1 Software Architecture

1.1 Deployment Diagram

The deployment diagram illustrates the distribution and interaction of the components within our dApp, highlighting how the system's parts connect and collaborate to deliver a functional application.

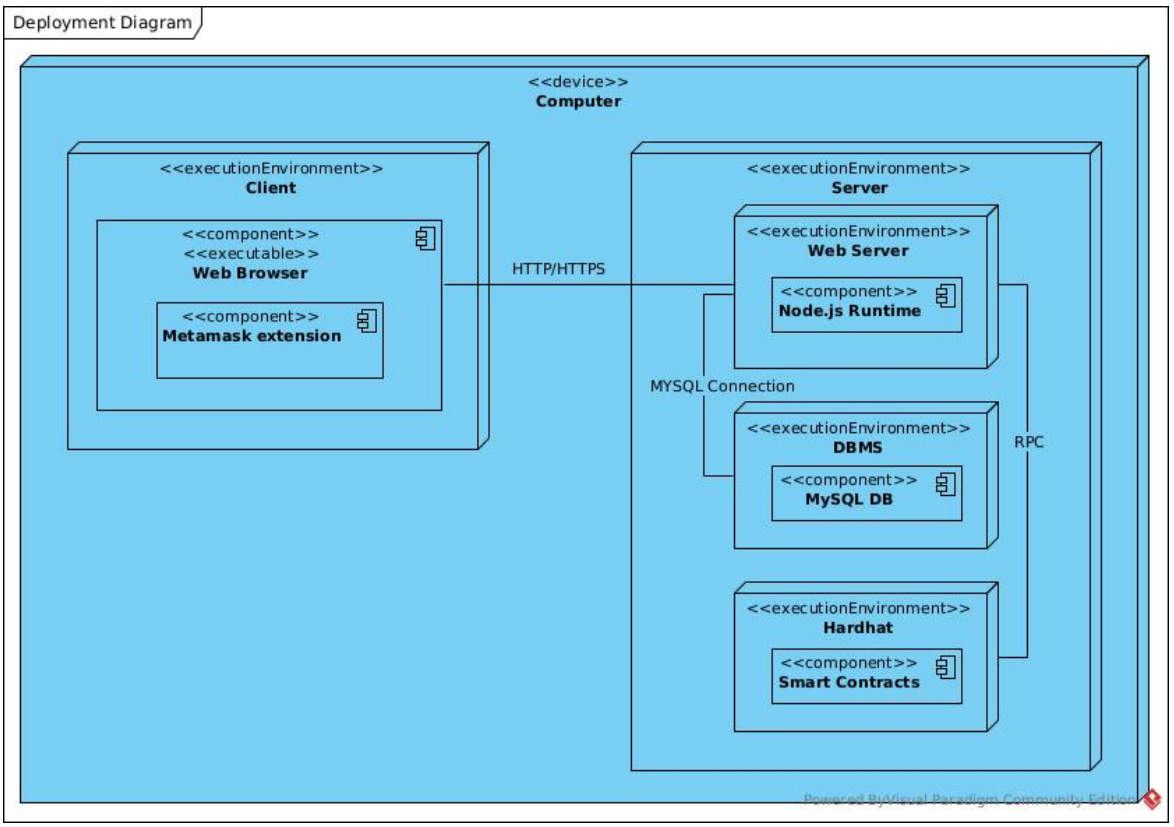

Figure 1: dApp Deployment Diagram

The web application is developed using JavaScript with Handlebars for dynamic page rendering. It facilitates interaction with the blockchain via libraries like Web3.js and Ethers.js, while MetaMask is used to manage transactions and secure wallets.

On the backend, Node.js manages the server logic, while a MySQL database stores essential information, such as wallet-username associations and game scores. Blockchain interactions are handled via Hardhat, simplifying the deployment and testing of smart contracts that govern token management and score recording.

The architecture shown reflects the development environment, where all components run on a single machine, optimal for local testing but not indicative of the final production architecture.

1.2 Smart Contract Class Diagram

The smart contract manages the token economy of the dApp, handling token creation, distribution, and transfer. Below is the class diagram of the contract, outlining its core attributes and functions.

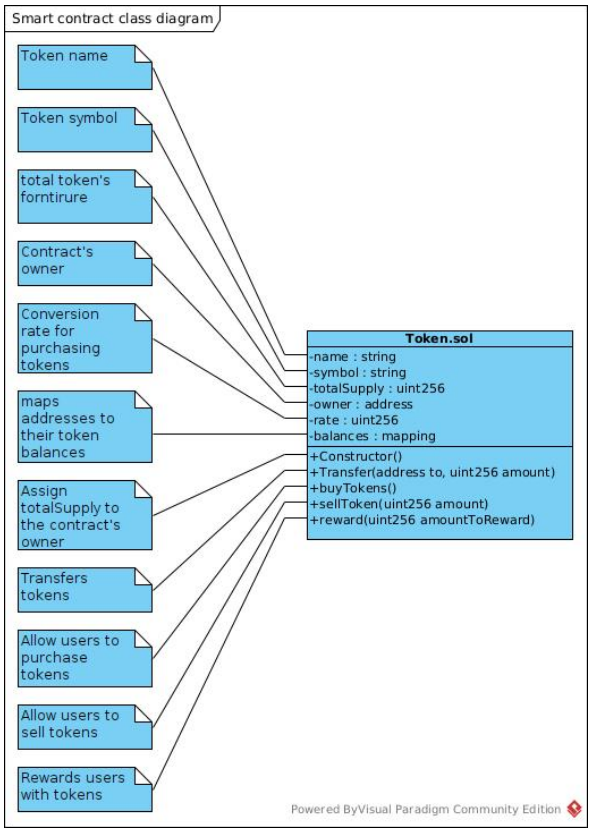

Figure 2: Smart Contract Class Diagram

The contract defines essential functions such as token transfers, purchases, and rewards. It manages user balances and token supply, while the ‘buyTokens‘ and ‘sellToken‘ functions handle the exchange between Ether and tokens. The ‘reward‘ function allows the contract owner to incentivize user participation.

1.3 Use Case Diagram

The following diagram outlines the key use cases for the Donuts Games dApp, including connecting a MetaMask wallet, playing minigames and quizzes, and managing Donuts (tokens).

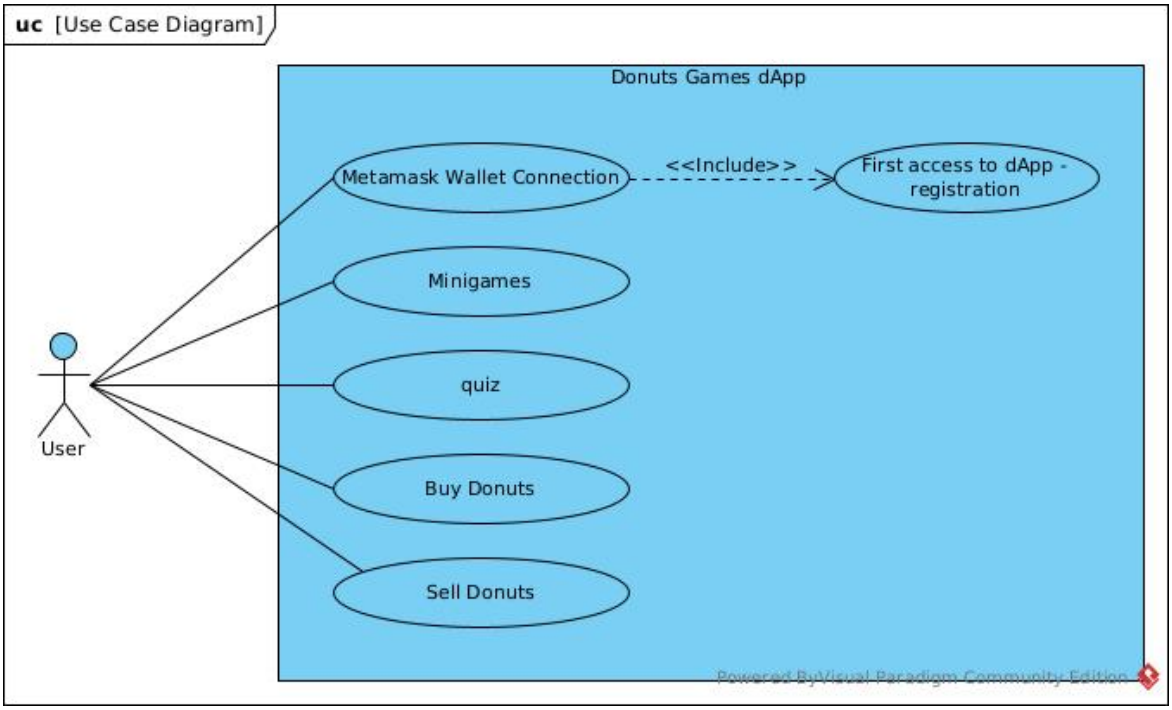

Figure 3: dApp Use Case Diagram

Users start by connecting their MetaMask wallet, which also serves as a quick registration process for new users. They can then engage in various games, quizzes, and token management activities (buying and selling Donuts) through the smart contract.

1.4 Sequence Diagrams

The sequence diagrams illustrate the interaction between the user's wallet, frontend, backend, and smart contracts during key user actions, each corresponding to the use cases previously described.

1.4.1 Wallet Connection Sequence Diagram

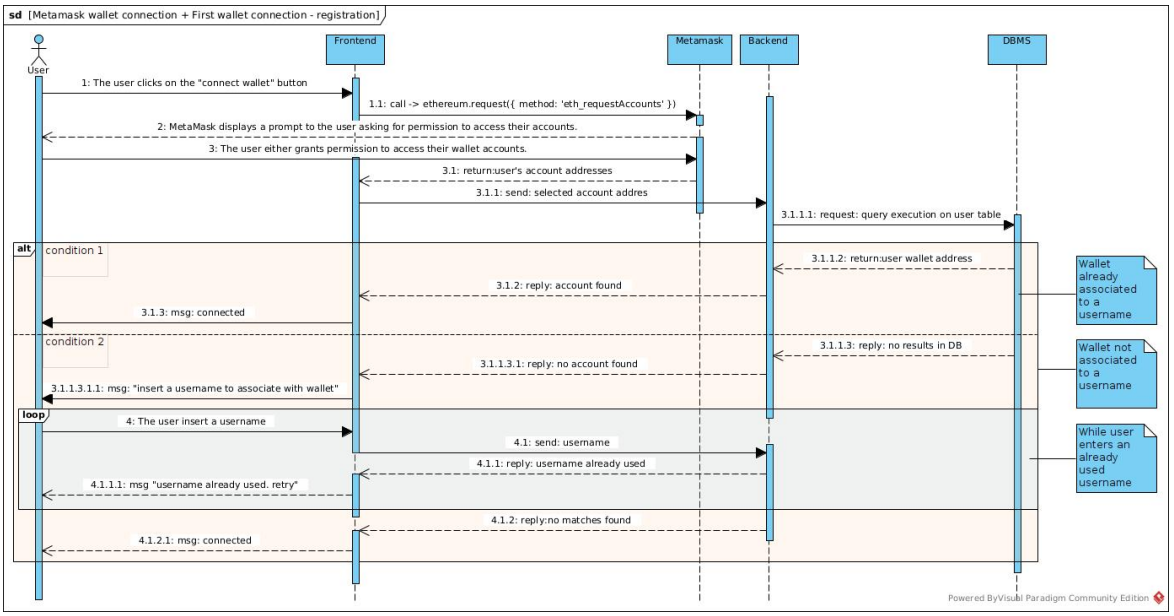

Figure 4: Wallet connection sequence diagram

This diagram shows the process of connecting a MetaMask wallet, either associating it with an existing username or creating a new profile if not previously registered. Error handling for duplicate usernames is included.

1.4.2 Minigames Sequence Diagram

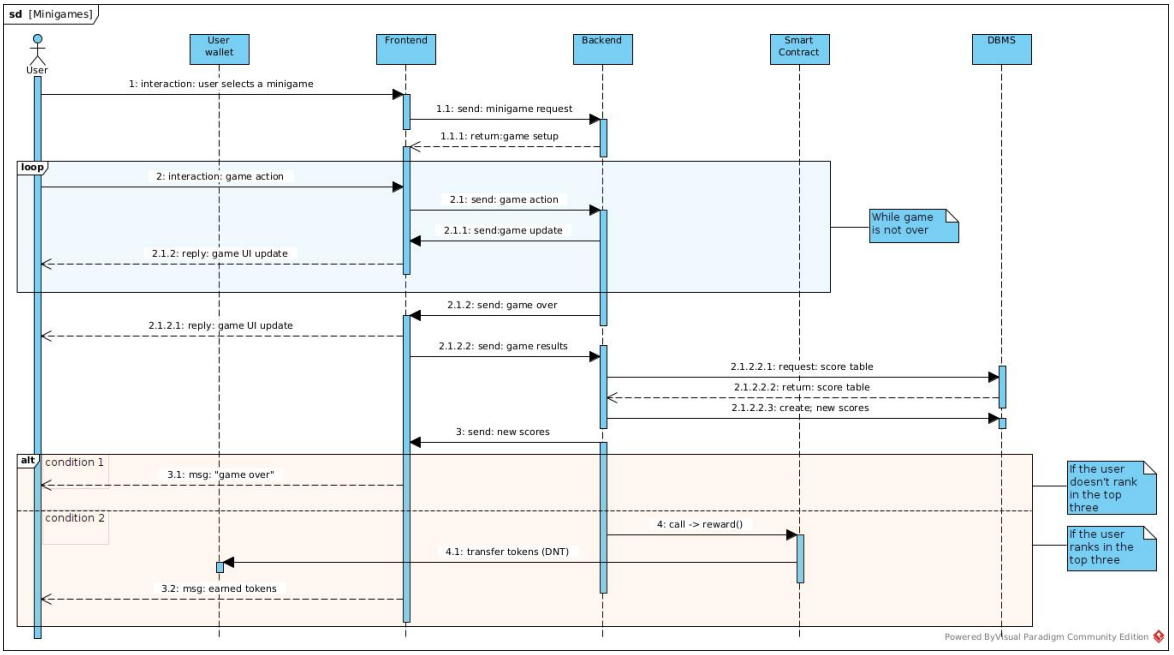

Figure 5: Minigames sequence diagram

This diagram illustrates how users engage with minigames, showing interactions between the frontend, backend, and smart contract during gameplay. Since the minigames (Tetris, Minesweeper, and Snake) share the same mechanics, they are generalized into a single use case.

1.4.3 Quiz Sequence Diagram

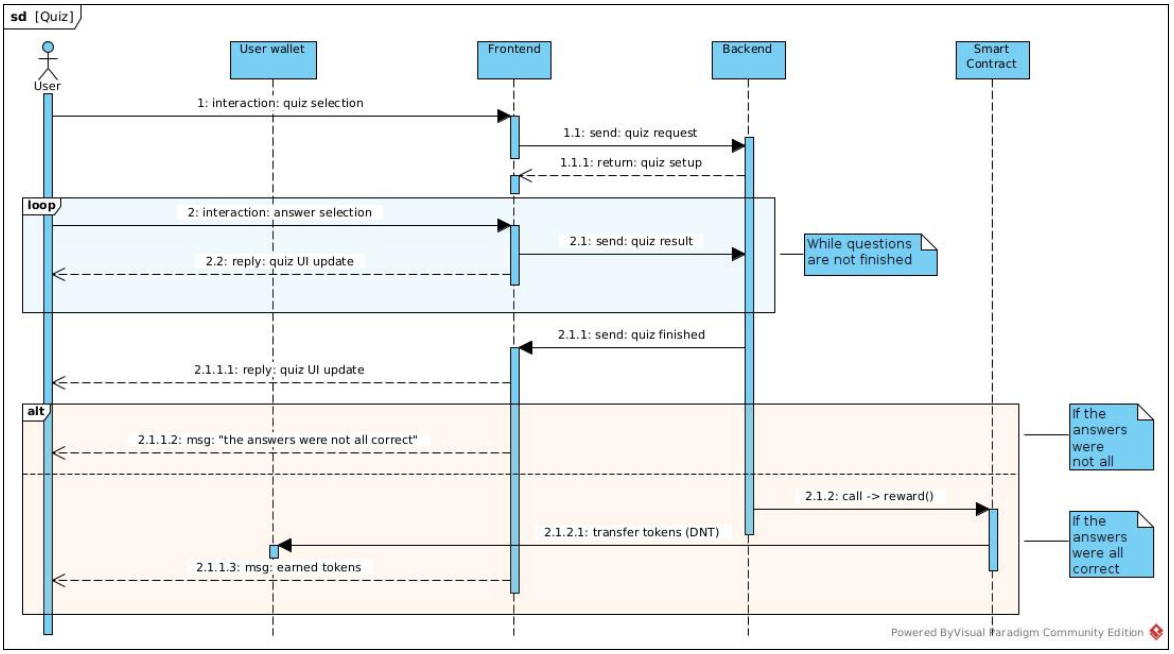

Figure 6: Quiz sequence diagram

This diagram shows the flow when users participate in quizzes. The frontend and backend present quiz questions, validate answers, and issue rewards based on correctness. The quizzes (Blockchain, Smart Contracts, and Ethereum) are generalized due to their shared structure.

1.4.4 Buy Donuts Sequence Diagram

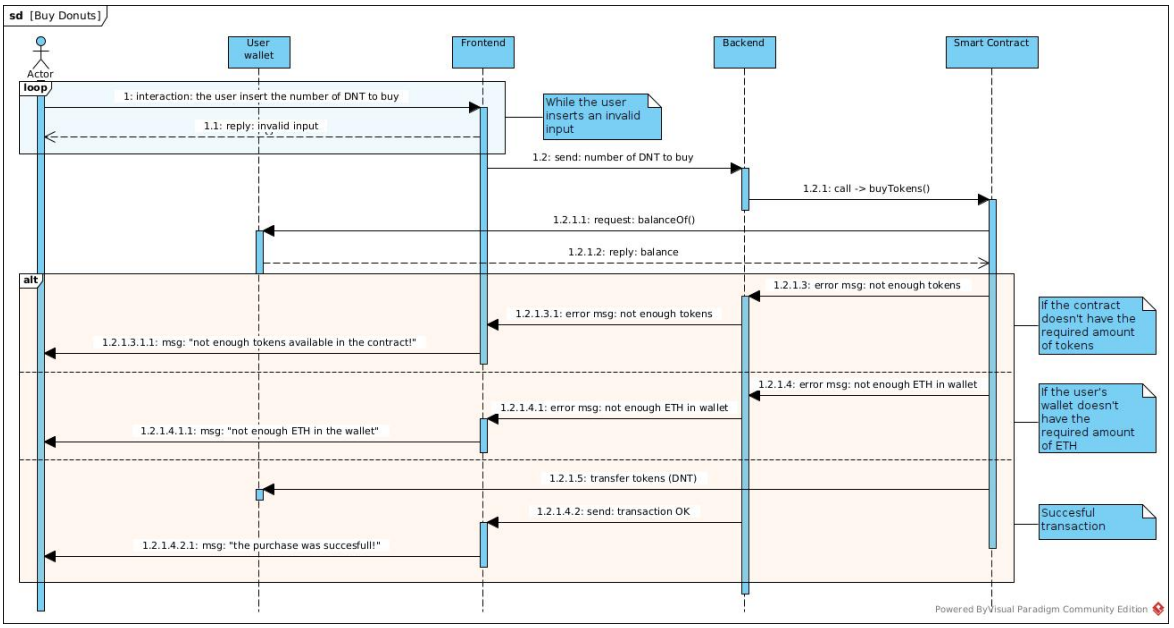

Figure 7: Buy Donuts sequence diagram

This diagram outlines the process of buying DNT tokens, verifying whether the user has enough ETH and if the contract has sufficient tokens. Error handling for insufficient funds or tokens is included.

1.4.5 Sell Donuts Sequence Diagram

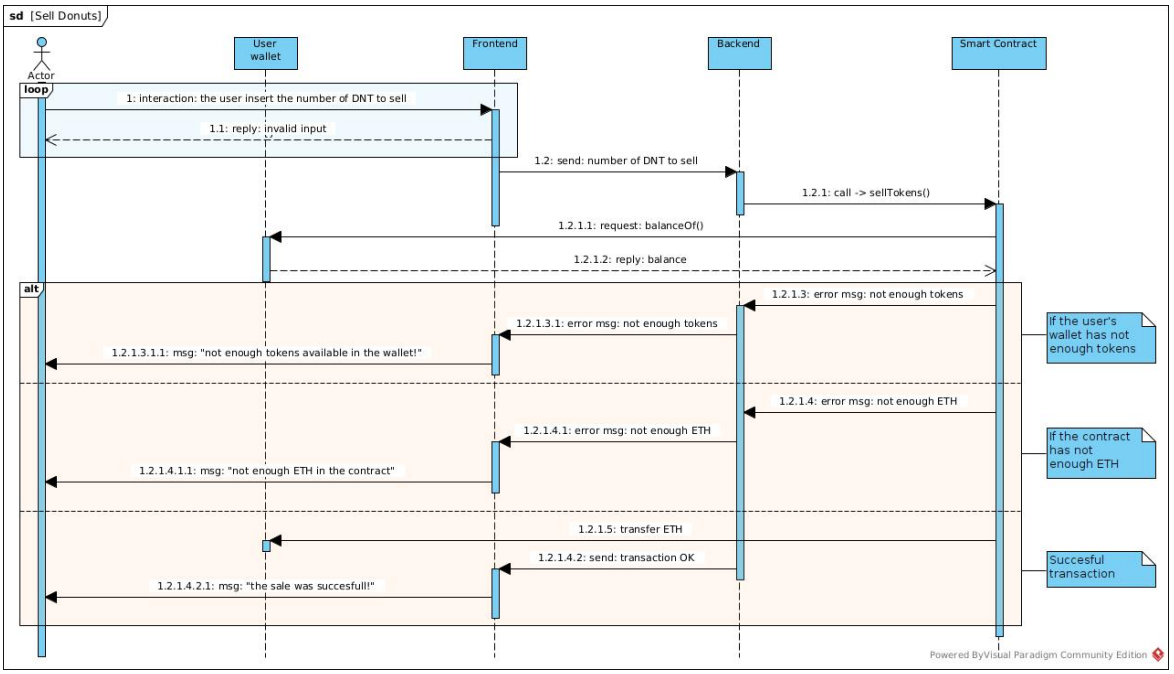

Figure 8: Sell Donuts sequence diagram

This diagram shows the process of selling DNT tokens, including balance validation and the availability of ETH for payout. Error handling for insufficient tokens or ETH is also covered.