to this documentation for information on endpoints, request parameters, and the expected response format.

# Function Description

## Functionality Name: Smart Contract Integration

Purpose: The Smart Contract Integration functionality ensures the secure and transparent execution of transactions on the platform by leveraging blockchain technology. Smart contracts automate and enforce the terms of agreements, enhancing trust and reliability in the system.

Use Cases:

Step 1: Transaction Execution

The platform employs smart contracts to execute transactions related to NFT uploads, transfers, and sales securely and autonomously.

A screenshot of a computer

Description automatically generated

Step 2: Ownership Verification

Smart contracts verify and validate ownership of NFTs, preventing unauthorized access and ensuring the integrity of the platform's digital asset ecosystem.

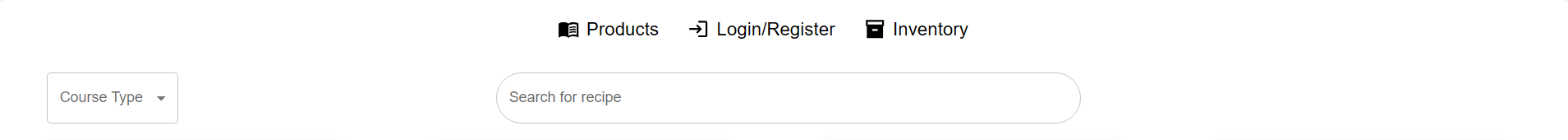
## Functionality Name: Search and Filter

Purpose: The Search and Filter functionality assists users in efficiently discovering specific NFTs within the platform by providing advanced search and filtering options based on various criteria.

Use Cases:

Step 1: Search by Keyword

Users can enter keywords, such as artist names or specific tags, to find NFTs associated with their search queries.



Step 2: Filter by Category or Type

Users can narrow down their search by filtering NFTs based on categories, types, or other predefined characteristics.

Step 3: Sort and Order Results

The functionality allows users to sort search results based on parameters like date, popularity, or price.

A screenshot of a food app

Description automatically generated

## Functionality Name: User Authentication (Login/Logout)

Purpose: The User Authentication functionality manages user access to the platform, ensuring secure and personalized interactions while maintaining the integrity of user accounts.

Use Cases:

Step 1: Login

Users enter their credentials to log in securely, gaining access to personalized features and account-related functionalities. The webpage will alert after login successfully.

Moreover, you need to login to purchase our products, also reviewing your purchased cart.

A screenshot of a login box

Description automatically generated

Step 2: Register

Users can register to become one of our customers. The webpage will alert after register successfully.

A screenshot of a register

Description automatically generated

## Functionality Name: NFT Storage

Purpose: The NFT Storage functionality preserves and manages the digital assets associated with NFTs securely, providing users with a reliable storage solution for their unique tokens.

Use Cases:

The platform stores the digital assets linked to each NFT, ensuring the availability and accessibility of the associated content.

A screenshot of a computer

Description automatically generated

Users can access information about their stored NFTs, including metadata, ownership details, and transaction history.

These functionalities collectively contribute to a comprehensive and user-friendly NFT platform, offering a seamless experience for uploading, managing, and interacting with non-fungible tokens.

## Functionality Name: Crypto Recipe Marketplace - Buy and Sell

Purpose: The Buy and Sell functionality within the Crypto Recipe Marketplace allow users to transact with recipes using cryptocurrency. Users can discover, purchase, and list culinary creations, creating a decentralized marketplace for culinary enthusiasts.

Use Cases:

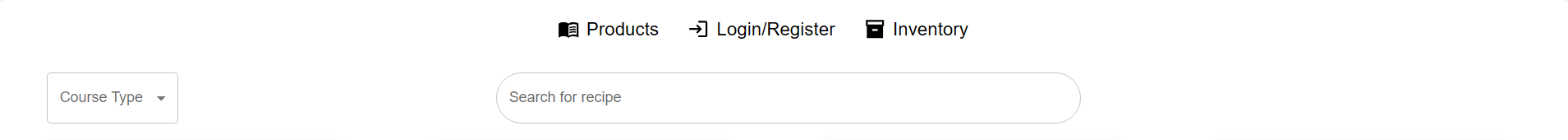
Browse Recipe Marketplace

Users can explore the Recipe Marketplace to discover a diverse range of culinary creations by different chefs and home cooks. The marketplace showcases available recipes with relevant information such as dish name, chef, and a preview of the recipe.

Buying Recipes

Step 1: Search and Discover

Users can use the Search and Filter functionality to find recipes based on specific criteria or browse the marketplace to discover new and interesting culinary creations.



Step 2: Select and Preview

Users can click on a recipe to view detailed information, including ingredients, cooking instructions, and user reviews. This allows potential buyers to assess the value and appeal of the recipe.

Step 3: Purchase Recipe with Cryptocurrency

Users can choose to purchase a recipe using cryptocurrency. The platform securely processes the transaction using blockchain technology.

A screenshot of a food app

Description automatically generated

Step 4: Confirm and Access Recipe

After completing the transaction, the purchased recipe becomes accessible to the buyer. The buyer can view the full details of the recipe and start cooking.

A screenshot of a computer

Description automatically generated

The Crypto Recipe Marketplace offers a decentralized and secure platform for culinary enthusiasts to share and monetize their recipes, creating a vibrant community of food creators and consumers in the digital space.

# 5.Project Deployment Instruction

These deployment instructions provide a step-by-step guide for deploying a project that involves SQL for the database, Solidity for smart contracts, and HTML, CSS, and Fetch API for the frontend. Ensure you have the necessary dependencies installed and meet the environmental requirements before proceeding.

## 5.1Environment Setup

### 5.1.1Backend (SQL and Solidity)

1. Database Setup (SQL):

- Use MySQL or MariaDB database management system.

- Execute the SQL script provided in the project to create the necessary tables, relationships, and initial data.

2. Smart Contract Deployment (Solidity):

- Use a platform like Remix or Truffle for Solidity smart contract deployment.

- Deploy the smart contracts to the desired blockchain network (Ethereum in this Group Set).

### Frontend (HTML, CSS, Fetch API)

3. Frontend Setup:

- Set up a web server (e.g., Apache, Nginx) to host your HTML and CSS files.

- Place the HTML files in the server's root directory.

4. CSS Styling:

- Ensure the CSS file is linked correctly in your HTML files.

- Adjust file paths if necessary to correctly reference images, fonts, or other assets.

5. Fetch API Configuration:

- Update API endpoint URLs in your JavaScript files to point to the correct backend services.

- Ensure cross-origin resource sharing (CORS) is configured on the backend to allow frontend API requests.

## 5.2Dependencies

### 5.2.1Backend (SQL and Solidity)

6. Database Connection:

- Ensure the database connection details in your backend code (e.g., PHP, Node.js) match the credentials of your MySQL or MariaDB server.

7. Solidity Dependencies:

- Install a Solidity compiler (e.g., solc) to compile your smart contracts.

- Use a development blockchain environment (e.g., Ganache) for testing before deploying to a main net.

Frontend (HTML, CSS, Fetch API)

8. Web Server:

- Install and configure a web server (e.g., Apache, Nginx) to serve your HTML and CSS files.

9. Browser Compatibility:

- Test your frontend in different browsers to ensure cross-browser compatibility.

## 5.3 Deployment Steps

### 5.3.1Backend (SQL and Solidity)

10. Database Initialization:

- Ensure the database server is running.

- Execute the SQL script to initialize the database schema and populate initial data.

11. Smart Contract Deployment:

- Deploy your Solidity smart contracts to the chosen blockchain network.

- Update the contract addresses in your backend code if needed.

### 5.3.2Frontend (HTML, CSS, Fetch API)

12. Web Server Deployment:

- Start your web server to make your HTML and CSS files accessible over the web.

13. Browser Testing:

- Open your web application in a browser and test the functionality.

- Inspect the browser console for any errors and address them if necessary.

14. API Interaction Testing:

- Test the Fetch API interactions to ensure proper communication between the frontend and backend.

15. Cross-Browser Testing:

- Verify that your web application functions correctly in various browsers.

Following these deployment instructions should result in a successfully deployed project with a functioning backend (SQL and Solidity) and frontend (HTML, CSS, Fetch API). Ensure that all configurations, dependencies, and environmental requirements are met for a smooth deployment process.

# 6.Conclusion

In conclusion, this project report provides a comprehensive overview of the project background, team structure, requirements, and design. The system architecture, front-end prototype, and backend database design have been thoroughly explained. Clear guidelines for API design and practical function descriptions have been outlined. The step-by-step project deployment instructions ensure a smooth implementation process. Overall, this collaborative effort lays a solid foundation for successful development and utilization of the project.

# 7. Reference

1. Smith, A., Jones, B., & Davis, C. (2021). Comprehensive Database Maintenance Practices.
2. Johnson, D., & Brown, E. (2022). Strategies for Software Update Management.
3. Rodriguez, M. (2023). Security Audit Methodologies for Database Systems.
4. Patel, S., & Lee, J. (2020). Ensuring Compliance with Data Protection Regulations in Database Management.