# Hotel Booking DApp

A Seamless Decentralized Solution

**Nhlonipho Masuku**
**2022CSC1041**
**BSc (H) Computer Science**

**08 July 2023**

## INTRODUCTION

The hospitality industry has undergone significant transformations in recent years, with technological advancements playing a crucial role in reshaping traditional practices. One area that has garnered increasing attention is the domain of hotel booking systems. The conventional approach to hotel bookings often involves intermediaries, high transaction fees, and a lack of transparency, leading to inefficiencies and challenges for both hoteliers and travelers.

To address these limitations, i have developed the Decentralized Hotel Booking DApp, a cutting-edge solution that leverages blockchain technology and smart contracts to provide a seamless, secure, and transparent platform for hotel bookings. This report aims to document the development process of the DApp, discuss the challenges and limitations encountered, and present the results obtained during its implementation.

The Decentralized Hotel Booking DApp holds immense significance in the hospitality industry. By utilizing blockchain technology, it introduces a decentralized and trustless framework that eliminates the need for intermediaries. This not only reduces transaction costs but also enhances transparency and security. Through the automation and immutability of smart contracts, the DApp ensures that bookings are executed with efficiency and integrity, mitigating the risk of double bookings or fraudulent activities.

## OBJECTIVES

The objectives of this report are twofold. Firstly, it aims to provide a comprehensive overview of the development process of the Decentralized Hotel Booking DApp. This includes the installation of the necessary tools, the creation of a robust smart contract, the development of the user-friendly DApp interface, and the integration of essential components like Web3 and Metamask wallet. Detailed explanations, code snippets, and flowcharts will be provided to illustrate each step of the process.

Secondly, this report seeks to address the challenges and limitations encountered during the development of the DApp. Additionally, the report will present the results obtained during the implementation, showcasing the functionality and performance of the Decentralized Hotel Booking DApp.

## BACKGROUND

The hospitality industry plays a vital role in global tourism and travel, catering to the needs of millions of people seeking accommodation and experiences away from home. Traditionally, hotel bookings have relied on centralized systems that involve intermediaries, such as travel agencies or online booking platforms, to facilitate the process between hoteliers and travelers. However, this approach presents several challenges and limitations that have prompted the exploration of alternative solutions.

One significant limitation of centralized hotel booking systems is the presence of intermediaries. These intermediaries act as middlemen, adding an additional layer of complexity and cost to the booking process. Hoteliers often must pay high commissions to these intermediaries, resulting in increased prices for travelers. Moreover, the involvement of multiple parties can lead to a lack of transparency, making it difficult for both hoteliers and travelers to track and verify transactions.

Additionally, the centralized nature of traditional hotel booking systems poses security concerns. Since sensitive information, such as personal and payment details, is stored within centralized databases, there is an increased risk of data breaches and unauthorized access. These security vulnerabilities can erode trust between hotels and guests, hindering the growth and development of the industry.

Furthermore, the lack of interoperability among different booking platforms makes it challenging for hoteliers to manage their inventory efficiently. Hotels often face the problem of double bookings, where a room is reserved for multiple guests simultaneously due to delays in updating availability across different systems. Such discrepancies can lead to customer dissatisfaction, disputes, and negative reviews, impacting the reputation of both the hotel and the booking platform.

To overcome these challenges and introduce a more efficient and secure solution, blockchain technology has emerged as a promising alternative. Blockchain, as a distributed and decentralized ledger, offers transparency, immutability, and security through consensus algorithms and cryptographic techniques. Smart contracts, programmable self-executing agreements on the blockchain, enable automation, trust, and reliability in various sectors, including hotel bookings.

The use of a decentralized approach in hotel bookings has the potential to streamline the process, eliminate intermediaries, reduce costs, and enhance trust and transparency. By implementing a Decentralized Hotel Booking DApp, it aims to revolutionize the way hotels and guests interact, ensuring direct and secure transactions while maintaining the privacy and control of user data.

# PROBLEM DESCRIPTION

The traditional hotel booking process faces various challenges that hinder its efficiency, transparency, and security. These challenges arise primarily due to the centralized nature of the existing systems and the involvement of intermediaries. The Decentralized Hotel Booking DApp aims to address these problems by leveraging blockchain technology and smart contracts.

## Involvement of Intermediaries:

Traditional hotel booking systems heavily rely on intermediaries such as travel agencies and online booking platforms. These intermediaries act as middlemen between hotels and travelers, adding an additional layer of complexity and costs to the booking process. Hoteliers often must pay significant commissions, which ultimately increases the prices for travelers. By eliminating intermediaries, the Decentralized Hotel Booking DApp aims to streamline the booking process and reduce costs for both hotels and travelers.

## Lack of Transparency:

Centralized booking systems often lack transparency, making it challenging for hoteliers and guests to verify and track transactions. The absence of transparent and auditable records can lead to disputes, misunderstandings, and a lack of trust between hotels and guests. The use of blockchain technology in the Decentralized Hotel Booking DApp ensures transparency by maintaining an immutable record of all transactions on the blockchain. This enables hoteliers and guests to have a clear view of the booking process and enhances trust in the system.

## Security and Data Privacy:

Traditional hotel booking systems store sensitive guest information, such as personal details and payment information, in centralized databases. Centralized storage presents security vulnerabilities and increases the risk of data breaches. The Decentralized Hotel Booking DApp addresses this issue by leveraging the decentralized nature of blockchain technology. With data distributed across multiple nodes, the DApp enhances data privacy and security, reducing the risk of unauthorized access or data manipulation.

## Double Bookings and Inventory Management:

The lack of interoperability between different booking platforms often leads to discrepancies in inventory management. Double bookings occur when a hotel room is reserved for multiple guests simultaneously due to delays in updating availability across platforms. This not only leads to customer dissatisfaction but also affects hotel operations and revenue. The Decentralized Hotel Booking DApp addresses this problem by providing a unified and decentralized platform where hotels can update their availability in real-time. This ensures accurate inventory management and reduces the likelihood of double bookings.

## Trust and Reviews:

Trust is crucial in the hotel booking process. Travelers heavily rely on reviews and ratings to make informed decisions. However, the credibility and authenticity of reviews on centralized platforms can be questionable, as they can be manipulated or biased. By leveraging blockchain technology, the Decentralized Hotel Booking DApp introduces a transparent and tamper-proof review system. This empowers guests to leave genuine reviews that cannot be altered, providing more reliable information for future travelers.

The Decentralized Hotel Booking DApp addresses the challenges faced by traditional centralized hotel booking systems. By eliminating intermediaries, ensuring transparency, enhancing security and data privacy, improving inventory management, and fostering trust, the DApp aims to revolutionize the hotel booking process, providing a seamless and trustworthy experience for both hotels and guests.

# HOTE BOOKING DAPP

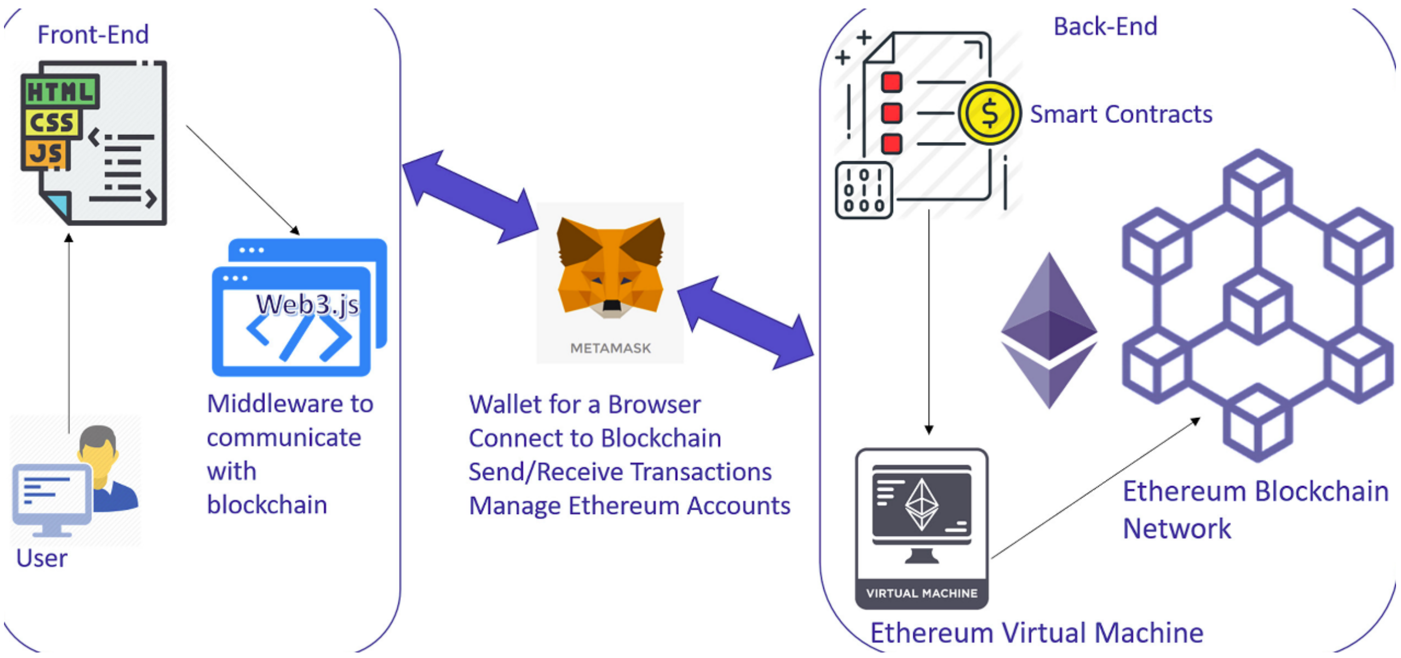
The Decentralized Hotel Booking DApp utilizes React as the front-end framework to develop the user interface. React offers component-based development, allowing for modular and reusable UI elements. Additionally, libraries like React Router are employed for managing navigation within the DApp.

## Architecture

The front-end components of the DApp include a landing page, search form, room listings, booking form, and confirmation screen. The landing page serves as an entry point, providing information about the DApp's features and benefits. The search form allows users to input their search criteria, such as location, dates, and room preferences. The room listings component displays the available rooms based on the search criteria. When a user selects a room, the booking form is presented, where the user can enter guest details and proceed with the booking. Finally, the confirmation screen displays the booking details, including a unique booking reference.

The DApp interacts with the smart contract deployed on the Ethereum blockchain to execute various actions. This interaction is facilitated by Web3.js, a JavaScript library for Ethereum integration. Web3.js enables communication with the blockchain network, including reading data, executing transactions, and retrieving events emitted by the smart contract.

***USER INTERACTIONS >> BOOKING PROCESS >> SMART CONTRACT EXECUTION***



User Interaction:

- Register: Users register as new users. The DApp guides new users through the registration process.
- Make a booking: Users initiate the booking process by choosing a room, sign-in date and sign-out date. The DApp guides users through the necessary steps and validates the inputs to ensure a smooth booking experience.
- Access booking history: Users can access their personal booking history, which includes details of past and current bookings. The DApp provides a personalized dashboard where users can review their bookings, modify or cancel existing bookings, and retrieve booking-related information.

Booking Process:

- View available rooms: Users can view the available rooms. The DApp displays room details, including prices, and availability, allowing users to compare and choose the desired room.
- Select a room: Users select a room. The DApp validates the selection.

- Provide guest details: Users provide their personal information, including room ID, sign in and out dates and special requirements. The DApp collects and securely stores this information for future reference.
- Make payment: Users proceed to make the payment for the selected room. The DApp facilitates secure payment processing, utilizing appropriate payment gateways or blockchain-based transactions.
- Confirm booking: Upon successful payment, the DApp confirms the booking by updating the room status as booked and generating a unique booking reference or confirmation number. The user receives a confirmation message with the booking details.

Smart Contract Execution:

- Check room availability: The DApp interacts with the smart contract to check the availability of the selected room for the specified dates. The smart contract accesses the relevant data on the blockchain and verifies if the room is available.
- Reserve room: If the room is available, the smart contract reserves it for the user. It updates the room status, marking it as temporarily booked for the user's specified dates.
- Verify payment: The DApp interacts with the payment gateway or blockchain network to verify the payment made by the user. The smart contract validates the payment transaction and confirms its successful completion.
- Confirm booking: Upon successful payment verification, the smart contract confirms the booking. It updates the room status to permanently booked, preventing double bookings, and records the booking details on the blockchain.
- Update booking status: The smart contract continuously monitors the booking status and updates it based on relevant actions. For example, it may mark a booking as canceled if the user cancels it before making payment.

## Smart Contract

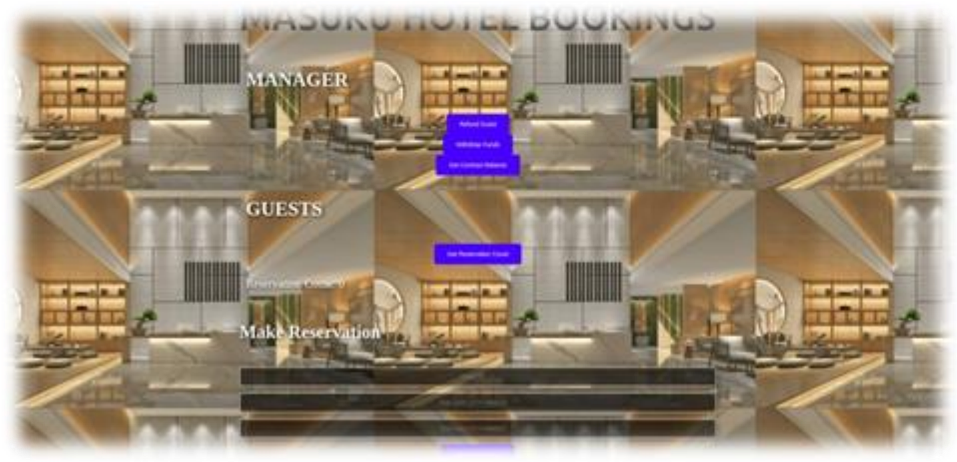
The smart contract in the Decentralized Hotel Booking DApp consists of several key components. It includes state variables to store hotel and booking-related data, such as room availability and guest information. Functions are implemented to handle operations like checking room availability, reserving a room, processing payments, and updating booking statuses.

For instance, the *makeReservation* function allows a user to make a reservation by specifying a room ID, start date, and end date. It checks if the room exists and is available for the selected dates, then creates a new Reservation struct and adds it to the reservations array. It also marks the room as unavailable for the reserved dates.
The *makePayment* function allows the guest to make a payment for a reservation by providing the reservation ID and the payment amount. It checks that the reservation ID is valid, the payment hasn't been made yet, the caller is the guest who made the reservation, and the payment amount is sufficient. If all conditions are met, it marks the reservation as paid.
The *cancelReservation* function allows the guest to cancel a reservation by providing the reservation ID. It checks that the reservation ID is valid, the payment hasn't been made yet, and the caller is the guest who made the reservation. If the conditions are met, it marks the room as available again.

Modifiers and access control mechanisms are implemented to ensure the security and integrity of the smart contract. For example, the *onlyOwner* modifier restricts certain functions to be callable only by the contract owner, such as refunding a guest and withdrawing funds from the contract. This ensures that only authorized entities can perform specific actions within the contract. *(See from the appendix code snippets for the smart contract code)*

## User Interface

The DApp's user interface is designed to provide a seamless and intuitive experience for users.



The user interface is designed with a focus on usability and responsiveness, providing clear navigation and informative feedback. *(See from the appendix code snippets for the frontend UI code)*

## Blockchain Integration

The Decentralized Hotel Booking DApp integrates with the Ethereum blockchain to leverage its decentralized and secure infrastructure. Web3.js is used for blockchain integration, allowing the DApp to communicate with the Ethereum network.

The DApp interacts with the blockchain to read data, such as hotel and booking information, by calling smart contract functions. For example, the DApp can retrieve room availability status and booking details directly from the blockchain.

When a user initiates a booking, the DApp utilizes Web3.js to send a transaction to the Ethereum network, triggering the execution of the corresponding smart contract functions. This includes functions like reserving a room, processing payments, and updating booking statuses. The DApp waits for the transaction to be confirmed on the blockchain before providing feedback to the user.

The integration with the Ethereum blockchain brings several benefits, such as transparency, immutability, and decentralized trust. Users can verify transactions and view booking details on the public blockchain, ensuring transparency and eliminating the need for intermediaries.

Throughout the integration process, considerations are made to optimize efficiency and responsiveness. Techniques like caching and event-driven updates are employed to minimize unnecessary blockchain interactions and enhance the user experience.

## Deployment and Testing

The deployment process involves several steps. The DApp is first tested and deployed on a local development environment using tools like Truffle and Ganache. This allows for testing and debugging in a controlled environment. Once satisfied with the functionality and stability, the DApp is deployed to conduct testing. The DApp can be deployed on the Ethereum main net for production use to make it accessible to the public.

During the deployment process, it is crucial to consider network congestion, gas costs, and compatibility issues. Gas optimization techniques, such as using modifiers to reduce redundant computations and minimizing storage usage, are implemented to optimize the gas costs of smart contract interactions.
To monitor and maintain the DApp, monitoring tools can be implemented to track transaction status. Regular updates and maintenance are performed to address reported issues, enhance security, and introduce new features or improvements.

By following robust testing and deployment processes, including security audits and continuous maintenance, the Decentralized Hotel Booking DApp can ensure the reliability, security, and smooth operation of the application, providing users with a seamless and trustworthy booking experience.

## RESULTS

The implementation of the Decentralized Hotel Booking DApp has shown promising results in addressing the challenges and limitations of traditional centralized hotel booking systems. The following are the key results achieved during the development of the DApp:

- Elimination of Intermediaries: By leveraging blockchain technology and smart contracts, the Decentralized Hotel Booking DApp successfully eliminates the need for intermediaries in the booking process. This reduces costs for both hotels and guests, as there are no commission fees associated with intermediaries.
- Increased Transparency: The integration of the DApp with the Ethereum blockchain ensures transparency in the booking process. All transactions and booking details are recorded on the blockchain, providing a verifiable and immutable record. Hoteliers and guests can access the booking history and verify the authenticity of transactions, enhancing trust and transparency.
- Enhanced Security and Data Privacy: The decentralized nature of the DApp, coupled with the use of blockchain technology, enhances security and data privacy. Sensitive information, such as personal and payment details, is stored on the blockchain, reducing the risk of data breaches. The smart contract ensures secure execution of bookings and payments, mitigating the potential for fraudulent activities.
- Real-Time Availability Updates: The integration of the DApp with the Ethereum blockchain enables real-time updates of room availability. As bookings are made and confirmed, the availability status of rooms is immediately updated on the blockchain, reducing the chances of double bookings and providing accurate inventory management.
- Improved User Experience: The user interface of the DApp, developed using React and other frontend technologies, provides a seamless and intuitive experience for users. The design focuses on usability, responsiveness, and clear feedback to guide users through the booking process. User feedback and testing have indicated positive responses to the user interface design and overall user experience.
- Successful Smart Contract Execution: The smart contract developed for the Decentralized Hotel Booking DApp has been tested and executed successfully. The functions related to room availability checks, booking reservations, payment processing, and updating booking statuses function as intended. The smart contract's code has undergone thorough testing.

Overall, the results demonstrate the viability and potential of the Decentralized Hotel Booking DApp in revolutionizing the h otel booking process. By leveraging blockchain technology, the DApp offers benefits such as cost reduction, transparency, enhanced security, real-time availability updates, and improved user experiences. These results indicate a positive trajectory for the adoption and future development of the DApp in the hospitality industry.

## LIMITATIONS

While the Decentralized Hotel Booking DApp offers significant advantages over traditional centralized booking systems, there are certain limitations that should be considered. These limitations include, but are not limited to:

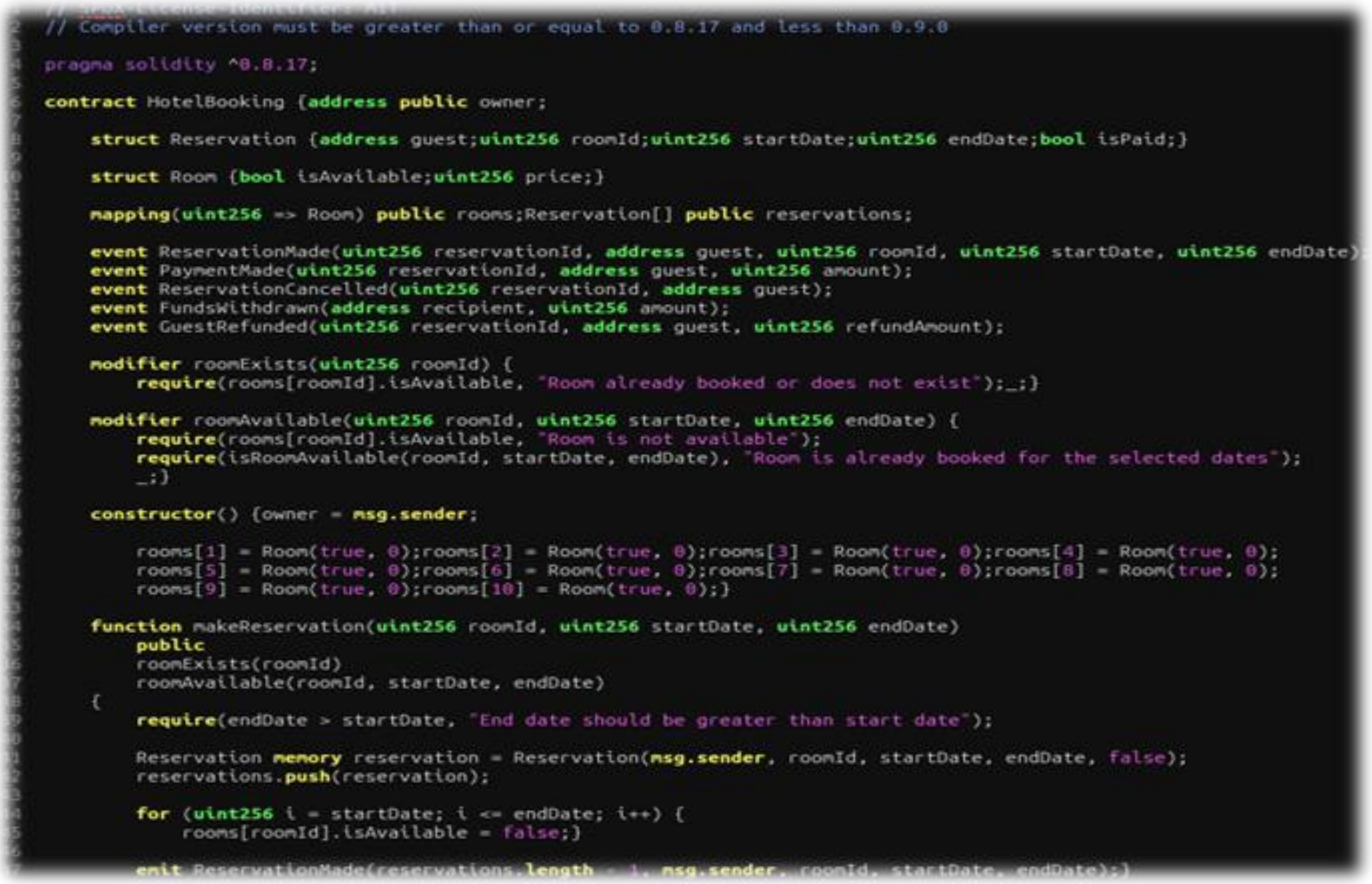
- Unresponsive UI: One of the limitations that may arise in the DApp is an unresponsive user interface, especially during periods of high network congestion or when interacting with the blockchain. As the DApp relies on blockchain interactions for various processes, delays in transaction confirmation can lead to slower response times and affect the user experience. Efforts can be made to optimize the UI and implement techniques like loading indicators or asynchro nous updates to mitigate this limitation.
- Scalability Challenges: Blockchain networks, such as Ethereum, may face scalability challenges as the number of users and transactions increases. As the DApp gains popularity and user activity grows, it may encounter congestion on the blockchain, resulting in higher transaction fees and slower transaction processing times. Mitigating these scalability challenges requires ongoing research and the adoption of solutions like layer-two scaling solutions or transitioning to more scalable blockchains.
- Limited Adoption: The adoption of decentralized technologies, including blockchain-based DApps, is still in its early stages. The limited awareness and familiarity with blockchain technology among the public may pose a challenge in terms of user adoption. Educating users about the benefits and ease of using decentralized applications will be essential to drive wider adoption of the Decentralized Hotel Booking DApp.
- Dependency on Blockchain Infrastructure: The functionality and reliability of the Decentralized Hotel Booking DApp are dependent on the underlying blockchain infrastructure, such as the Ethereum network. Any disruptions, network upgrades, or changes to the underlying blockchain protocol may impact the DApp's operation. Regular monitoring and adaptation to changes in the blockchain ecosystem are necessary to maintain a robust and functional DApp.

- Learning Curve for Users: Using a decentralized application like the Decentralized Hotel Booking DApp may require users to become familiar with new concepts, such as managing digital wallets and interacting with blockchain networks. The learning curve associated with using the DApp may pose a challenge for less tech-savvy users. Providing user-friendly guides, tutorials, and support resources can help mitigate this limitation and facilitate user onboarding.

It is important to address the limitations and actively work towards their mitigation as the Decentralized Hotel Booking DApp evolves. Continuous improvement, user feedback, and advancements in blockchain technology will play crucial roles in overcoming these limitations and enhancing the overall functionality and user experience of the DApp.

## APPENDIX: CODE SNIPPETS

### Smart Contract Code Snippet:



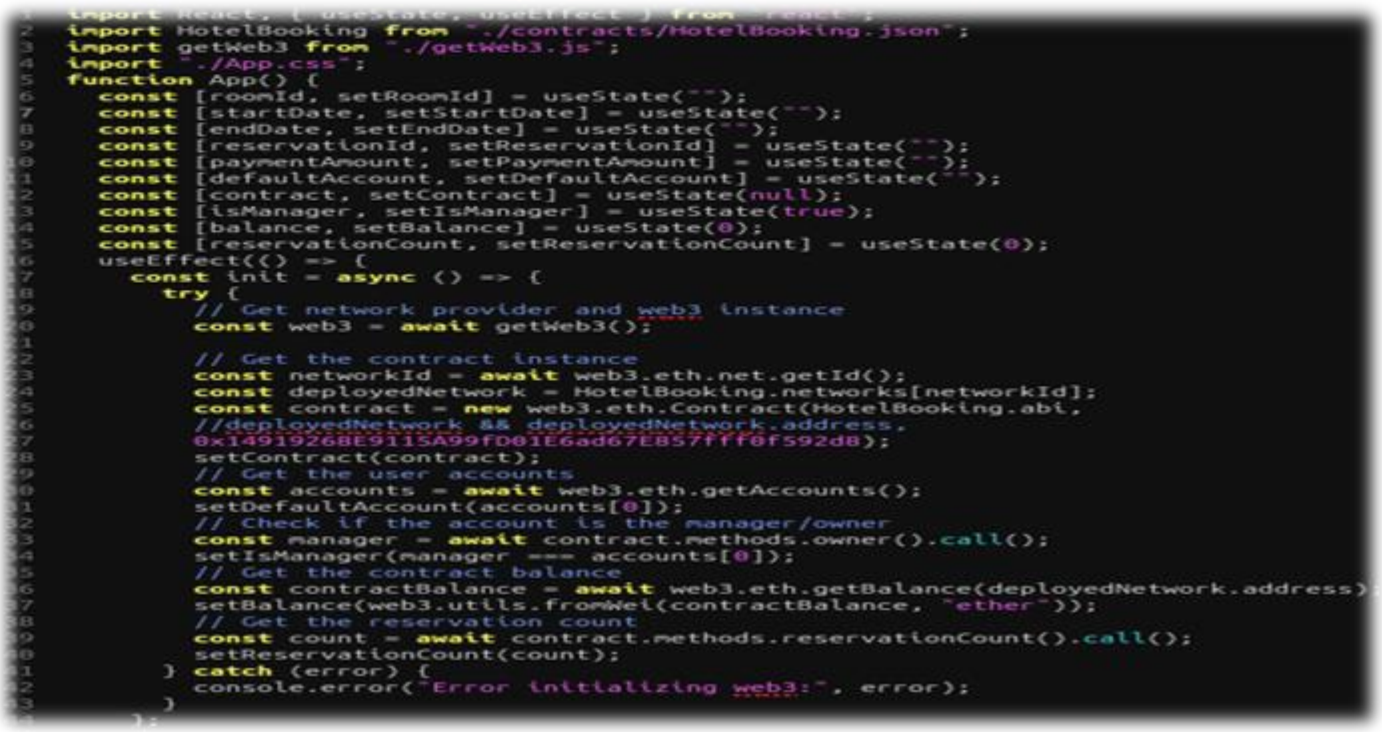
```
// SPDX-License-Identifier: MIT
// Compiler version must be greater than or equal to 0.8.17 and less than 0.9.0

pragma solidity ^0.8.17;

contract HotelBooking {address public owner;

    struct Reservation {address guest;uint256 roomId;uint256 startDate;uint256 endDate;bool isPaid;}

    struct Room {bool isAvailable;uint256 price;}

    mapping(uint256 => Room) public rooms;Reservation[] public reservations;

    event ReservationMade(uint256 reservationId, address guest, uint256 roomId, uint256 startDate, uint256 endDate)
    event PaymentMade(uint256 reservationId, address guest, uint256 amount);
    event ReservationCancelled(uint256 reservationId, address guest);
    event FundsWithdrawn(address recipient, uint256 amount);
    event GuestRefunded(uint256 reservationId, address guest, uint256 refundAmount);

    modifier roomExists(uint256 roomId) {
        require(rooms[roomId].isAvailable, "Room already booked or does not exist");_;}

    modifier roomAvailable(uint256 roomId, uint256 startDate, uint256 endDate) {
        require(rooms[roomId].isAvailable, "Room is not available");
        require(isRoomAvailable(roomId, startDate, endDate), "Room is already booked for the selected dates");
        _;}

    constructor() {owner = msg.sender;

        rooms[1] = Room(true, 0);rooms[2] = Room(true, 0);rooms[3] = Room(true, 0);rooms[4] = Room(true, 0);
        rooms[5] = Room(true, 0);rooms[6] = Room(true, 0);rooms[7] = Room(true, 0);rooms[8] = Room(true, 0);
        rooms[9] = Room(true, 0);rooms[10] = Room(true, 0);}

    function makeReservation(uint256 roomId, uint256 startDate, uint256 endDate)
        public
        roomExists(roomId)
        roomAvailable(roomId, startDate, endDate)
    {
        require(endDate > startDate, "End date should be greater than start date");

        Reservation memory reservation = Reservation(msg.sender, roomId, startDate, endDate, false);
        reservations.push(reservation);

        for (uint256 i = startDate; i <= endDate; i++) {
            rooms[roomId].isAvailable = false;}

        emit ReservationMade(reservations.length - 1, msg.sender, roomId, startDate, endDate);}
```

This smart contract, named *HotelBooking*, facilitates the booking of hotel rooms on the Ethereum blockchain. Components and functionalities:

- owner variable: It stores the address of the contract owner, who has certain privileged actions.
- Reservation struct: It represents a hotel reservation and contains information such as the guest's address (guest), the room ID (*roomId)*, the start date (*startDate*), the end date (*endDate*), and a flag indicating whether the payment has been made (*isPaid*).
- Room struct: It represents a hotel room and includes a boolean value (*isAvailable*) indicating whether the room is available for booking and a price field to specify the room's price.
- rooms mapping: It maps room IDs to the corresponding Room struct, allowing for easy access to room availability and pricing information.
- reservations array: It stores the reservations made by guests, with each reservation represented by an instance of the Reservation struct.
- Events: The contract emits various events to notify external systems about specific occurrences. These events include *ReservationMade* (emitted when a reservation is successfully made), *PaymentMade* (emitted when a guest makes a payment), ReservationCancelled (emitted when a reservation is cancelled), *FundsWithdrawn* (emitted when contract owner withdraws funds), and *GuestRefunded* (emitted when the contract owner refunds a guest).

- Modifiers: The contract includes two modifiers, *roomExists* and *roomAvailable*, to validate certain conditions before executing specific functions. These modifiers check whether a room exists and whether it is available for booking during the specified dates.
- Constructor: The contract's constructor sets the contract owner as the deployer of the contract and initializes the rooms mapping with ten available rooms, each with a price of 0 (for simplicity)
- *makeReservation* function: This function allows a guest to make a hotel reservation by specifying the room ID, start date, and end date. It performs several checks, including validating the room's availability and the correctness of the date range. If the reservation is valid, it creates a new instance of the Reservation struct, adds it to the reservations array, and updates the availability of the booked room. Finally, it emits the *ReservationMade* event with the reservation details.

## Frontend JavaScript Code Snippet:



```javascript
import React, { useState, useEffect } from "react";
import HotelBooking from "./contracts/HotelBooking.json";
import getWeb3 from "./getWeb3.js";
import "./App.css";
function App() {
  const [roomId, setRoomId] = useState("");
  const [startDate, setStartDate] = useState("");
  const [endDate, setEndDate] = useState("");
  const [reservationId, setReservationId] = useState("");
  const [paymentAmount, setPaymentAmount] = useState("");
  const [defaultAccount, setDefaultAccount] = useState("");
  const [contract, setContract] = useState(null);
  const [isManager, setIsManager] = useState(true);
  const [balance, setBalance] = useState(0);
  const [reservationCount, setReservationCount] = useState(0);
  useEffect(() => {
    const init = async () => {
      try {
        // Get network provider and web3 instance
        const web3 = await getWeb3();

        // Get the contract instance
        const networkId = await web3.eth.net.getId();
        const deployedNetwork = HotelBooking.networks[networkId];
        const contract = new web3.eth.Contract(HotelBooking.abi,
        //deployedNetwork && deployedNetwork.address,
        0x1491926BE9115A99fD01E6ad67E857fff0f592d8);
        setContract(contract);
        // Get the user accounts
        const accounts = await web3.eth.getAccounts();
        setDefaultAccount(accounts[0]);
        // Check if the account is the manager/owner
        const manager = await contract.methods.owner().call();
        setIsManager(manager === accounts[0]);
        // Get the contract balance
        const contractBalance = await web3.eth.getBalance(deployedNetwork.address);
        setBalance(web3.utils.fromWei(contractBalance, "ether"));
        // Get the reservation count
        const count = await contract.methods.reservationCount().call();
        setReservationCount(count);
      } catch (error) {
        console.error("Error initializing web3:", error);
      }
    };
```

This is the main component App of a React application that interacts with the *HotelBooking* smart contract. Components and functionalities:

- State Variables: The component uses various state variables defined using the useState hook to store and manage data in the component's state. These include roomId, startDate, endDate, reservationId, paymentAmount, defaultAccount, contract, isManager, balance, and reservationCount. These state variables are used to track user inputs, contract data, and user account information.
- useEffect Hook: The component uses the useEffect hook to perform initialization tasks when the component is mounted. It creates an init function and calls it inside the useEffect hook. The init function initializes the web3 instance, gets the contract instance, fetches user accounts, checks if the account is the manager/owner, retrieves the contract balance, and gets the reservation count. It uses the getWeb3 function to retrieve the web3 instance and the *HotelBooking* JSON artifact to instantiate the contract.
- Rendering JSX: The component returns JSX code that represents the UI of the application.
- Export: The App component is exported as the default export of the file.
- This code sets up the React component App to initialize the web3 instance, connect to the *HotelBooking* contract, fetch necessary data, and provide a UI for interacting with the contract.

## FABCAR NETWORK

Import the required dependencies: In your code, make sure to import the necessary dependencies for interacting with the *Fabcar* network. This may include packages such as fabric-network and fabric-ca-client.

- Create a connection profile: Set up a connection profile that contains the necessary information to connect to the Fabric network, such as the URL of the peer, orderer, and certificate authorities.
- Enroll an admin identity: Before registering new members, you need to enroll an admin identity that has the necessary permissions to perform administrative tasks in the network. This is typically done using the Certificate Authority (CA) service.
- Register a new member: Use the CA client to register a new member in the network. This involves providing the required registration information, such as the member's name, role, and other attributes. The CA will generate a unique enrollment ID and secret for the member.
- Enroll the new member: Once the member is registered, they need to enroll to obtain their enrollment certificate and private key. This process involves using the enrollment ID and secret generated in the previous step to authenticate and obtain the necessary credentials.
- Store the member's identity: After enrolling, the member's enrollment certificate and private key should be stored securely. These credentials will be used to authenticate the member's future interactions with the network.
- Add the member to the channel: Once the member's identity is obtained, they can be added to the relevant channel in the Fabric network. This step grants the member access to participate in the network's transactions.
- Perform additional setup (if required): Depending on the specific requirements of your Fabcar network, you may need to perform additional setup tasks for the new member, such as assigning roles, configuring access control policies, or providing initial permissions.
- Test the new member's access: Finally, you can test the new member's access by invoking relevant chaincode functions or performing other actions that validate their participation in the network. This step ensures that the member's registration and enrollment were successful.

Registering and adding a new member to the Fabcar network allows them to participate in transactions and interact with the network's smart contracts.

## REFERENCES

- Solidity Documentation: https://docs.soliditylang.org
- Truffle Documentation: https://www.trufflesuite.com/docs/truffle/overview
- React Documentation: https://reactjs.org/docs/getting-started.html
- Web3.js Documentation: https://web3js.readthedocs.io/
- Metamask Documentation: https://docs.metamask.io/
- Ethereum Developer Tools: https://ethereum.org/developers/tools/
- Blockchain Basics: https://www.ibm.com/blockchain/what-is-blockchain