Introduction:

Welcome to the report documenting the development of a decentralized marketplace on the Ethereum testnet. The primary objective of this project was to create a platform where users can list and purchase items using Ethereum smart contracts for transaction management.

My marketplace is about Handmade crafts and Artisanal products.

Throughout this report, you will find detailed insights into the setup process, smart contract development, frontend implementation, integration, testing, challenges faced, GitHub repository link, and the live marketplace website link.

Accessing Website:

To access the decentralized marketplace website, users can simply navigate to the following URL: https://mota0005.github.io/eth-marketplace/

Once on the website, users will be able to interact with the marketplace interface, where they can list items for sale and purchase items using Ethereum transactions.

Smart Contract Code:

The following section presents the Solidity smart contract code used in the development of the decentralized marketplace. This smart contract is instrumental in managing the listing and purchasing of items on the marketplace.

Below is the code,

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract Marketplace {

  // Structure to represent an item for sale

  struct Item {

    uint256 id;            // Unique identifier for the item

    address payable seller;    // Address of the seller

    string title;          // Title of the item

    string description;       // Description of the item

    uint256 price;          // Price of the item

    bool isSold;           // Flag indicating whether the item has been sold

  }


  // Mapping to store items by their ID
```

```
mapping(uint256 => Item) public items;
uint256 public itemCount;      // Counter to keep track of the number of items listed

// Event emitted when a new item is listed
event ItemListed(
   uint256 indexed itemId,
   address indexed seller,
   string title,
   uint256 price
);

// Event emitted when an item is purchased
event ItemPurchased(
   uint256 indexed itemId,
   address indexed buyer,
   uint256 price
);

// Function to list a new item for sale
function listNewItem(string memory _title, string memory _description, uint256 _price) public {
   itemCount++;    // Increment item count
   // Create a new Item struct and add it to the mapping
   items[itemCount] = Item(itemCount, payable(msg.sender), _title, _description, _price, false);

   // Emit an event to notify listeners about the new listing
   emit ItemListed(itemCount, msg.sender, _title, _price);
}

// Function to purchase an item
```

```
    function purchaseItem(uint256 _itemId) public payable {
        Item storage item = items[_itemId];
        // Ensure the buyer is not the seller
        require(msg.sender != item.seller, "Seller cannot buy their own item");
        // Ensure the item is not already sold
        require(!item.isSold, "Item already sold");
        // Ensure the buyer sends the correct amount of Ether
        require(msg.value == item.price, "Please submit the asking price in order to complete the
purchase");

        // Transfer the Ether to the seller
        item.seller.transfer(msg.value);
        // Mark the item as sold
        item.isSold = true;

        // Emit an event to notify listeners about the purchase
        emit ItemPurchased(_itemId, msg.sender, item.price);
    }

    // Function to retrieve all listed items
    function getAllItems() public view returns (Item[] memory) {
        Item[] memory itemList = new Item[](itemCount);
        // Loop through all items and add them to the array
        for (uint256 i = 1; i <= itemCount; i++) {
            itemList[i - 1] = items[i];
        }
        return itemList;
    }
}
```

Key Functionalities:

- Item Structure: Defines the structure of an item for sale, including its ID, seller address, title, description, price, and sale status.
- Mapping: Stores items by their ID for easy retrieval and management.
- Events: Emits events when a new item is listed or purchased to notify listeners about the actions.
- List New Item Function: Allows sellers to list new items for sale, incrementing the item count and emitting an event upon listing.
- Purchase Item Function: Enables buyers to purchase items by transferring the correct amount of Ether to the seller and marking the item as sold.
- Get All Items Function: Retrieves all listed items for display on the marketplace interface.

Challenges and Issues Faced:

Difficulty Obtaining Test Ether:

Initially, there was difficulty obtaining test Ether from the faucet provided by the Ethereum testnet.

Resolution: After persistent attempts and familiarization with the faucet process, test Ether was successfully obtained from https://www.ethereum-ecosystem.com/faucets/ethereum-sepolia.

Choosing Development Environment:

Difficulty arose in choosing between different development environments such as Remix IDE, Truffle, and Hardhat for smart contract development.

Resolution: After experimenting with each environment, I chose Remix IDE for its user-friendly interface and the ability to compile and deploy contracts directly from the web browser.

Handling Item Storage:

After implementing functionalities for listing and purchasing items in the smart contract, I realized there was no mechanism in place to store the items.

Resolution: Instead of integrating with a database, I added a function to the smart contract to retrieve all listed items, which were then displayed directly on the frontend interface. This simplified the architecture and eliminated the need for external storage.

Frontend Development Challenges:

I encountered challenges during the development of the frontend interface, particularly in implementing notifications for user interactions.

Resolution: Toast notifications were integrated into the frontend code to provide users with informative messages and feedback as needed, enhancing the overall user experience.

These challenges were addressed through thorough problem-solving and adaptation throughout the development process along with a few other issues.

GitHub Repository Link:

The GitHub repository for the decentralized marketplace project can be accessed at the following link: https://github.com/mota0005/eth-marketplace

This repository contains all project files, including:

- Smart Contract: Solidity smart contract code used for managing the decentralized marketplace transactions.
- Frontend Code: HTML, CSS, and JavaScript files comprising the user interface of the decentralized marketplace.
- Documentation: Detailed documentation providing insights into project design, implementation, and usage instructions.

The repository is publicly accessible.

Note:

Additionally, I utilized ChatGPT for assistance and guidance during the development process.