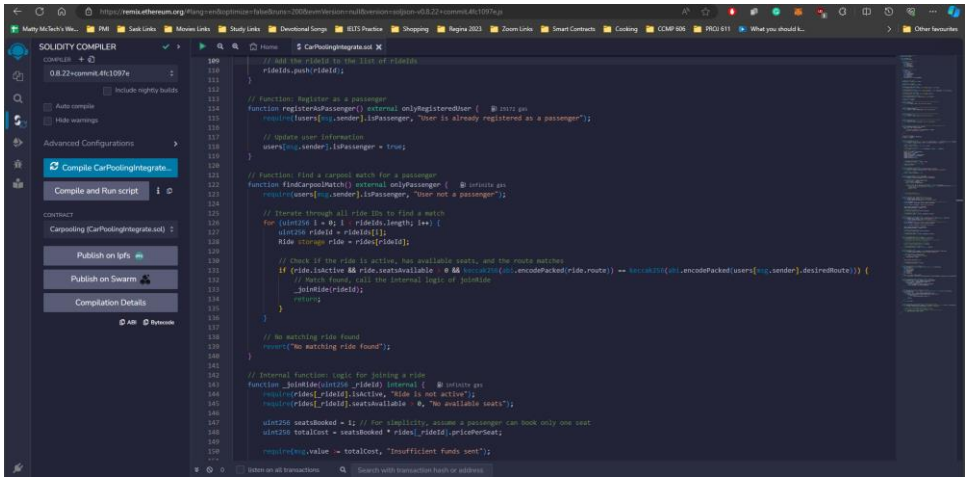
INTRODUCTION

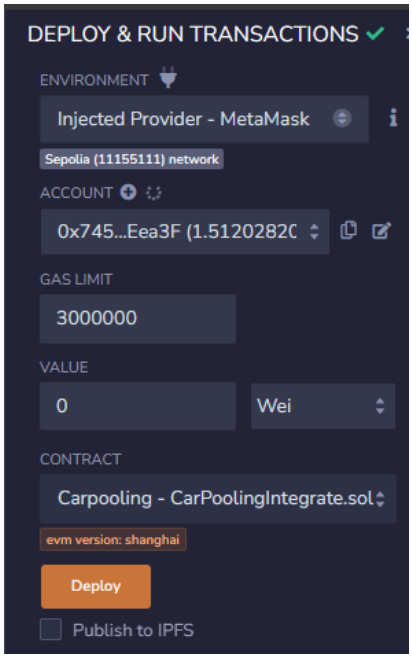
The goal of this project is to develop a decentralized application (Dapp) that facilitates secure carpooling using a blockchain-based smart contract. This Dapp will address the common issues related to trust, payment, and safety in carpooling, providing a secure and transparent solution for users.

IMPLEMENTATION

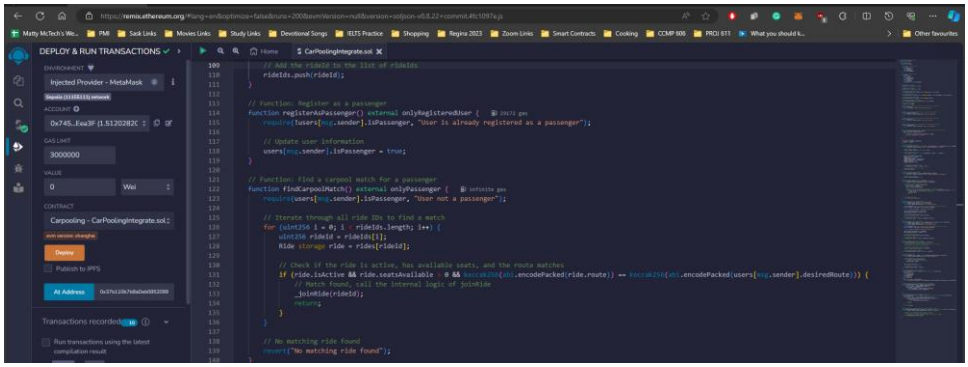
In Remix IDE, save the smart contract and compile.



In Metamask, ensure that Sepolia Testnet network has been connected.



In Remix IDE, deploy the smart contract and get the contract address.



Function 1: Registration as a Driver

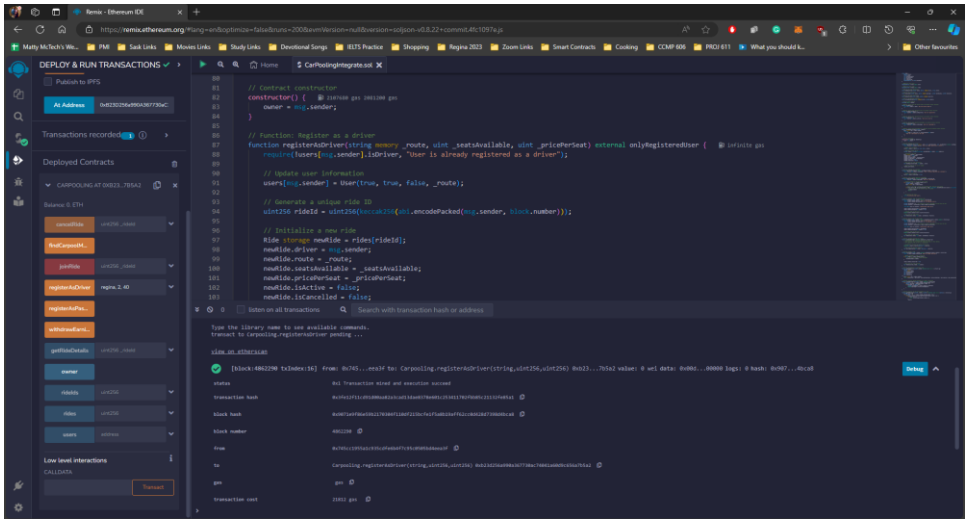
Coding:

```
// Function: Register as a driver
function registerAsDriver(string memory _route, uint _seatsAvailable, uint _pricePerSeat) external onlyRegisteredUser {
    require(!users[msg.sender].isDriver, "User is already registered as a driver");

    // Update user information
    users[msg.sender] = User(true, true, false, _route);

    // Generate a unique ride ID
    uint256 rideId = uint256(keccak256(abi.encodePacked(msg.sender, block.number)));

    // Initialize a new ride
    Ride storage newRide = rides[rideId];
    newRide.driver = msg.sender;
    newRide.route = _route;
    newRide.seatsAvailable = _seatsAvailable;
    newRide.pricePerSeat = _pricePerSeat;
    newRide.isActive = false;
    newRide.isCancelled = false;

    // Emit event for ride registration
    emit RideRegistered(rideId, msg.sender, _route);

    // Add the rideId to the list of rideIds
    rideIds.push(rideId);
}
```

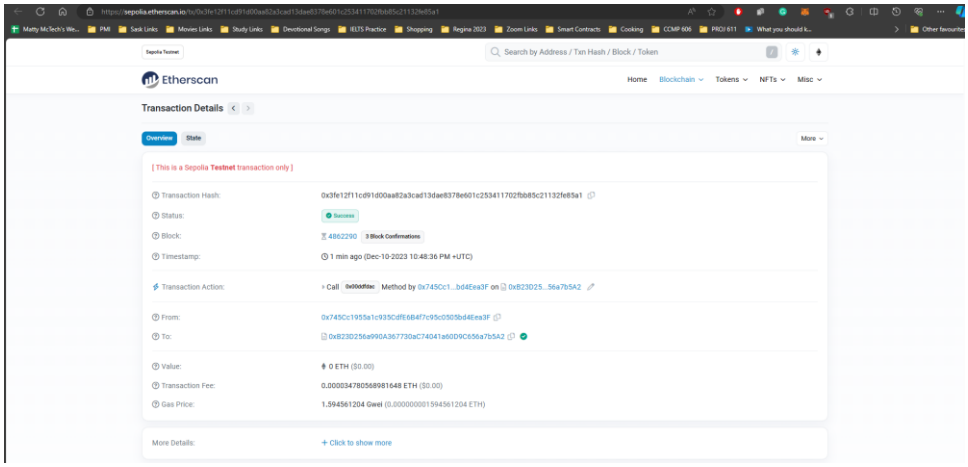
At Address: 0xB23D256a990A367730aC74041a60D9C656a7b5A2

Explanation:

This function allows a registered user to become a driver by updating their information, generating a unique ride ID, creating a new ride with the specified details, emitting an event to log the registration, and adding the ride ID to the list of registered rides.



In Etherscan (Sepolia TestNet network), search for the transaction details.



Function 2: Registration as a Passenger

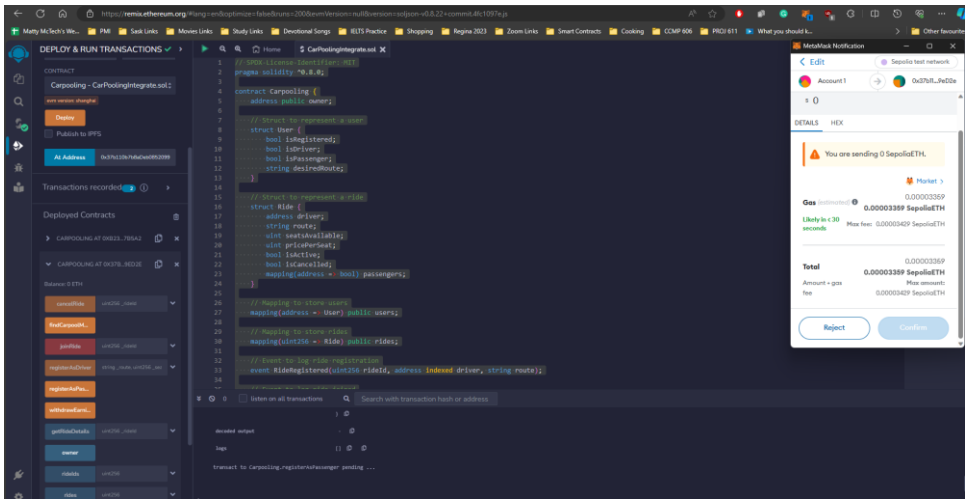
Coding

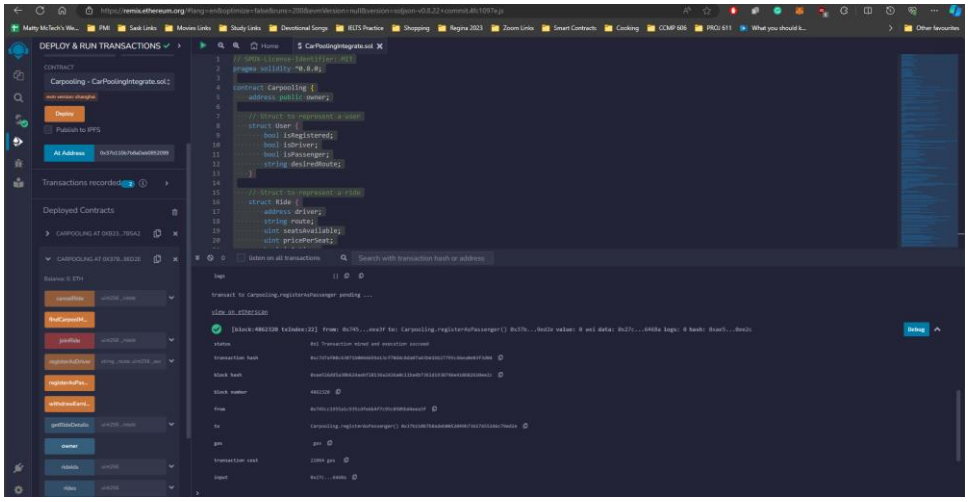
```
// Function: Register as a passenger
function registerAsPassenger() external onlyRegisteredUser {    29172 gas
    require(!users[msg.sender].isPassenger, "User is already registered as a passenger");

    // Update user information
    users[msg.sender].isPassenger = true;
}
```

At Address: 0x37b110b7b8aDeb0852099b73617455246C79eD2e

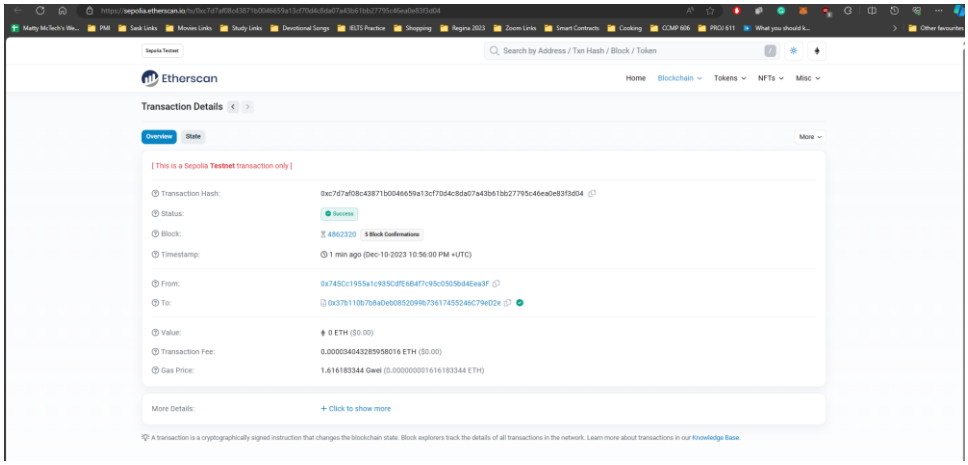
Explanation:

This function allows a registered user to become a passenger by updating their information. Before updating, it checks whether the user is already registered as a passenger to avoid duplicate registrations. The function is part of a larger smart contract, and it assumes that user registration information is stored in a mapping called users. The onlyRegisteredUser modifier ensures that only users who are already registered can execute this function.





In Etherscan (Sepolia TestNet network), search for the transaction details.



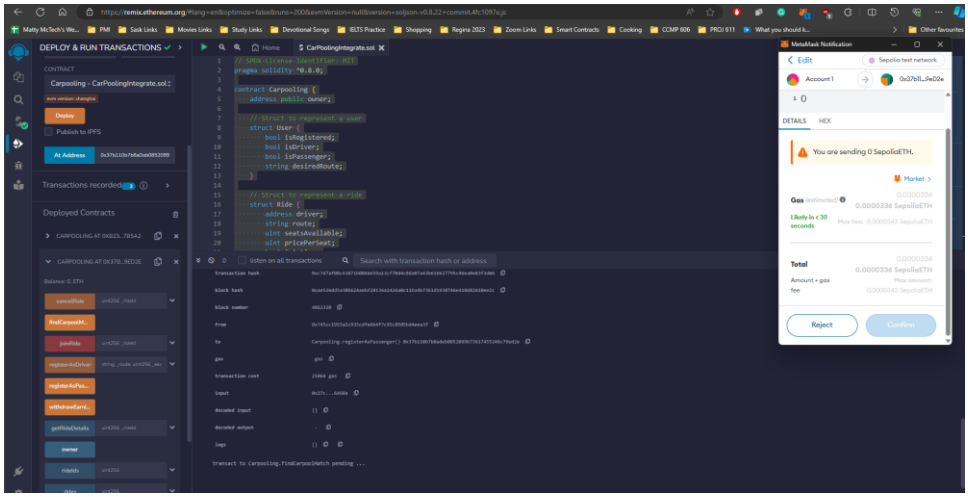
Function 3: To find Carpooling Match

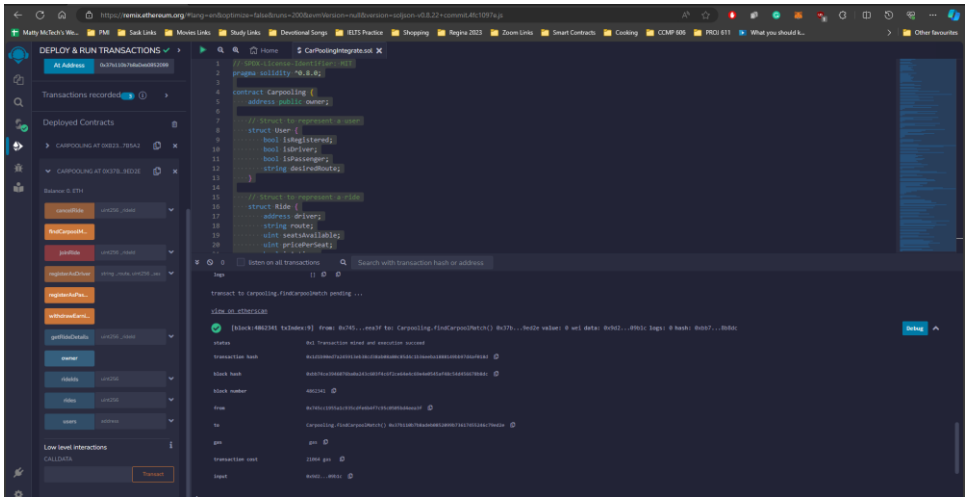
Coding:

```
// Function: Find a carpool match for a passenger
function findCarpoolMatch() external onlyPassenger {
    require(users[msg.sender].isPassenger, "User not a passenger");

    // Iterate through all ride IDs to find a match
    for (uint256 i = 0; i < rideIds.length; i++) {
        uint256 rideId = rideIds[i];
        Ride storage ride = rides[rideId];

        // Check if the ride is active, has available seats, and the route matches
        if (ride.isActive && ride.seatsAvailable > 0 && keccak256(abi.encodePacked(ride.route)) == keccak256(abi.encodePacked(users[msg.sender].desiredRoute))) {
            // Match found, call the internal logic of joinRide
            _joinRide(rideId);
            return;
        }
    }

    // No matching ride found
    revert("No matching ride found");
}
```

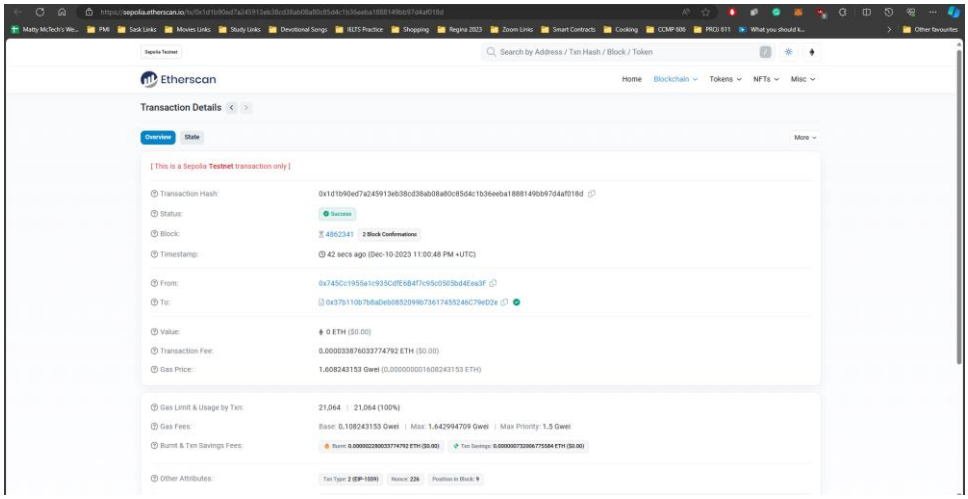
Explanation:

This function allows a registered passenger to find a carpool match by iterating through available rides and joining the first ride that meets the specified criteria (active, available seats, and matching route). If no matching ride is found, it reverts with an error message.





In Etherscan (Sepolia TestNet network), search for the transaction details.



Function 4: To join a ride

Coding:

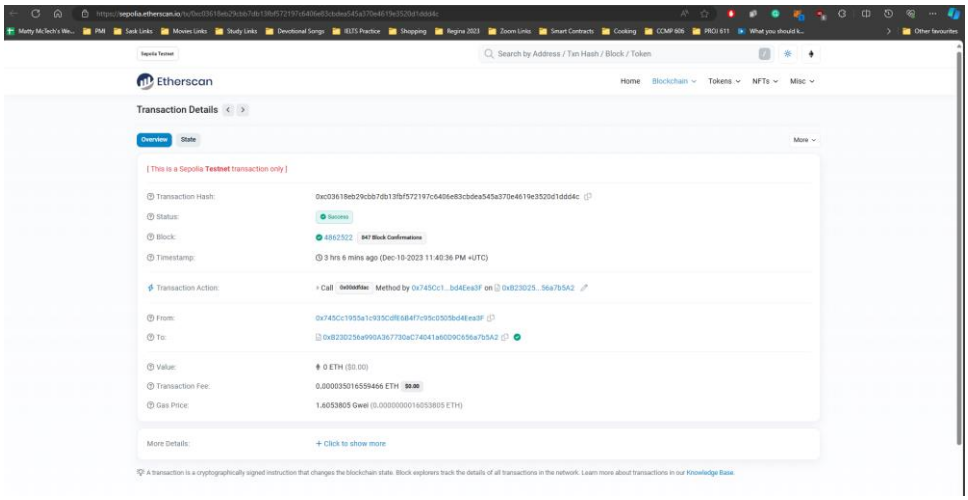
```
// Internal function: Logic for joining a ride
function _joinRide(uint256 _rideId) internal {    infinite gas
    require(rides[_rideId].isActive, "Ride is not active");
    require(rides[_rideId].seatsAvailable > 0, "No available seats");

    uint256 seatsBooked = 1; // For simplicity, assume a passenger can book only one seat
    uint256 totalCost = seatsBooked * rides[_rideId].pricePerSeat;

    require(msg.value >= totalCost, "Insufficient funds sent");

    rides[_rideId].seatsAvailable -= seatsBooked;
    rides[_rideId].passengers[msg.sender] = true;

    // Transfer funds to the driver
    payable(rides[_rideId].driver).transfer(totalCost);

    // Emit event for ride joined
    emit RideJoined(_rideId, msg.sender, seatsBooked, totalCost);
}
```

```
// Function: Allow a user to join a ride
function joinRide(uint256 _rideId) external onlyRideParticipant(_rideId) payable {    infinite gas
    require(rides[_rideId].isActive, "Ride is not active");
    require(rides[_rideId].seatsAvailable > 0, "No available seats");

    uint256 seatsBooked = 1; // For simplicity, assume a passenger can book only one seat
    uint256 totalCost = seatsBooked * rides[_rideId].pricePerSeat;

    require(msg.value >= totalCost, "Insufficient funds sent");

    rides[_rideId].seatsAvailable -= seatsBooked;
    rides[_rideId].passengers[msg.sender] = true;

    // Transfer funds to the driver
    payable(rides[_rideId].driver).transfer(totalCost);

    // Emit event for ride joined
    emit RideJoined(_rideId, msg.sender, seatsBooked, totalCost);
}
```

Explanation:

In summary, these functions handle the logic for a user to join a ride, ensuring the ride is active, seats are available, and the necessary funds are sent. The internal function _joinRide encapsulates the core logic, while the external function joinRide is the entry point for users to join rides.

In Etherscan (Sepolia TestNet network), search for the transaction details.



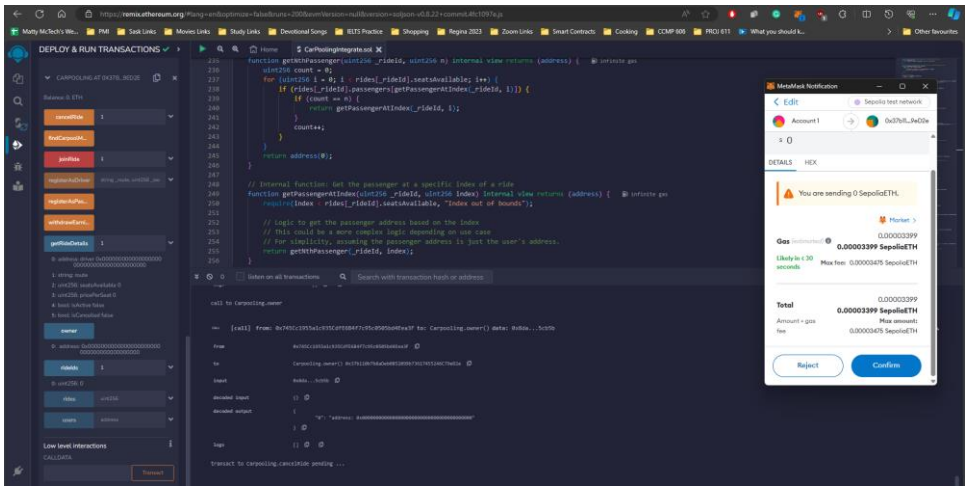
Function 5: To cancel a ride

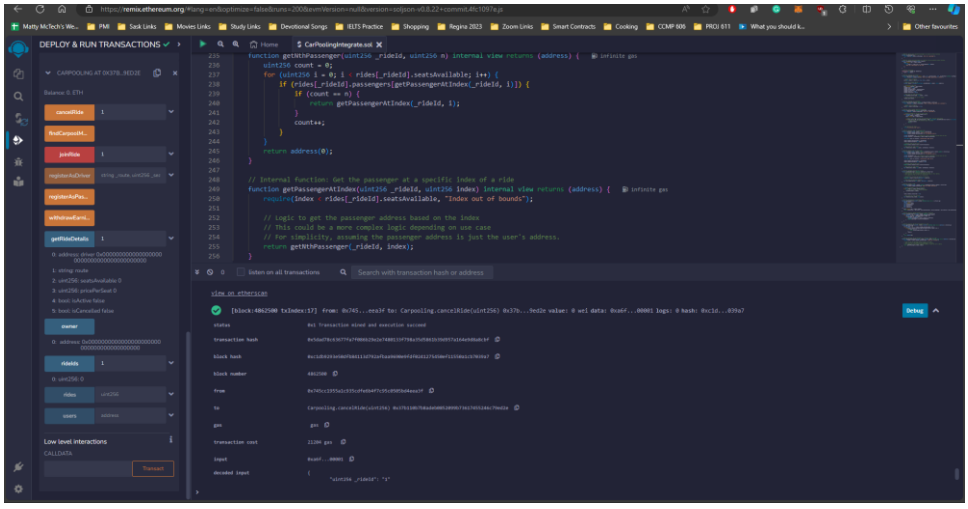
Coding:

```
// Function: Cancel a ride
function cancelRide(uint256 _rideId) external onlyRideParticipant(_rideId) {    infinite gas
    require(!rides[_rideId].isCancelled, "Ride is already cancelled");

    // Refund passengers if ride is cancelled
    if (msg.sender == rides[_rideId].driver) {
        // If the driver cancels, refund passengers
        refundPassengers(_rideId);
    }

    rides[_rideId].isCancelled = true;

    // Emit event for ride cancelled
    emit RideCancelled(_rideId, msg.sender);
}
```

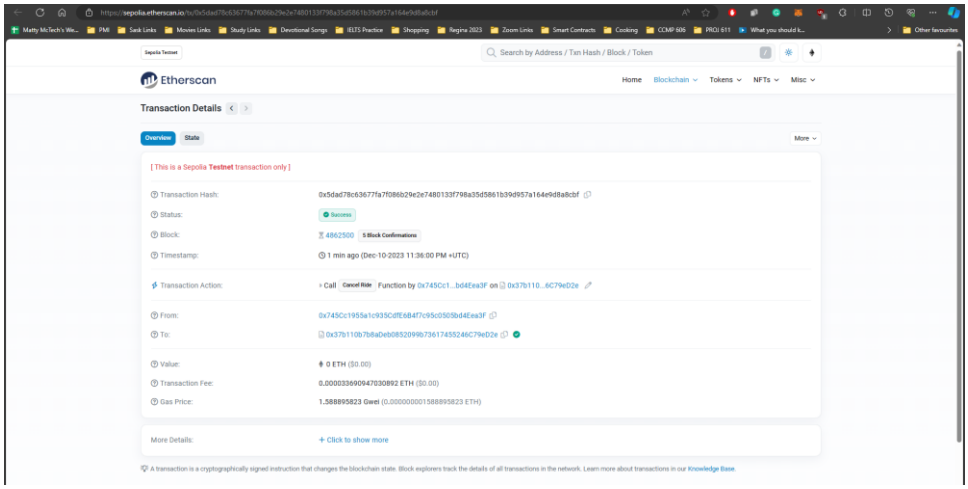
Explanation:

This function allows a ride participant (either the driver or a passenger) to cancel a ride, marking it as cancelled and emitting an event to log the cancellation. If the driver cancels, it also triggers the refund logic for passengers. The onlyRideParticipant modifier ensures that only participants of the specified ride can execute this function.





In Etherscan (Sepolia TestNet network), search for the transaction details.



FRONT-END PROTOTYPE

FUTURE WORK

- User Interface - Implementing enhancements to the user interface for improved usability.
- New Feature - Introducing a "Rate and Review" functionality to gather user feedback.
- Security Measures - Adding multiple layers of authentication for enhanced user security.
- Scalability - Exploring options to scale the application for increased user capacity.
- Integration - Integrating additional features to further enhance the user experience.
- Platform Compatibility - Ensuring compatibility with various devices and platforms for broader accessibility.