Donuts Games - Report

A game-based decentralized app

Cybersecurity course - Blockchain and Distributed Ledgers project

Students:

Andrea Nisio - 1993155
nisio.1993155@studenti.uniroma1.it

Federico Giustozzi - 1692615
giustozzi.1692615@studenti.uniroma1.it

Stefano Bruonori - 1802004
brunori.1802004@studenti.uniroma1.it

Academic Year 2022/2023

Contents

1 Preface

As part of a university blockchain project, we developed a decentralized application (DApp) that offers an engaging experience through mini-games like Tetris, Minesweeper, and Snake. Players compete to earn Donuts (DNT) tokens, which can be used to purchase unique NFTs, traded for Ethereum (ETH), or acquired by answering quizzes on topics such as Blockchain, Smart Contracts, and Ethereum. The integration of gaming with blockchain technology provides users with a dynamic platform where they can have fun while gaining and trading digital assets. Additionally, users can monitor their progress and standings through a dedicated ”scores” section within the DApp.

- **Andrea Nisio**:
  - Solidity Development
  - JavaScript Development
  - Architectural Design
  - Documentation Design
  - Presentation
- **Stefano Brunori**:
  - Solidity Development
  - JavaScript Development
  - Token Engineering
  - Presentation
- **Federico Giustozzi**:
  - HTML Front-end Development
  - JavaScript Development
  - Presentation

This report outlines the development and functionality of our decentralized application (DApp) designed for a university blockchain project. It details the system architecture, from front-end design to back-end implementation, and the integration of smart contracts and token systems. The report also covers the deployment process, testing methodologies, and documentation provided, concluding with an evaluation of the project’s success and suggestions for future improvements.

2 Background

Blockchain technology, originally introduced as the underlying system for Bitcoin, has evolved into a versatile and disruptive innovation. The concept of a blockchain dates back to the early 1990s when Stuart Haber and W. Scott Stornetta proposed a system to timestamp digital documents to prevent backdating or tampering. However, it was not until 2008, when an individual or group under the pseudonym Satoshi Nakamoto published the Bitcoin white paper, that the blockchain concept was formalized and implemented as the foundational technology for cryptocurrency.

The rationale behind blockchain technology is to create a decentralized ledger that records transactions across a distributed network of computers. This approach enhances security and transparency by eliminating the need for a central authority and reducing the risk of single points of failure. Blockchain operates on the principle of consensus, where network participants agree on the validity of transactions through various mechanisms such as Proof of Work (PoW) or Proof of Stake (PoS). Each transaction is recorded in a "block," which is then cryptographically linked to the previous block, forming a continuous "chain" of blocks. This structure ensures that once data is recorded, it cannot be altered without altering all subsequent blocks, thus providing a high level of security and integrity.

Key concepts in blockchain technology include:

- **Decentralization**: Unlike traditional centralized databases, blockchain distributes data across a network of nodes, ensuring that no single entity has control over the entire database.

- **Consensus Mechanisms**: Techniques like Proof of Work (PoW) and Proof of Stake (PoS) are used to validate and agree on transactions within the network, maintaining data consistency and security.

- **Immutability**: Once data is recorded in a blockchain, it is extremely difficult to alter or delete, providing a permanent and tamper-evident record.

- **Smart Contracts**: Self-executing contracts with the terms of the agreement directly written into code, allowing for automated and trustless transactions between parties.

2.1 Application Domain

The application domain of blockchain technology extends far beyond its initial use case in cryptocurrency. Its potential to transform various industries is becoming increasingly evident as organizations explore its benefits for enhancing transparency, security, and efficiency.

In the financial sector, blockchain is revolutionizing traditional banking and payment systems by enabling faster, more secure transactions with reduced costs. Decentralized Finance (DeFi) platforms leverage blockchain to offer financial services such as lending, borrowing, and trading without intermediaries, thus democratizing access to financial resources.

Supply chain management is another area where blockchain technology is making significant strides. By providing an immutable record of each transaction and movement within the supply chain, blockchain enhances traceability and accountability. This capability helps in verifying the authenticity of goods, reducing fraud, and improving overall efficiency in logistics.

Healthcare is also experiencing transformative changes due to blockchain. The technology can securely manage patient records, ensuring that data is only accessible to authorized individuals while maintaining privacy and compliance with regulations such as GDPR and HIPAA. Additionally, blockchain can facilitate secure sharing of medical research data, accelerating the development of treatments and fostering collaboration among researchers.

In the realm of digital identity, blockchain offers a solution to the problem of identity theft and data breaches. By providing a decentralized and secure way to manage digital identities, individuals can have greater control over their personal information and reduce the risk of unauthorized access.

Overall, the application domain of blockchain technology is vast and growing. Its ability to provide secure, transparent, and efficient solutions has the potential to disrupt and enhance various sectors, driving innovation and creating new opportunities for businesses and individuals alike.

3 Context Presentation

The primary aim of our decentralized application (DApp) is to create an engaging and rewarding platform that combines gaming with blockchain technology. By integrating classic mini-games such as Tetris, Minesweeper, and Snake, the DApp not only provides entertainment but also incentivizes player participation through a unique token economy.

The DApp allows players to earn Donuts (DNT) tokens based on their performance in these games. These tokens can then be used to purchase exclusive NFTs, which represent unique digital assets within the platform. Additionally, players have the option to trade DNT tokens for Ethereum (ETH), thereby bridging the gap between virtual rewards and real-world value. Another significant feature of the DApp is the inclusion of quizzes related to blockchain technology. Players can earn DNT tokens by correctly answering questions on Blockchain, Smart Contracts, and Ethereum, further enhancing their engagement and knowledge.

By combining gaming with a blockchain-based reward system, the DApp aims to:

- **Engage Users**: Provide an interactive gaming experience that keeps users entertained while incentivizing their participation with valuable rewards.

- **Educate Users**: Offer educational content in the form of quizzes to improve users' understanding of blockchain technology.

- **Create Value**: Enable users to earn and trade digital assets, adding real-world value to their achievements within the platform.

Overall, the DApp seeks to merge fun and functionality, creating a compelling platform that not only entertains but also educates and rewards its users.

3.1 Why Using a Blockchain, and What Type Thereof to Use in Production

Blockchain technology is central to the functionality and value proposition of our DApp. The decision to use a blockchain was driven by several key factors:

- **Decentralization**: Blockchain provides a decentralized framework, ensuring that the game data and token transactions are not controlled by a single entity. This decentralization enhances transparency and reduces the risk of fraud or manipulation.

- **Security**: Blockchain's cryptographic nature ensures that transactions are secure and tamper-proof. Each transaction is recorded in a block and linked to previous transactions, making it nearly impossible to alter past records.

- **Transparency**: All transactions and token transfers are publicly recorded on the blockchain, allowing for full transparency. This transparency builds trust among users and ensures the integrity of the reward system.
- **Token Economy**: Blockchain enables the creation and management of digital tokens (DNT) that can be used within the platform. These tokens can be traded or redeemed for real-world value, providing users with tangible benefits.

When choosing the type of blockchain for production, several factors need to be considered:

- **Public vs. Private Blockchains**: For our DApp, a public blockchain like Ethereum is preferred due to its widespread adoption, security features, and support for smart contracts. Public blockchains offer greater transparency and are more suitable for applications that require open and trustless interactions.
- **Scalability and Cost**: While public blockchains provide numerous benefits, they may face issues related to scalability and transaction costs. To address these challenges, Layer 2 solutions such as rollups or sidechains can be considered to enhance scalability and reduce costs without compromising security.
- **Smart Contract Support**: The chosen blockchain must support smart contracts, which are essential for automating token transactions and implementing game logic. Ethereum, with its robust support for smart contracts, is a suitable choice for this purpose.

In conclusion, the use of blockchain technology is pivotal for achieving the goals of our DApp, providing a secure, transparent, and decentralized platform. Ethereum, with its support for smart contracts and its established ecosystem, is the chosen blockchain for production, ensuring that the DApp meets its objectives effectively and efficiently.

4 Software Architecture

4.1 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.

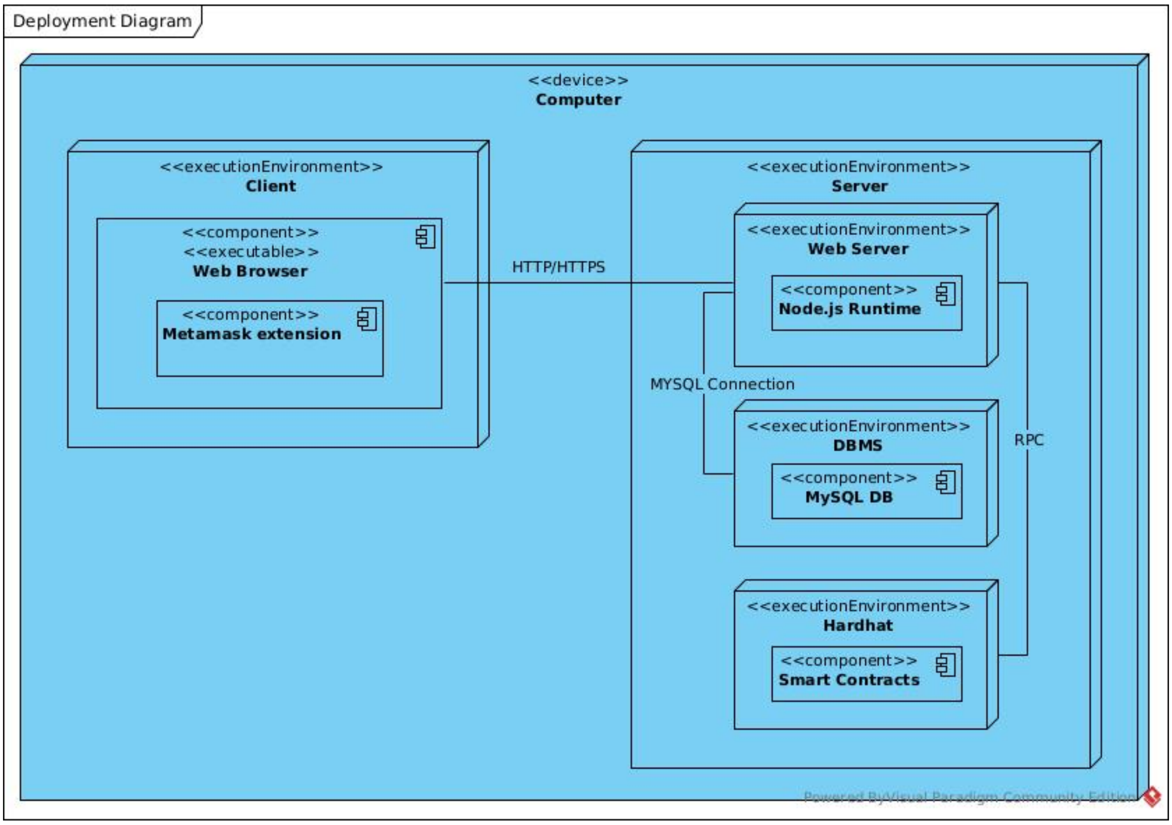

Figure 1: dApp Deployment Diagram

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.

4.2 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.

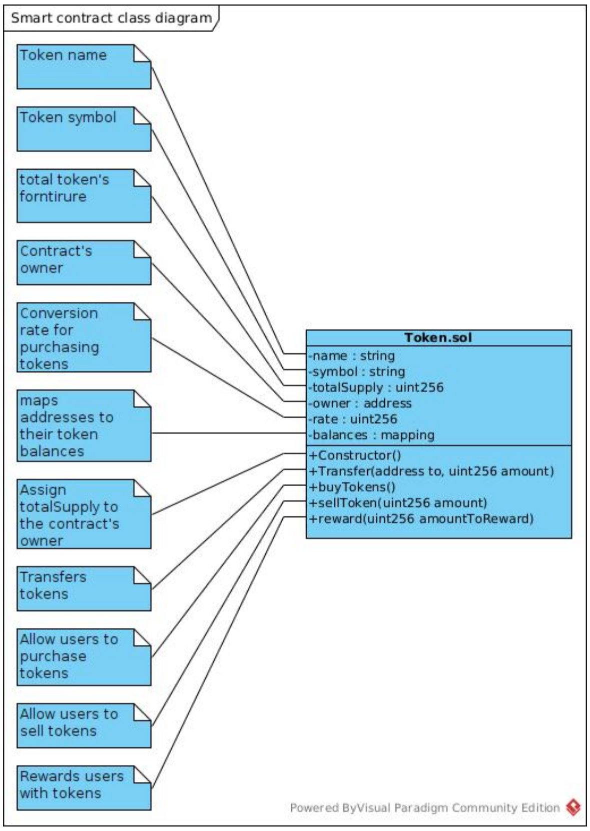
Smart contract class diagram
Token name
Token symbol
total token's forntirure
Contract's owner
Conversion rate for purchasing tokens
maps addresses to their token balances
Assign totalSupply to the contract's owner
Transfers tokens
Allow users to purchase tokens
Allow users to sell tokens
Rewards users with tokens
Token.sol
-name : string
-symbol : string
-totalSupply : uint256
-owner : address
-rate : uint256
-balances : mapping
+Constructor()
+Transfer(address to, uint256 amount)
+buyTokens()
+sellToken(uint256 amount)
+reward(uint256 amountToReward)
Powered ByVisual Paradigm Community Edition

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.

4.3 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).

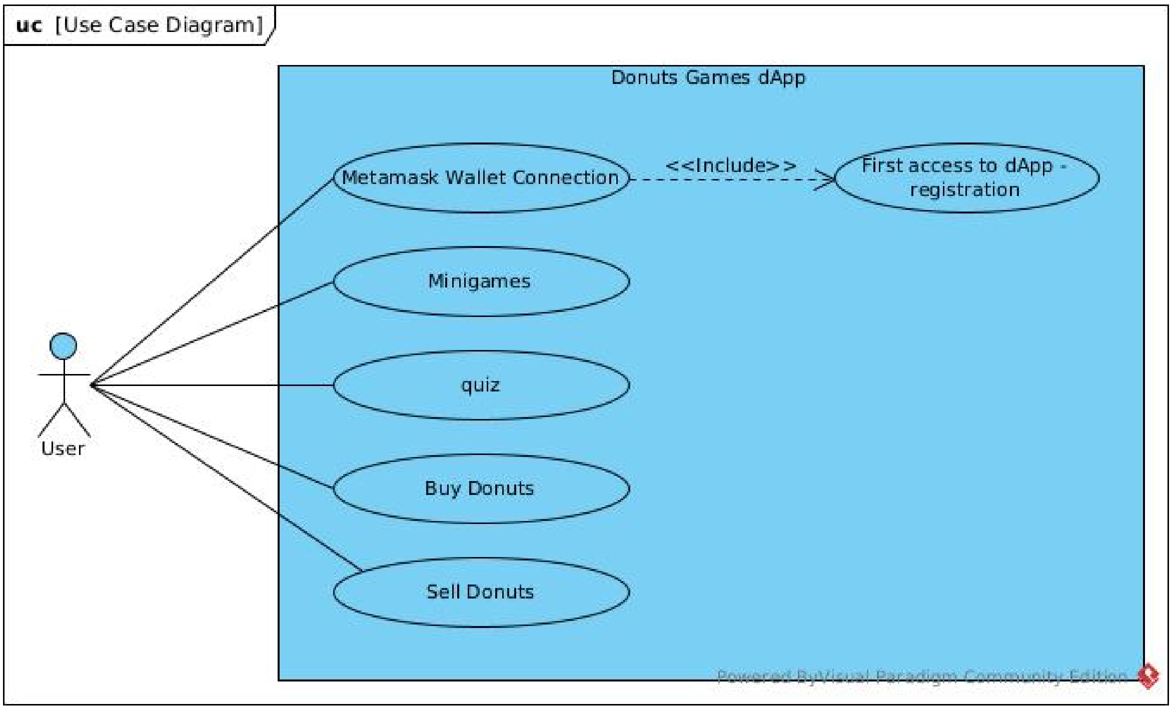
uc [Use Case Diagram]
Donuts Games dApp
Metamask Wallet Connection
<<Include>>
First access to dApp - registration
Minigames
quiz
Buy Donuts
Sell Donuts
User
Powered By Visual Paradigm Community Edition

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.

4.4 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.

4.4.1 Wallet Connection Sequence Diagram

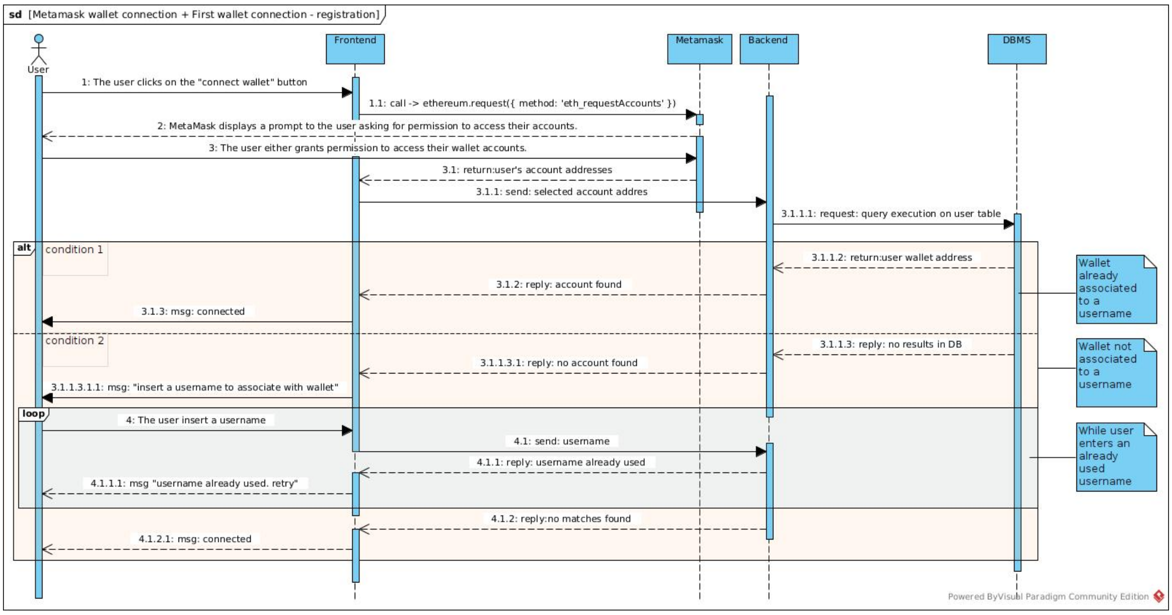
sd [Metamask wallet connection + First wallet connection - registration]
User
Frontend
Metamask
Backend
DBMS
1: The user clicks on the "connect wallet" button
1.1: call -> ethereum.request({ method: 'eth_requestAccounts' })
2: MetaMask displays a prompt to the user asking for permission to access their accounts.
3: The user either grants permission to access their wallet accounts.
3.1: return:user's account addresses
3.1.1: send: selected account addres
3.1.1.1: request: query execution on user table
alt
condition 1
3.1.1.2: return:user wallet address
3.1.2: reply: account found
3.1.3: msg: connected
Wallet already associated to a username
condition 2
3.1.1.3: reply: no results in DB
3.1.1.3.1: reply: no account found
3.1.1.3.1.1: msg: "insert a username to associate with wallet"
Wallet not associated to a username
loop
4: The user insert a username
4.1: send: username
4.1.1: reply: username already used
4.1.1.1: msg "username already used. retry"
While user enters an already used username
4.1.2: reply:no matches found
4.1.2.1: msg: connected
Powered ByVisual Paradigm Community Edition

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.

4.4.2 Minigames Sequence Diagram

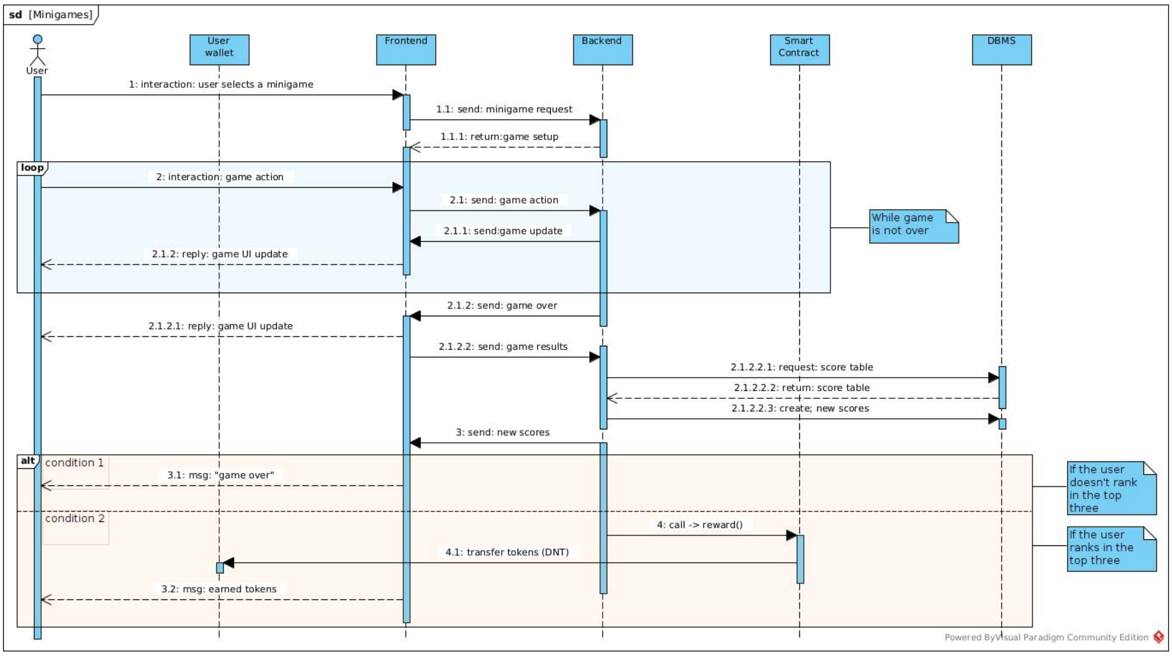
sd [Minigames]
User
User wallet
Frontend
Backend
Smart Contract
DBMS
1: interaction: user selects a minigame
1.1: send: minigame request
1.1.1: return:game setup
loop
2: interaction: game action
2.1: send: game action
2.1.1: send:game update
2.1.2: reply: game UI update
While game is not over
2.1.2: send: game over
2.1.2.1: reply: game UI update
2.1.2.2: send: game results
2.1.2.2.1: request: score table
2.1.2.2.2: return: score table
2.1.2.2.3: create; new scores
3: send: new scores
alt
condition 1
3.1: msg: "game over"
If the user doesn't rank in the top three
condition 2
4: call -> reward()
4.1: transfer tokens (DNT)
3.2: msg: earned tokens
If the user ranks in the top three
Powered ByVisual Paradigm Community Edition

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.

4.4.3 Quiz Sequence Diagram

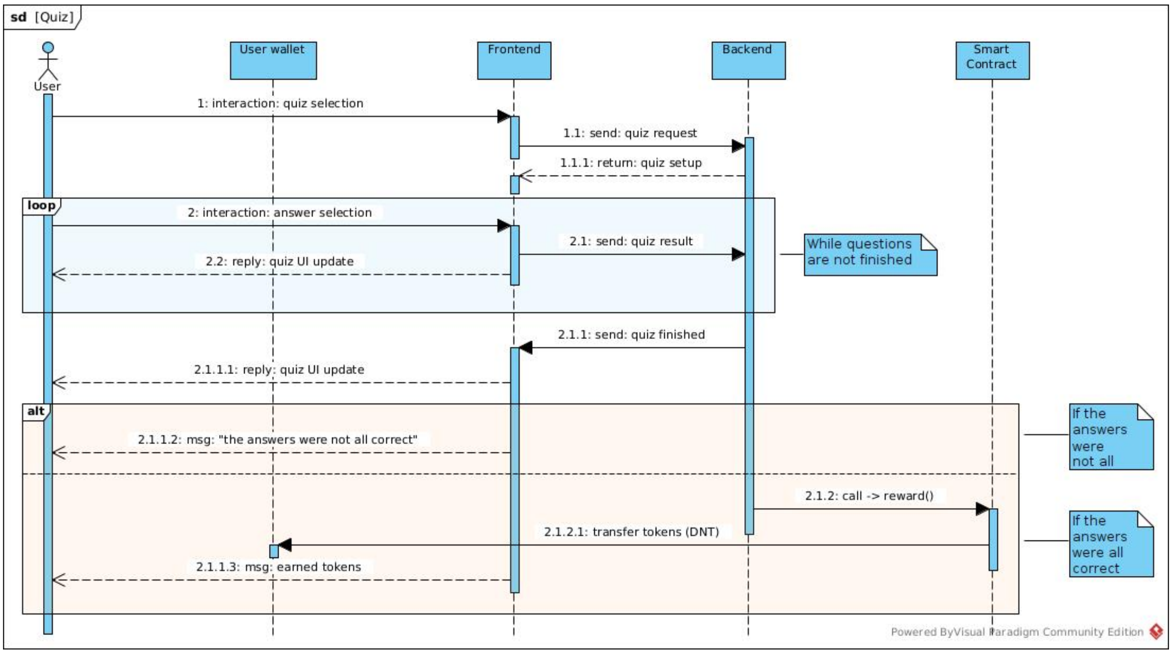
sd [Quiz]
User
User wallet
Frontend
Backend
Smart Contract
1: interaction: quiz selection
1.1: send: quiz request
1.1.1: return: quiz setup
loop
2: interaction: answer selection
2.1: send: quiz result
2.2: reply: quiz UI update
While questions are not finished
2.1.1: send: quiz finished
2.1.1.1: reply: quiz UI update
alt
2.1.1.2: msg: "the answers were not all correct"
If the answers were not all
2.1.2: call -> reward()
2.1.2.1: transfer tokens (DNT)
2.1.1.3: msg: earned tokens
If the answers were all correct
Powered ByVisual Paradigm Community Edition

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.

4.4.4 Buy Donuts Sequence Diagram

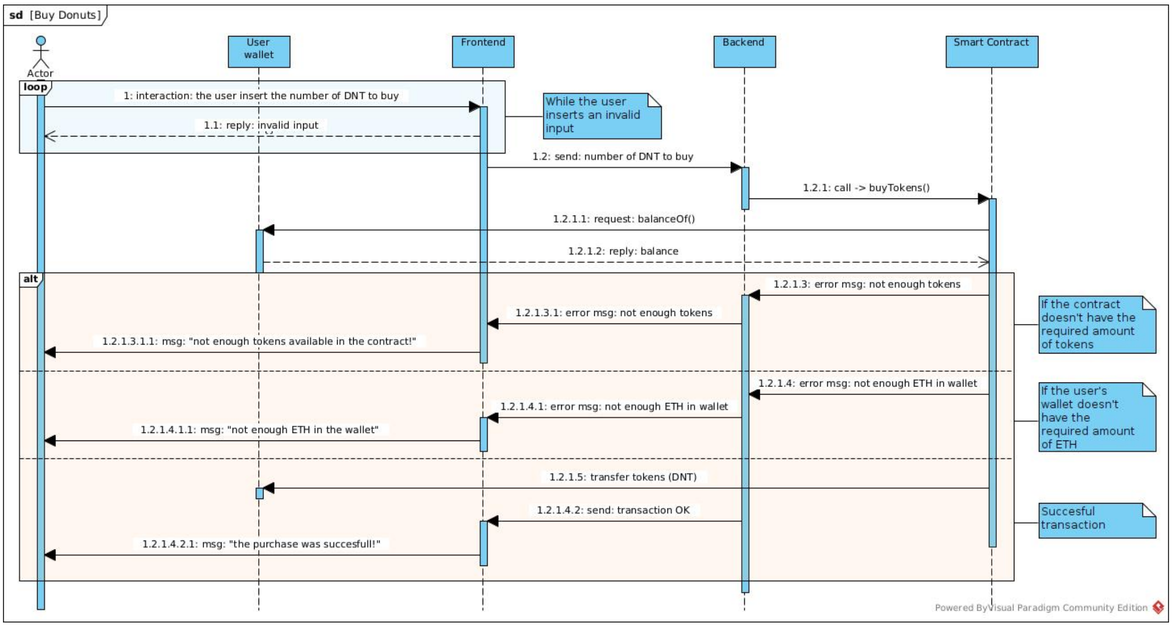
sd [Buy Donuts]
Actor
User wallet
Frontend
Backend
Smart Contract
loop
1: interaction: the user insert the number of DNT to buy
1.1: reply: invalid input
While the user inserts an invalid input
1.2: send: number of DNT to buy
1.2.1: call -> buyTokens()
1.2.1.1: request: balanceOf()
1.2.1.2: reply: balance
alt
1.2.1.3: error msg: not enough tokens
1.2.1.3.1: error msg: not enough tokens
1.2.1.3.1.1: msg: "not enough tokens available in the contract!"
If the contract doesn't have the required amount of tokens
1.2.1.4: error msg: not enough ETH in wallet
1.2.1.4.1: error msg: not enough ETH in wallet
1.2.1.4.1.1: msg: "not enough ETH in the wallet"
If the user's wallet doesn't have the required amount of ETH
1.2.1.5: transfer tokens (DNT)
1.2.1.4.2: send: transaction OK
1.2.1.4.2.1: msg: "the purchase was succesfull!"
Succesful transaction
Powered By Visual Paradigm Community Edition

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.

4.4.5 Sell Donuts Sequence Diagram

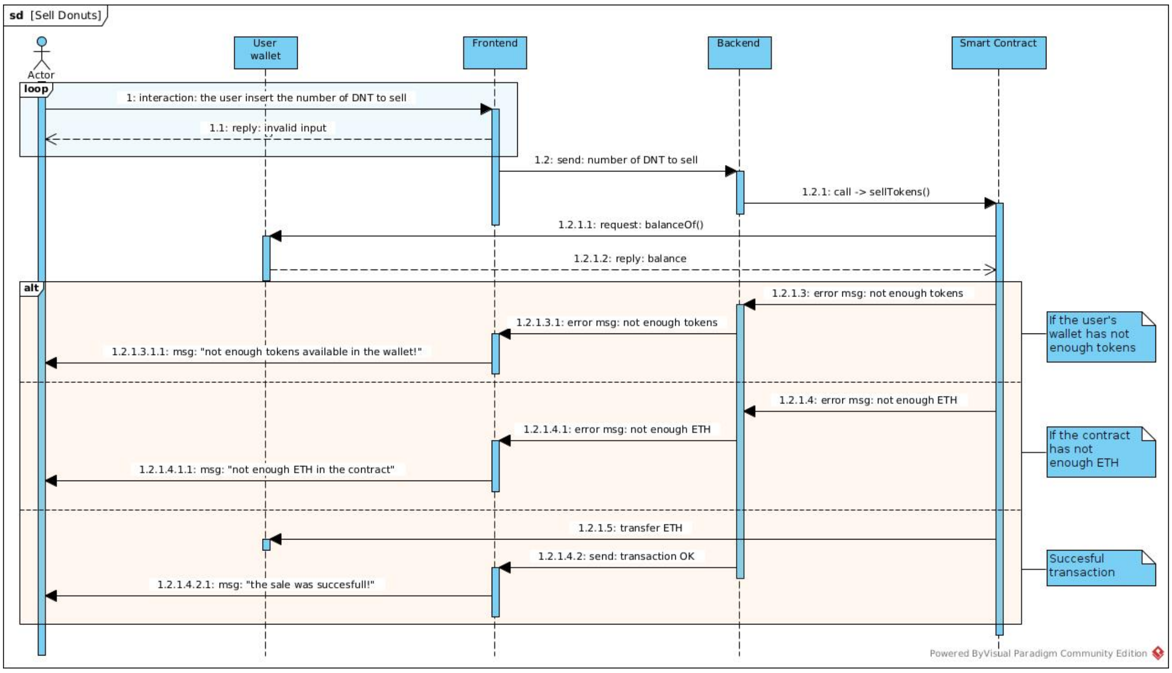
sd [Sell Donuts]
Actor
User wallet
Frontend
Backend
Smart Contract
loop
1: interaction: the user insert the number of DNT to sell
1.1: reply: invalid input
1.2: send: number of DNT to sell
1.2.1: call -> sellTokens()
1.2.1.1: request: balanceOf()
1.2.1.2: reply: balance
alt
1.2.1.3: error msg: not enough tokens
1.2.1.3.1: error msg: not enough tokens
1.2.1.3.1.1: msg: "not enough tokens available in the wallet!"
If the user's wallet has not enough tokens
1.2.1.4: error msg: not enough ETH
1.2.1.4.1: error msg: not enough ETH
1.2.1.4.1.1: msg: "not enough ETH in the contract"
If the contract has not enough ETH
1.2.1.5: transfer ETH
1.2.1.4.2: send: transaction OK
1.2.1.4.2.1: msg: "the sale was succesfull!"
Succesful transaction
Powered By Visual Paradigm Community Edition

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.

5 Implementation

This chapter provides an overview of the implementation of the Donuts Games dApp, starting with the smart contract responsible for managing Donut Tokens (DNT), including functions for token creation, transfers, buying, selling, and reward distribution. It then describes the backend integration with the Ethereum blockchain using `ethers.js`, enabling interactions with the smart contract. Lastly, the chapter covers the testing strategy used to verify the correct operation of the smart contract.

5.1 Smart Contract - Token.sol

The core of the dApp is the Donut Token smart contract, which manages token creation, transfers, and NFT minting. Written in Solidity, the contract defines the platform's token economy, with 1,000,000 DNT tokens initially assigned to the contract owner.

5.1.1 Token Creation and Initialization

The contract initializes the name, symbol, and total supply of the token, assigning the full supply to the deployer's address.

```
constructor() {
    balances[msg.sender] = totalSupply;
    owner = msg.sender;
}
```

5.1.2 Token Transfer

The `transfer` function allows users to send tokens to other Ethereum addresses, ensuring the sender has enough tokens before processing the transfer.

```
function transfer(address to, uint256 amount) external {
    require(balances[msg.sender] >= amount, "Not enough tokens");
    balances[msg.sender] -= amount;
    balances[to] += amount;
    emit Transfer(msg.sender, to, amount);
}
```

5.1.3 Buying Tokens

Users can purchase DNT by sending Ether to the contract. The amount of tokens received depends on the exchange rate, and tokens are transferred from the owner's balance to the buyer.

```
function buyTokens() external payable {
    uint256 amountToBuy = msg.value / rate;
    require(amountToBuy > 0, "You need to send some Ether");
    require(balances[owner] >= amountToBuy, "Not enough tokens available");
    balances[owner] -= amountToBuy;
    balances[msg.sender] += amountToBuy;
    emit Transfer(owner, msg.sender, amountToBuy);
    emit Buy(msg.sender, amountToBuy);
}
```

5.1.4 Selling Tokens

The `sellToken` function enables users to sell DNT in exchange for Ether, ensuring the contract has sufficient Ether to complete the transaction.

```
function sellToken(uint256 amountToSell) external {
    require(balances[msg.sender] >= amountToSell, "Not enough tokens");
    uint256 etherAmount = amountToSell * rate;
    require(address(this).balance >= etherAmount, "Not enough Ether in the contract"
    balances[msg.sender] -= amountToSell;
    balances[owner] += amountToSell;
    payable(msg.sender).transfer(etherAmount);
    emit Transfer(msg.sender, owner, amountToSell);
}
```

5.1.5 Rewarding Users

The contract owner can reward users with tokens using the `reward` function, which transfers tokens from the owner's balance.

```
function reward(uint256 amountToReward) external {
    require(balances[owner] >= amountToReward, "Not enough tokens available");
    balances[owner] -= amountToReward;
    balances[msg.sender] += amountToReward;
    emit Transfer(owner, msg.sender, amountToReward);
}
```

5.2 Testing the Smart Contract

Testing ensures that the Donut Token contract works as expected, covering token transfers, purchases, and sales. Tests are written using **Hardhat** and **Chai** to verify the functionality.

5.2.1 Deployment Tests

Tests confirm that the contract is deployed correctly, assigning the total token supply to the owner.

```
it("Should set the right owner", async function () {
    expect(await token.owner()).to.equal(owner.address);
});

it("Should assign the total supply of tokens to the owner", async function () {
    const ownerBalance = await token.balanceOf(owner.address);
    expect(ownerBalance).to.equal(await token.totalSupply());
});
```

5.2.2 Transaction Tests

Tests verify token transfers between users and ensure that transactions fail if there are insufficient funds.

```
it("Should transfer tokens between accounts", async function () {
    await token.transfer(addr1.address, 50);
    const addr1Balance = await token.balanceOf(addr1.address);
    expect(addr1Balance).to.equal(50);
});

it("Should fail if sender doesn’t have enough tokens", async function () {
    await expect(token.connect(addr1).transfer(addr2.address, ethers.parseEther("101
        .to.be.revertedWith("Not enough tokens");
});
```

5.2.3 Buy Tokens Test

This test verifies that users can purchase tokens by sending Ether, receiving the correct number of tokens based on the rate.

```
it("Should allow users to buy tokens by sending ether", async function () {
    const amountInEther = ethers.parseEther("1");
    await addr1.sendTransaction({ to: token.target, value: amountInEther });
    await token.connect(addr1).buyTokens({ value: amountInEther });
    const addr1Balance = await token.balanceOf(addr1.address);
    expect(addr1Balance).to.equal(amountInEther / (await token.rate()));
});
```

5.2.4 Withdraw Ether Test

Tests ensure only the contract owner can withdraw Ether and that their balance increases accordingly.

```
it("Should allow the owner to withdraw Ether", async function () {
    const amountInEther = ethers.parseEther("1");
    await addr1.sendTransaction({ to: token.target, value: amountInEther });
    await token.connect(owner).withdraw();
    const finalOwnerBalance = await ethers.provider.getBalance(owner.address);
    expect(finalOwnerBalance).to.be.above(initialOwnerBalance);
});

it("Should not allow non-owner to withdraw Ether", async function () {
    await expect(token.connect(addr1).withdraw()).to.be.revertedWith("Only the owner
});
```

5.2.5 Selling Tokens Test

Tests ensure that users can sell tokens for Ether and verify that transactions fail when users attempt to sell more than their balance.

```
it("Should allow a user to sell tokens and receive Ether", async function () {
    await token.connect(owner).transfer(addr1.address, 1000);
    const amountToSell = 10;
    await token.connect(addr1).sellToken(amountToSell);
    const addr1Balance = await token.balanceOf(addr1.address);
    expect(addr1Balance).to.equal(1000 - amountToSell);
});

it("Should fail if user tries to sell more tokens than they have", async function ()
    await expect(token.connect(addr1).sellToken(2000)).to.be.revertedWith("Not enoug
});
```

5.3 Deploying the Smart Contract

The deployment of the Donut Token contract is automated using a script written in **ethers.js**. The script retrieves the deployer's account, creates an instance of the contract, and deploys it to the Ethereum blockchain.

5.3.1 Deploy Script

The script defines an asynchronous `main` function to handle the contract deployment process.

```
async function main() {
    const [deployer] = await ethers.getSigners();
    const Token = await ethers.getContractFactory("Token");
    const token = await Token.deploy();
    console.log("Token deployed to:", token.target);
}
```

The script concludes by handling potential errors and ensuring the deployment completes successfully.

```
main()
    .then(() => process.exit(0))
    .catch(error => {
        console.error(error);
        process.exit(1);
    });
```

5.4 Backend.js Module

The `backend.js` module facilitates interactions between the dApp's backend and the Ethereum smart contract. Using **ethers.js**, it manages token transactions, rewards, and NFTs.

5.4.1 Configuring the Provider and Wallet

The module initializes a connection to the Ethereum network using a provider and configures the wallet using private keys.

```
const provider = ethers.getDefaultProvider('http://127.0.0.1:8545');
const privateKey = process.env.PRIVATE_KEY_1;
const wallet = new ethers.Wallet(privateKey, provider);
```

5.4.2 Contract Connection

An instance of the contract is created using its ABI and address, allowing the backend to interact with the deployed contract.

```
const contractAddress = process.env.CONTRACT_ADDRESS;
const contractABI = [...];
const contract = new ethers.Contract(contractAddress, contractABI, wallet);
```

5.4.3 Buying Tokens

The buyToken function allows users to buy tokens by sending Ether to the contract.

```
async function buyToken(amount) {
    const valueToSend = ethers.parseEther(amount);
    const tx = await contract.buyTokens({ value: valueToSend });
    const receipt = await tx.wait();
    await updateBalance();
}
```

5.4.4 Selling Tokens

The sell function enables users to sell tokens back to the contract and receive Ether.

```
async function sell(amountToSell) {
    const tx = await contract.sellToken(amountToSell);
    const receipt = await tx.wait();
    await updateBalance();
}
```

5.4.5 Rewarding Users

The reward function allows the backend to distribute rewards to users, transferring tokens from the owner's balance.

```
async function reward(amountToReward) {
    const tx = await contract.reward(amountToReward);
    const receipt = await tx.wait();
    await updateBalance();
}
```

6 Known Issues and Limitations

In the ever-changing digital landscape, we are witnessing a technology that has captured the attention of experts, media, and users worldwide: deepfakes...

7 Conclusions and Future Remarks

In the ever-changing digital landscape, we are witnessing a technology that has captured the attention of experts, media, and users worldwide: deepfakes...

References