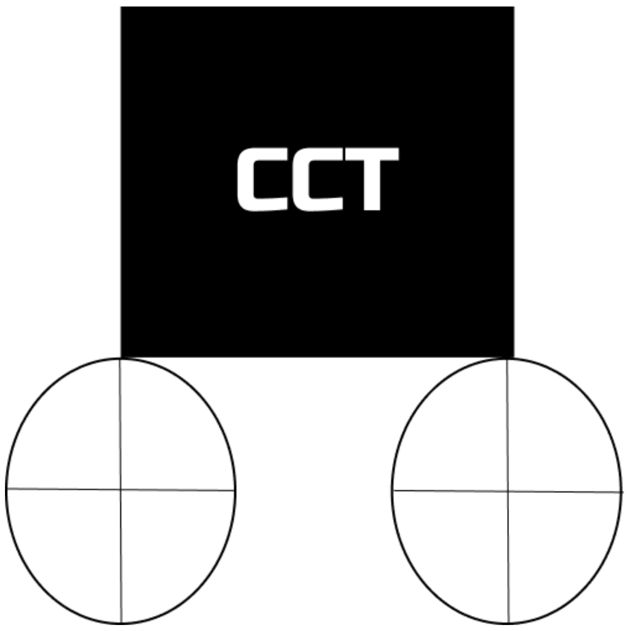
CarConTracks - Car Rental DApp Report

EECE 571G

Dr. Zehua Wang

Negar Yassaie, Sara Gargoum, Mateusz Faltyn

GitHub Repo: https://github.com/mattfaltyn/CarConTracksV1



Abstract

"On-demand" car-sharing services such as Modo and evo have spread to many urban areas across the world. However, all of these services suffer from issues of centralization such as security issues, privacy issues, and central points of failure; there is a need for decentralized solutions. Our aim in this project is to fill this gap in the market by creating a peer-to-peer car rental decentralized application. CarConTracks is an Ethereum-based DApp that allows individuals with the appropriate legal qualifications (i.e., age and driver's license) to rent a car for a specific duration of time. Instead of signing in to an app of a centralized carsharing service provider such as Modo or evo, users of CarConTracks will be able to rent a car in a peer-to-peer manner anywhere across the globe where cars are located.

Keywords: Car Rental, Blockchain, Decentralized Application

Table of Contents

Introduction

As COVID-19 pandemic restrictions continue to lift across Canada and much of the rest of the world, the commuting landscape looks very different than it had before March 2020. With many large knowledge work workspaces, such as university and tech campuses, closing for intermittent periods, working from home eliminated the need for a daily commute for many individuals. Knowledge workers who were able to shift most, if not all, of their work to home were posed with an interesting question: without a commute to work, what is the need for purchasing a vehicle?

The overall trend is clear: individuals want to increasingly rent, not buy, vehicles [1], [2]. Global vehicle sales have been decreasing since their peak in 2016 with a decrease of around 13.8% between 2019 and 2022 [3]. Additionally, global vehicle production has fallen from around 97 million per year in 2017 to 78 million per year in 2020 [3]. With the rise of app-based “on-demand” transportation options such as ride-hailing, carsharing, and scooter-rentals, individuals living in ever increasingly urban areas are finding that renting a car is both the more affordable and accessible mode of transportation.

“On-demand” ride-hailing services such as Uber and Lyft have steadily seen a rise in customer usage in recent years. Similarly, “on-demand” car-sharing services such as Modo and evo have spread to many urban areas across the world. However, all of these services suffer from issues of centralization such as security issues, privacy issues, and central points of failure; there is a need for decentralized solutions. Our aim in this project is to fill this gap in the market by creating a peer-to-peer car rental decentralized application.

Background

Previous Work in Car Rental Apps

Many traditional applications have been developed for car rental and sharing in the Vancouver area. One of the most prominent existing businesses and applications is Modo [4], a car rental service that allows a user to pick up a car, rent it for a specific duration time, and return it to the same location as where it was picked up. Another car rental business and application that exists in the Vancouver area is evo [5]. In contrast to Modo, evo allows a user to rent a car for a specific duration and drop it off at any location listed on the app. To our knowledge, no DApp currently exists that fulfills the needs of car renters and rentees.

Car Rental Apps Using Blockchain

However, many limitations exist in traditional database car rental applications. One of the first major limitations is *mutability*. Data tampering is one of the major risks in centralized databases. If the central database is breached, valuable user information such as credit card information can be stolen. Another major limitation of traditional databases is a *lack of transparency*. There is a general lack of trust between renters and rentees within the car rental industry. If there is any fraud, it can be difficult to track in a centralized database. Blockchains offer solutions to the aforementioned problems. For an excellent introduction to blockchain technologies, see Casino et al, 2019 [6].

Business Model and Stakeholders

Business Model

CarConTracks has a simple commission-based business model: Rentees provide vehicles to renters while receiving 80% of the transaction value after gas fees. This commission-based business model has been tried and tested throughout a number of companies such as Uber and Lyft to great success across the world. All expenses related to vehicle operation will be covered by the renter during the specified rental period or the rentee outside of a specified rental period.

Stakeholders

The following are stakeholders of CarConTracks:

1. Renters - are users of the CarConTracks DApp who wish to rent a car for a specific period of time (consumer).
2. Rentees - are users of the CarConTracks DApp who wish to put their car up for rent for a specific period of time (provider).

Design

DApp Assumptions

To simplify the DApp architecture, the following assumptions were made:

1. Two types of users: renters and rentees.
2. Renter must be 18 years or above of age.
3. Renters must provide a valid license number.
4. Rental duration is one day.
5. Renter can only have at most one rental request.
6. Renter cannot make another rental until they cancel/complete the current rental.
7. A historical record of users must be saved.
8. Car rentals are offered at a fixed price.
9. Rentee can only offer one car for rent at a time.

DApp Architecture

Figure 1 displays the system architecture of the CarConTracks DApp.

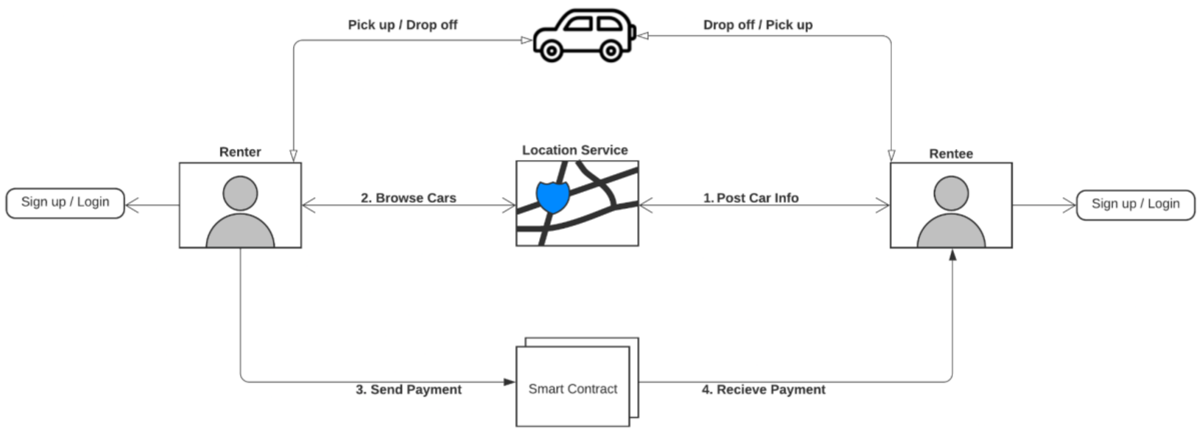


Figure 1. CarConTracks Architecture

Smart Contract - Basic Functionalities

1. Rentee will make their car available for rent and post their car information.

2. Renter will be asked to enter their name, age, and valid license number for verification purposes.
3. Renter is able to locate the nearest available car.
4. Renter will be able to identify the model description of the nearest available car.
5. Renter is able to create a rental order for said nearest available car.
6. Renter is able to confirm the rental order.
7. Renter is able to cancel the rental order.
8. Renter is able to return the car after the rental duration is complete.
9. Renter can manually update the web interface in addition to the interface automatically updating at a reasonable rate.

Smart Contract - Transactions

1. Renter must deposit ether into the smart contract to rent the nearest available car (gas fees apply).
2. Rentee must receive ether placed into the smart contract if the rental is successful (gas fees apply).
3. Renter must receive ether placed into the smart contract if they cancel their rent order (gas fees apply).

Frontend - Wireframes

The CarConTracks frontend is designed similar to many modern web applications that utilize the React Javascript framework.

Figure 2 displays the first section of the CarConTracks wireframes. This section includes user sign-up, user login, and user selection of either renting or loaning a car (i.e., renter vs rentee).

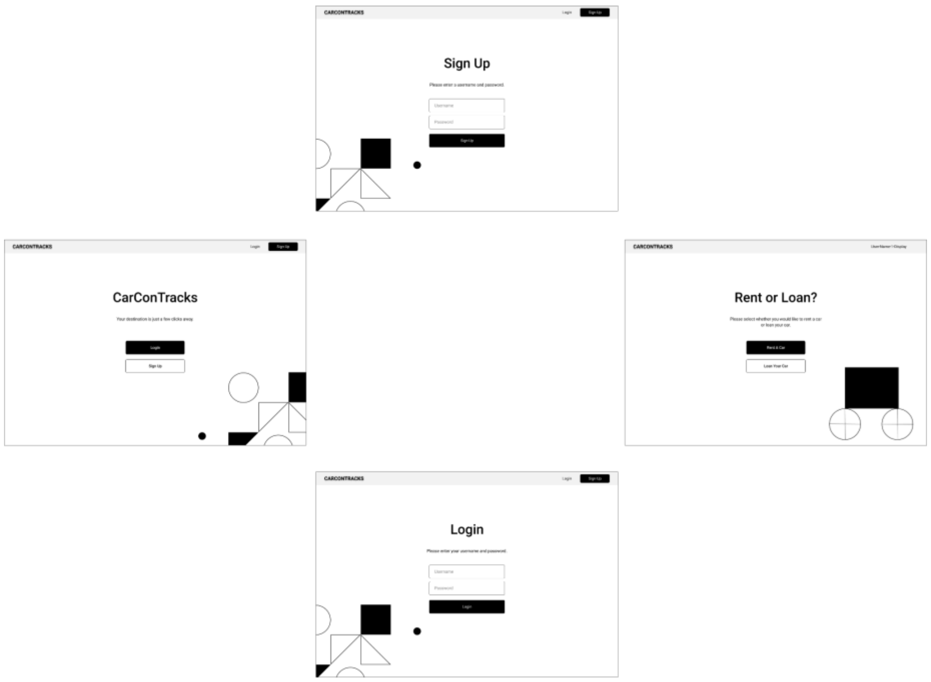


Figure 2. Wireframes - Section 1

Figure 3 displays the second section of the CarConTracks wireframes. This section is bifurcated into two components: one for a renter (top), and one for a rentee (bottom). The renter component includes car selection and confirmation, while the rentee component displays car information and confirmation.

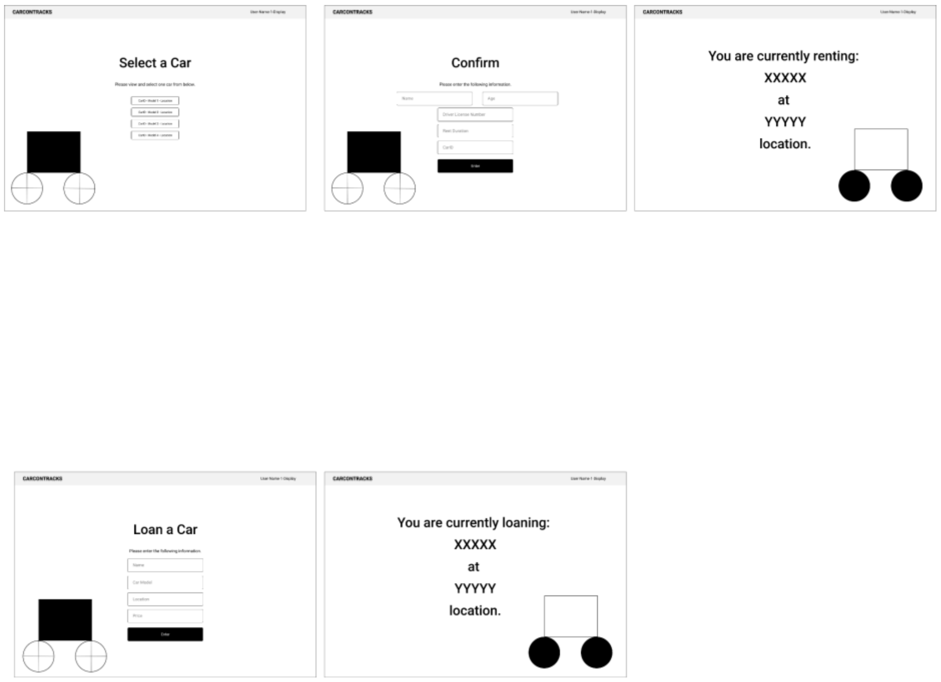


Figure 3. Wireframes - Section 2

Backend - Basic Functionalities

The CarConTracks backend is designed similar to many modern web applications that utilize traditional databases; however, we store our information on the Ethereum blockchain which provides a number of security improvements.

1. *User* struct will store the following information:
    a. Username
    b. Password
    c. User ID
    d. User address
    e. Whether or not the user is logged in
2. *CarInfo* struct will store the following information:
    a. Whether or not the car is available
    b. Car location
    c. Car ID
    d. Price

    e. Owner name

3. *CustomerInfo* struct will store the following information:
    a. Whether or not the rental is valid
    b. Customer name
    c. Customer age
    d. License ID
    e. Rental duration
    f. Whether or not the car has been returned
    g. Whether or not the rental has been confirmed

Implementation

Smart Contract UML

Figure 4 displays the CarRental smart contract UML.

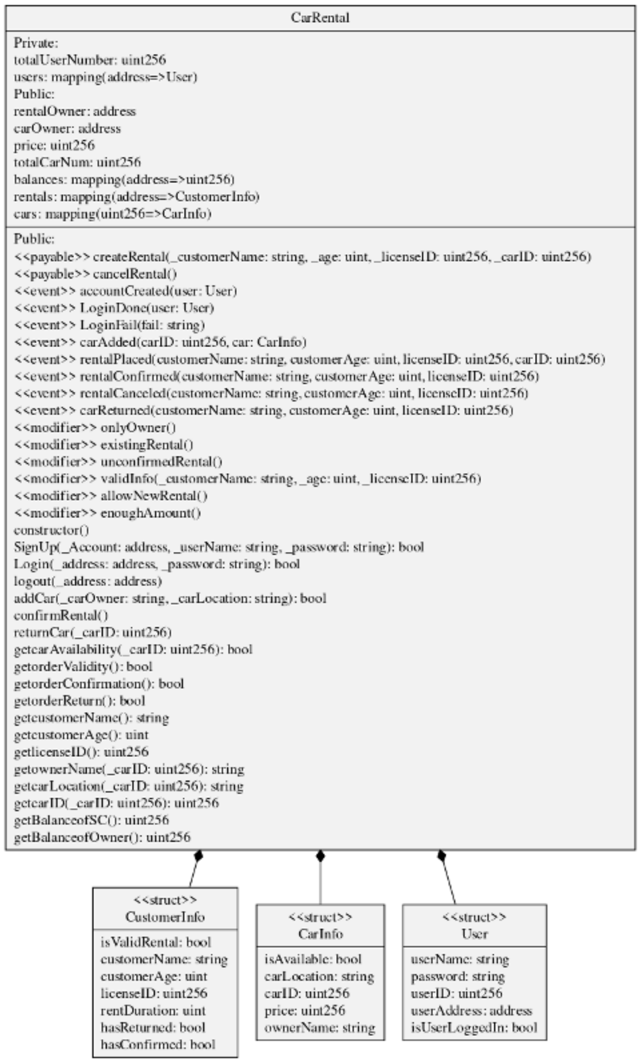
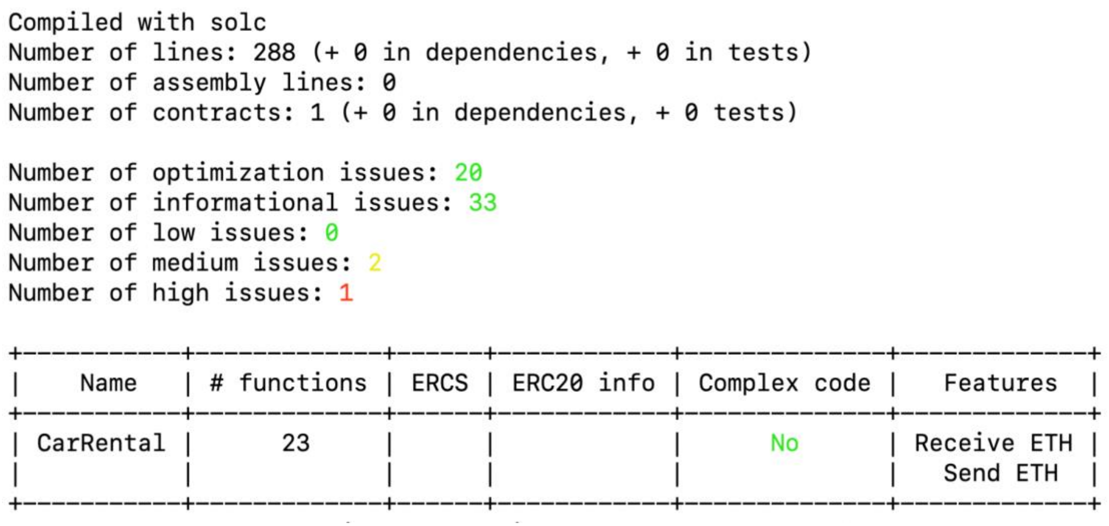

CarRental
Private:
totalUserNumber: uint256
users: mapping(address=>User)
Public:
rentalOwner: address
carOwner: address
price: uint256
totalCarNum: uint256
balances: mapping(address=>uint256)
rentals: mapping(address=>CustomerInfo)
cars: mapping(uint256=>CarInfo)
Public:
<<payable>> createRental(_customerName: string, _age: uint, _licenseID: uint256, _carID: uint256)
<<payable>> cancelRental()
<<event>> accountCreated(user: User)
<<event>> LoginDone(user: User)
<<event>> LoginFail(fail: string)
<<event>> carAdded(carID: uint256, car: CarInfo)
<<event>> rentalPlaced(customerName: string, customerAge: uint, licenseID: uint256, carID: uint256)
<<event>> rentalConfirmed(customerName: string, customerAge: uint, licenseID: uint256)
<<event>> rentalCanceled(customerName: string, customerAge: uint, licenseID: uint256)
<<event>> carReturned(customerName: string, customerAge: uint, licenseID: uint256)
<<modifier>> onlyOwner()
<<modifier>> existingRental()
<<modifier>> unconfirmedRental()
<<modifier>> validInfo(_customerName: string, _age: uint, _licenseID: uint256)
<<modifier>> allowNewRental()
<<modifier>> enoughAmount()
constructor()
SignUp(_Account: address, _userName: string, _password: string): bool
Login(_address: address, _password: string): bool
logout(_address: address)
addCar(_carOwner: string, _carLocation: string): bool
confirmRental()
returnCar(_carID: uint256)
getcarAvailability(_carID: uint256): bool
getorderValidity(): bool
getorderConfirmation(): bool
getorderReturn(): bool
getcustomerName(): string
getcustomerAge(): uint
getlicenseID(): uint256
getownerName(_carID: uint256): string
getcarLocation(_carID: uint256): string
getcarID(_carID: uint256): uint256
getBalanceofSC(): uint256
getBalanceofOwner(): uint256
<<struct>> CustomerInfo
isValidRental: bool
customerName: string
customerAge: uint
licenseID: uint256
rentDuration: uint
hasReturned: bool
hasConfirmed: bool
<<struct>> CarInfo
isAvailable: bool
carLocation: string
carID: uint256
price: uint256
ownerName: string
<<struct>> User
userName: string
password: string
userID: uint256
userAddress: address
isUserLoggedIn: bool

Figure 4. CarRental Smart Contract UML

Smart Contract Security

Smart contract security issues were first examined using *Slither* - a Python 3 Solidity static analysis framework created by TrailofBits [7].

Figure 5 displays a summary of the issues uncovered via Slither. Below can be found all of the issues in detail.



```
Compiled with solc
Number of lines: 288 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 1 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 20
Number of informational issues: 33
Number of low issues: 0
Number of medium issues: 2
Number of high issues: 1

+-----------+-------------+------+------------+--------------+-------------+
|    Name   | # functions | ERCS | ERC20 info | Complex code |   Features  |
+-----------+-------------+------+------------+--------------+-------------+
| CarRental |      23     |      |            |      No      | Receive ETH |
|           |             |      |            |              |   Send ETH  |
+-----------+-------------+------+------------+--------------+-------------+
```

Figure 5. Slither security summary.

High Issues

1. CarRental.balances (CarRental.sol#14) is never initialized. It is used in:
2. - CarRental.getBalanceofOwner() (CarRental.sol#284-286)
3. Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-variables

Medium Issues

1. CarRental.addCar(string,string)._car (CarRental.sol#193) is a local variable never initialized
2. CarRental.SignUp(address,string,string)._user (CarRental.sol#155) is a local variable never initialized
3. Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

Informational and Optimization Issues

1. CarRental.createRental(string,uint256,uint256,uint256) (CarRental.sol#210-223) compares to a boolean constant:
2. -require(bool,string)(_car.isAvailable == true,Car is not available.) (CarRental.sol#212)

3. CarRental.existingRental() (CarRental.sol#123-126) compares to a boolean constant:
4. -require(bool,string)(rentals[rentalOwner].isValidRental == true,Please confirm/cancel existing order) (CarRental.sol#124)
5. CarRental.unconfirmedRental() (CarRental.sol#127-130) compares to a boolean constant:
6. -require(bool,string)(rentals[rentalOwner].hasConfirmed == false,Cannot cancel confirmed rental) (CarRental.sol#128)
7. CarRental.allowNewRental() (CarRental.sol#136-139) compares to a boolean constant:
8. -require(bool,string)(rentals[rentalOwner].isValidRental == false,Please start new rental) (CarRental.sol#137)
9. Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
10. Pragma version^0.8.0 (CarRental.sol#2) allows old versions
11. solc-0.8.13 is not recommended for deployment
12. Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
13. Event CarRentalaccountCreated(CarRental.User) (CarRental.sol#65-67) is not in CapWords
14. Event CarRentalcarAdded(uint256,CarRental.CarInfo) (CarRental.sol#75-79) is not in CapWords
15. Event CarRentalrentalPlaced(string,uint256,uint256,uint256) (CarRental.sol#83-88) is not in CapWords
16. Event CarRentalrentalConfirmed(string,uint256,uint256) (CarRental.sol#90-96) is not in CapWords
17. Event CarRentalrentalCanceled(string,uint256,uint256) (CarRental.sol#98-104) is not in CapWords
18. Event CarRentalcarReturned(string,uint256,uint256) (CarRental.sol#105-111) is not in CapWords
19. Function CarRental.SignUp(address,string,string) (CarRental.sol#150-167) is not in mixedCase
20. Parameter CarRental.SignUp(address,string,string)._Account (CarRental.sol#150) is not in mixedCase
21. Parameter CarRental.SignUp(address,string,string)._userName (CarRental.sol#150) is not in mixedCase
22. Parameter CarRental.SignUp(address,string,string)._password (CarRental.sol#150) is not in mixedCase
23. Function CarRental.Login(address,string) (CarRental.sol#168-179) is not in mixedCase

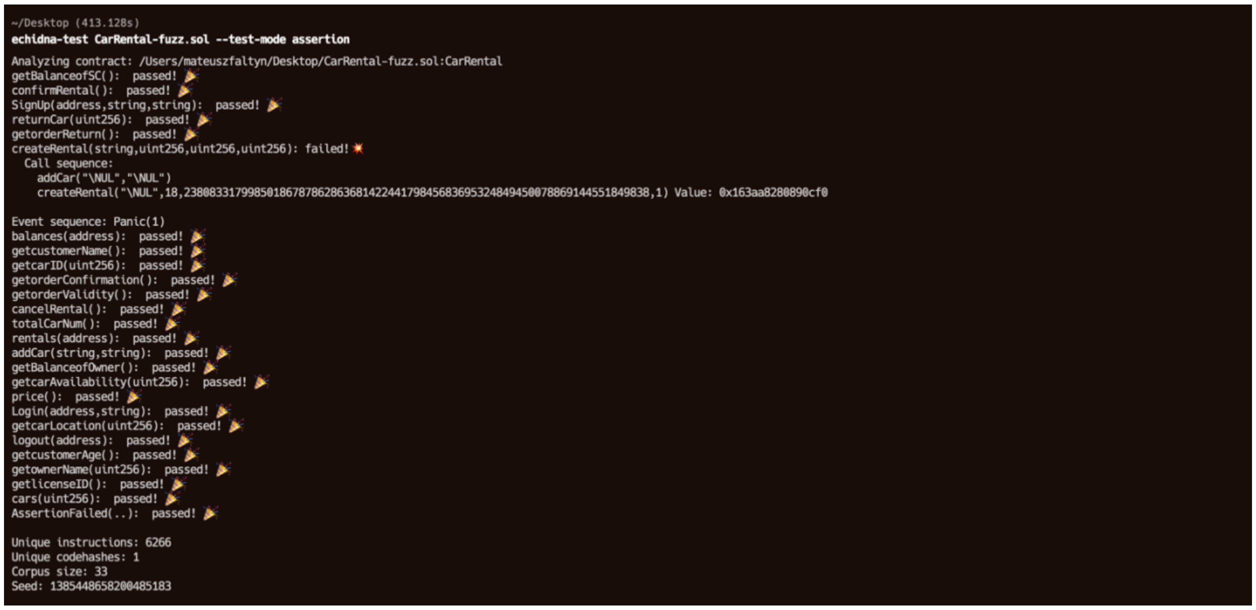
24. Parameter CarRental.Login(address,string)._address (CarRental.sol#168) is not in mixedCase
25. Parameter CarRental.Login(address,string)._password (CarRental.sol#168) is not in mixedCase
26. Parameter CarRental.logout(address)._address (CarRental.sol#180) is not in mixedCase
27. Parameter CarRental.addCar(string,string)._carOwner (CarRental.sol#187) is not in mixedCase
28. Parameter CarRental.addCar(string,string)._carLocation (CarRental.sol#187) is not in mixedCase
29. Parameter CarRental.createRental(string,uint256,uint256,uint256)._customerName (CarRental.sol#210) is not in mixedCase
30. Parameter CarRental.createRental(string,uint256,uint256,uint256)._age (CarRental.sol#210) is not in mixedCase
31. Parameter CarRental.createRental(string,uint256,uint256,uint256)._licenseID (CarRental.sol#210) is not in mixedCase
32. Parameter CarRental.createRental(string,uint256,uint256,uint256)._carID (CarRental.sol#210) is not in mixedCase
33. Parameter CarRental.returnCar(uint256)._carID (CarRental.sol#237) is not in mixedCase
34. Parameter CarRental.getcarAvailability(uint256)._carID (CarRental.sol#250) is not in mixedCase
35. Parameter CarRental.getownerName(uint256)._carID (CarRental.sol#272) is not in mixedCase
36. Parameter CarRental.getcarLocation(uint256)._carID (CarRental.sol#275) is not in mixedCase
37. Parameter CarRental.getcarID(uint256)._carID (CarRental.sol#278) is not in mixedCase
38. Constant CarRental.price (CarRental.sol#10) is not in UPPER_CASE_WITH_UNDERSCORES
39. Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
40. Reentrancy in CarRental.cancelRental() (CarRental.sol#230-236):
41. External calls:
42. - rentalOwner.transfer(price) (CarRental.sol#233)
43. Event emitted after the call(s):
44. - rentalCanceled(rentals[rentalOwner].customerName,rentals[rentalOwner].customerAge,rentals[rentalOwner].licenseID) (CarRental.sol#235)

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4
SignUp(address,string,string) should be declared external:
	- CarRental.SignUp(address,string,string) (CarRental.sol#150-167)
Login(address,string) should be declared external:
	- CarRental.Login(address,string) (CarRental.sol#168-179)
logout(address) should be declared external:
	- CarRental.logout(address) (CarRental.sol#180-182)
addCar(string,string) should be declared external:
	- CarRental.addCar(string,string) (CarRental.sol#187-206)
createRental(string,uint256,uint256,uint256) should be declared external:
	- CarRental.createRental(string,uint256,uint256,uint256) (CarRental.sol#210-223)
confirmRental() should be declared external:
	- CarRental.confirmRental() (CarRental.sol#224-228)
cancelRental() should be declared external:
	- CarRental.cancelRental() (CarRental.sol#230-236)
returnCar(uint256) should be declared external:
	- CarRental.returnCar(uint256) (CarRental.sol#237-244)
getcarAvailability(uint256) should be declared external:
	- CarRental.getcarAvailability(uint256) (CarRental.sol#250-252)
getorderValidity() should be declared external:
	- CarRental.getorderValidity() (CarRental.sol#253-255)
getorderConfirmation() should be declared external:
	- CarRental.getorderConfirmation() (CarRental.sol#256-258)
getorderReturn() should be declared external:
	- CarRental.getorderReturn() (CarRental.sol#260-262)
getcustomerName() should be declared external:
	- CarRental.getcustomerName() (CarRental.sol#263-265)
getcustomerAge() should be declared external:
	- CarRental.getcustomerAge() (CarRental.sol#266-268)
getlicenseID() should be declared external:
	- CarRental.getlicenseID() (CarRental.sol#269-271)
getownerName(uint256) should be declared external:
	- CarRental.getownerName(uint256) (CarRental.sol#272-274)
getcarLocation(uint256) should be declared external:
	- CarRental.getcarLocation(uint256) (CarRental.sol#275-277)
getcarID(uint256) should be declared external:
	- CarRental.getcarID(uint256) (CarRental.sol#278-280)
getBalanceofSC() should be declared external:
```

83. - CarRental.getBalanceofSC() (CarRental.sol#281-283)
84. getBalanceofOwner() should be declared external:
85. - CarRental.getBalanceofOwner() (CarRental.sol#284-286)
86. Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external

To provide further insights, smart contract security issues were next examined using *Echidna* - a Haskell program designed for fuzzing/property-based testing of Ethereum smarts contracts created by TrailofBits [7].

Figure 6 displays a summary of the issues uncovered via Echidna.

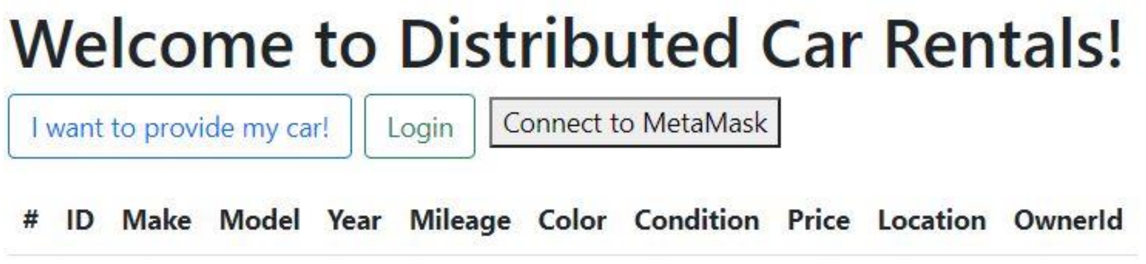


```
~/Desktop (413.128s)
echidna-test CarRental-fuzz.sol --test-mode assertion
Analyzing contract: /Users/mateuszfaltyn/Desktop/CarRental-fuzz.sol:CarRental
getBalanceofSC():  passed! 🎉
confirmRental():  passed! 🎉
SignUp(address,string,string):  passed! 🎉
returnCar(uint256):  passed! 🎉
getorderReturn():  passed! 🎉
createRental(string,uint256,uint256,uint256): failed! 💥
  Call sequence:
    addCar("\NUL","\NUL")
    createRental("\NUL",18,2380833179985018678786286368142244179845683695324849450078869144551849838,1) Value: 0x163aa8280890cf0

Event sequence: Panic(1)
balances(address):  passed! 🎉
getcustomerName():  passed! 🎉
getcarID(uint256):  passed! 🎉
getorderConfirmation():  passed! 🎉
getorderValidity():  passed! 🎉
cancelRental():  passed! 🎉
totalCarNum():  passed! 🎉
rentals(address):  passed! 🎉
addCar(string,string):  passed! 🎉
getBalanceofOwner():  passed! 🎉
getcarAvailability(uint256):  passed! 🎉
price():  passed! 🎉
Login(address,string):  passed! 🎉
getcarLocation(uint256):  passed! 🎉
logout(address):  passed! 🎉
getcustomerAge():  passed! 🎉
getownerName(uint256):  passed! 🎉
getlicenseID():  passed! 🎉
cars(uint256):  passed! 🎉
AssertionFailed(..):  passed! 🎉

Unique instructions: 6266
Unique codehashes: 1
Corpus size: 33
Seed: 1385448658200485183
```

Figure 6: Echidna security summary.

Front End

Figure 7 displays the initial banner that is displayed on a user's webpage when they access CarConTracks DApp.

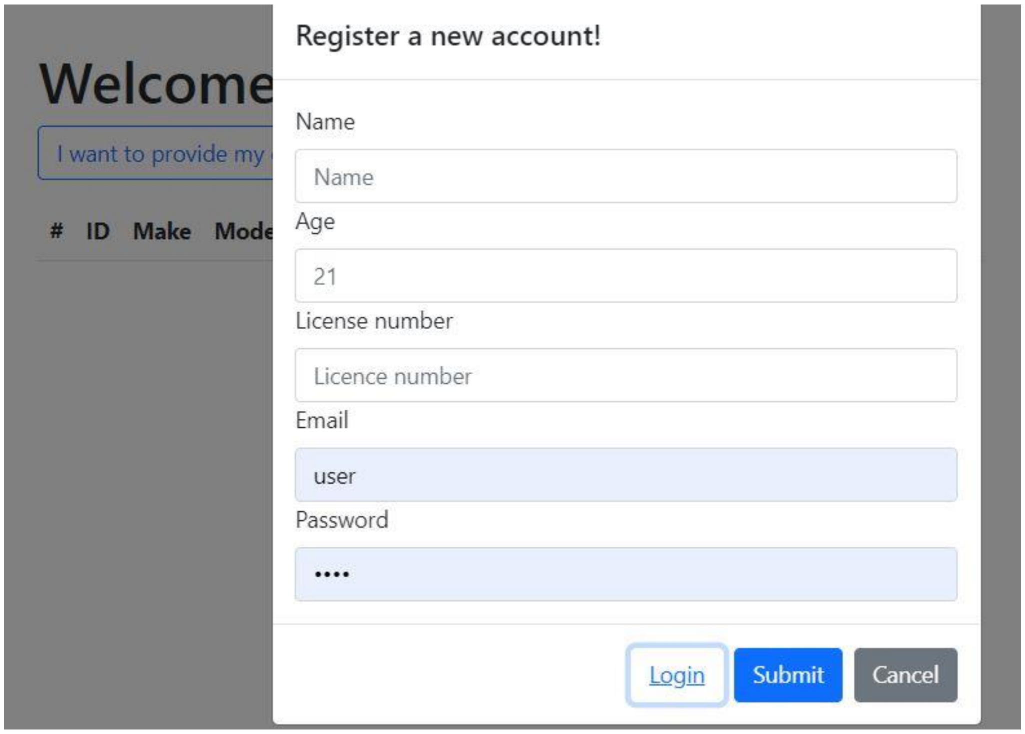

Welcome to Distributed Car Rentals!

I want to provide my car! | Login | Connect to MetaMask

#	ID	Make	Model	Year	Mileage	Color	Condition	Price	Location	OwnerId

Figure 7: Initial DApp banner.

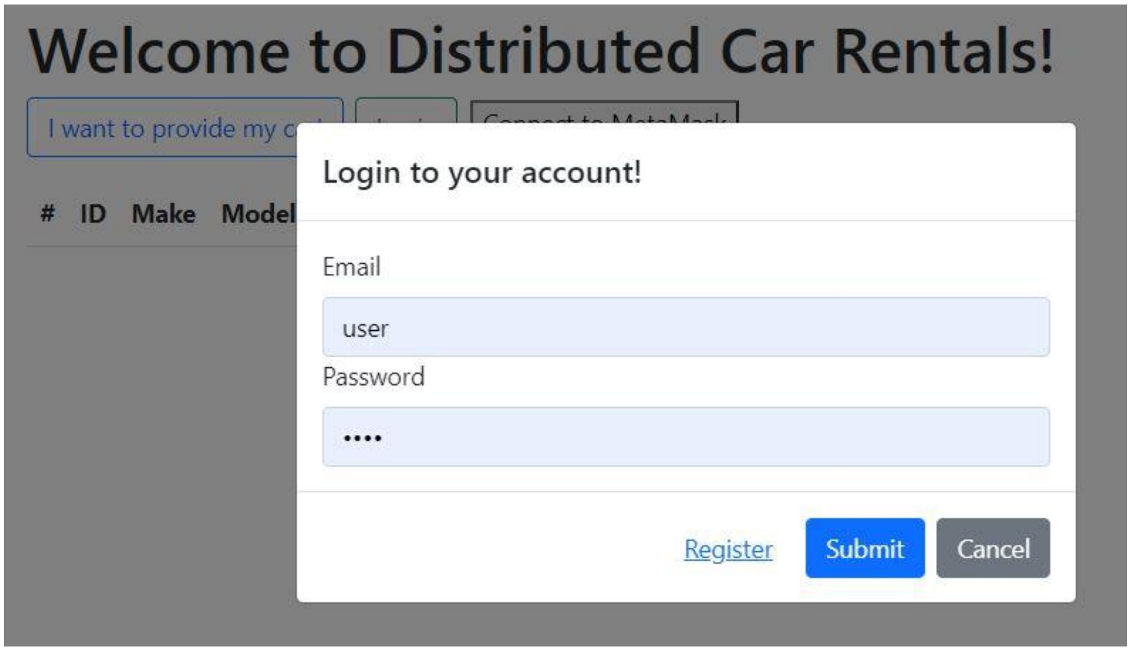
Once a user has accessed the dapp, they are able to register (Figure 8) or sign in (Figure 9) if they already have an existing account.


Register a new account!

Name: Name
Age: 21
License number: Licence number
Email: user
Password: ••••

Login | Submit | Cancel

Figure 8: Registration.



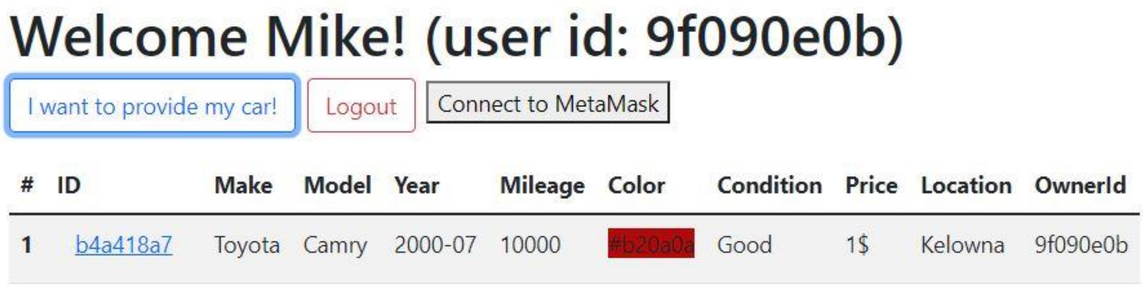
Login to your account!

Email: user

Password: ••••

Figure 9: Sign in.

Figure 10 displays the banner that is displayed on a user's webpage when they have successfully signed into the CarContracks DApp.

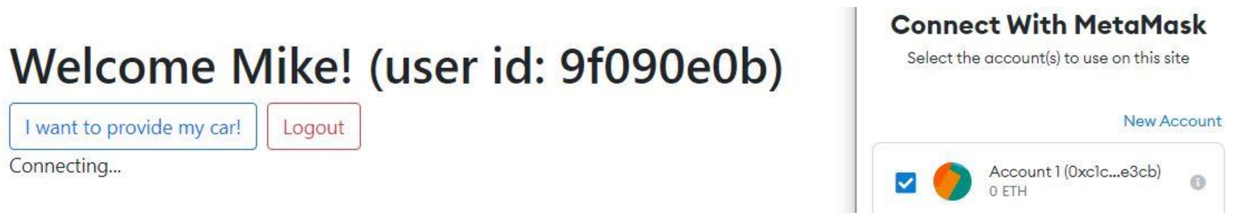


Welcome Mike! (user id: 9f090e0b)

#	ID	Make	Model	Year	Mileage	Color	Condition	Price	Location	OwnerId
1	b4a418a7	Toyota	Camry	2000-07	10000	#b20a0a	Good	1$	Kelowna	9f090e0b

Figure 10: User is logged in.

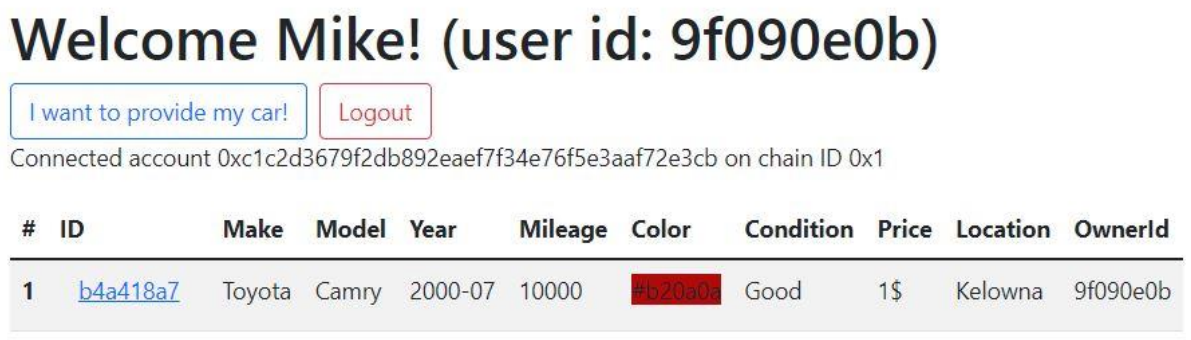
Figure 11 displays the banner that is displayed on a user's webpage when they are in the process of connecting to MetaMask.



Welcome Mike! (user id: 9f090e0b)

Figure 11: Connecting to MetaMask.

Figure 12 displays the banner that is displayed on a user's webpage when they have successfully logged into MetaMask.



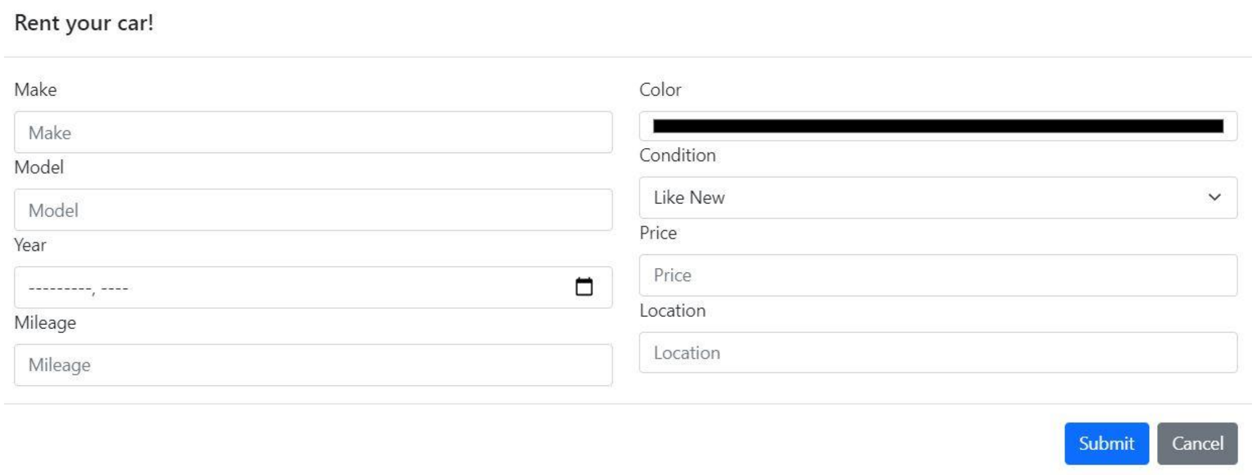
Welcome Mike! (user id: 9f090e0b)

Connected account 0xc1c2d3679f2db892eaef7f34e76f5e3aaf72e3cb on chain ID 0x1

#	ID	Make	Model	Year	Mileage	Color	Condition	Price	Location	OwnerId
1	b4a418a7	Toyota	Camry	2000-07	10000	#b20a0a	Good	1$	Kelowna	9f090e0b

Figure 12: Connected to MetaMask.

Figure 13 displays the user's webpage during the action of creating a new car loan.

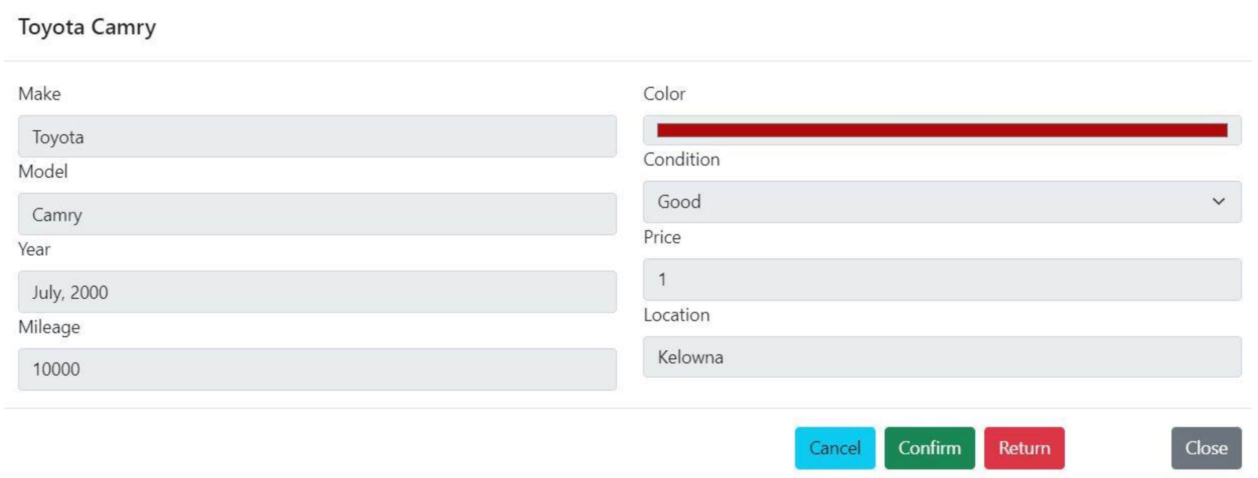


Rent your car!

Make: Make	Color:
Model: Model	Condition: Like New
Year: ---------, ----	Price: Price
Mileage: Mileage	Location: Location

Figure 13: Loaning a car.

Figure 14 displays the user’s webpage during the action of creating a new car rental.


Toyota Camry

Make	Toyota
Model	Camry
Year	July, 2000
Mileage	10000
Color	🟥
Condition	Good
Price	1
Location	Kelowna

Figure 14: Renting a car.

Tests

Smart Contract Tests

Table 1 displays the tests for the CarRental.sol smart contract.

Test Number	**Test Description**	**Passing?**
1	It should deploy smart contract properly.	Yes
2	It should provide owner name, unique car ID, and car location.	Yes
3	It should provide customer name, valid age, License number and car ID.	Yes
4	It should be a valid rental to confirm.	Yes
5	It should be an unconfirmed order to be able to cancel.	Yes
6	It should confirm/cancel existing order before creating new order.	Yes
7	Customer should have enough ethers.	Yes
8	Rental can be done if everything is ok.	Yes
9	Rental can be confirmed if everything is ok.	Yes
10	It should have correct balance of ethers.	Yes
11	Rental can be returned if everything is ok.	Yes
12	Customer receive the money after order cancellation.	Yes
13	User should be able to sign up.	Yes
14	User should be able to login.	Yes

Table 1. Smart Contract Tests

Discussion

The main objective of the course project of EECE571G was to build a decentralized application (DApp) including but not limited to a DeFi or a DAO product using the Hardhat + React + Web3 environment on Ethereum [8]. As our team continued throughout the development process of CarConTracks, we realized many different things as we encountered a variety of challenges. Due to the short time limit of this project, many issues across smart contract development, testing, security, as well as frontend development, were encountered yet not fully patched. With this in mind, CarConTracks has many limitations which we will explore throughout this section as well as the *Future Work* section.

One of the major limitations that currently exists in CarConTracks is a lack of a communication platform within the DApp between renters and rentees. The current version of CarConTracks only contains user and customer information in contrast to many modern applications which allow a seamless chat function between buyers and sellers. Developing a simple chat tool would be the next step in this project.

Another major limitation of CarConTracks is the scalability of the DApp. In the current version, all information and functions related to the DApp are stored in the CarRental smart contract. However, Ethereum is currently experiencing extremely high gas fees which makes CarConTracks encounter several deployment issues if the number of users was to scale up. Developing multiple smart contracts would be another major step in the right direction for this project.

More limitations are included and expanded upon in the *Future Work* section.

Future Work

Advanced Functionalities

Time-permitting, we will implement the following features:

1. In the current implementation, each car and user have a unique ID in the frontend; the next step would be to connect these IDs to the CarRental.sol smart contract.
2. In the current implementation, all data is stored locally; this creates several security issues and would be one of the first things addressed in the future.
3. In the current implementation, we use a simple string for location; the next step would be to implement a Google Maps interface via the Google Maps API to display the nearest available car.

Some larger features to develop in the future could include the following:

4. Implement a communication platform between renters and rentees.
5. Develop multiple smart contracts instead of just one.
6. Implement a verification function to confirm whether a user has a valid license number.
7. Implement user compliance features such as KYC to the DApp.

Smart Contract Security Improvements

Due to time limitations, several vulnerabilities exist in the current version of the CarRental smart contract. Future work would involve patching the following medium and high issues as well as the flags raised by fuzzing with Echidna:

High Issues

1. CarRental.balances (CarRental.sol#14) is never initialized. It is used in:
2. - CarRental.getBalanceofOwner() (CarRental.sol#284-286)
3. Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-variables

Medium Issues

1. CarRental.addCar(string,string)._car (CarRental.sol#193) is a local variable never initialized

2. CarRental.SignUp(address,string,string)._user (CarRental.sol#155) is a local variable never initialized
3. Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

Tokenomics

Time-permitting, we will launch a new ERC-20 token entitled the Car Rental (CARR) token. In accordance with the tokenomics theory proposed by van Oordt in 2016 [9], the CARR token must take three major factors into account in order to maintain long-term price stability: 1) token utility, 2) total supply regulation, and 3) token adoption.

Token Utility

The CARR token will be used to post car information to initiate a loan as well as rent a car on CarConTracks. In essence, it will be the medium of exchange between the renter and the rentee. Without acquiring the CARR token, users will not be able to utilize the DApp.

In essence, the flow of CARR tokens will follow this path. 1) a renter buys *N* CARR tokens using software such as Metamask. 2) a renter selects a vehicle to rent for a duration of time and transfers $N - G$ (G = gas fees) tokens to the rentee. 3) In accordance with the aforementioned business model, the rentee will receive $R = 0.8\ (N - G)$ of the CARR tokens from the renter. 4) The rentee can now become a renter with their newly acquired CARR tokens or sell their CARR tokens for profit.

Total Supply Regulation

In order to not price out new renters or rentees, the total supply of the CARR token will be regulated using a model similar to that of other utility tokens such as the Basic Utility Token launched by Brave Software [10]. The exchange rate of the CARR token will be proportional to the number of car rentals completed while being inversely proportional to the number of tokens not in use during a specific time period. In essence, the CARR token will increase in value if the number of users transacting on the CarConTracks DApp decreases. Holding CARR tokens will equal the discounted expected future exchange rate minus the risk premium for the expected uncertainty in the future value of the CARR token. Taken together, this model will make up the supply and demand of the CARR token and help stabilize the token value.

Token Adoption

Due to the tokenomics described in the previous paragraph, new users will be incentivized to join the platform as the CARR token will increase in value if the number of car rentals drops in number. Since CarConTracks is a peer-to-peer car rental service, renters will have access to cars in areas wherever rentees are interested in putting their cars up for loan.

Conclusion

In this project, we have created a decentralized application, CarConTracks, for peer-to-peer car rentals and loans. CarConTracks uses the React Javascript framework for the frontend as well as Solidity and Ethereum for its backend. This application allows users to sign-up, login, rent a car or post their car information, and confirm the rental or loan. We have created an application that brings the security, transparency, and trustlessness capabilities of blockchain technologies to the car rental industry.

Acknowledgments

We would like to thank Dr. Wang as well as all of the guest speakers of EECE 571G Winter 2022 for providing vital information and feedback throughout this project. Additionally, we are extremely grateful for many open-source communities (Solidity, Ethereum, React, etc) for developing and releasing incredible tools and resources.

References

[1] M. Garaus and C. Garaus, “The Impact of the Covid-19 Pandemic on Consumers' Intention to Use Shared-Mobility Services in German Cities”. *Front. Psychol. 12:646593*. doi: 10.3389/fpsyg.2021.646593

[2] K. M. Kiran Raj and K. G. Nandha Kumar, “Impact of Covid-19 Pandemic in the Automobile Industry: A Case Study”, *International Journal of Case Studies in Business*, IT, and Education (IJCSBE), vol. 5, no. 1, pp. 36–49, Feb. 2021, doi: 10.5281/zenodo.4505772.

[3] "Topic: Automotive industry worldwide", Statista, 2022. [Online].

[4] "Modo | Carsharing made easy", Modo.coop, 2022. [Online]. Available: https://modo.coop/. [Accessed: 20- Mar- 2022].

[5] "Car Sharing Vancouver", Evo, 2022. [Online]. Available: https://evo.ca/. [Accessed: 20- Mar- 2022].

[6] F. Casino, T. Dasaklis and C. Patsakis, "A systematic literature review of blockchain-based applications: Current status, classification and open issues", Telematics and Informatics, vol. 36, pp. 55-81, 2019. Available: 10.1016/j.tele.2018.11.006.

[7] "Trail of Bits", Trailofbits.com, 2022. [Online]. Available: https://www.trailofbits.com/. [Accessed: 20- Mar- 2022].

[8] V. Buterin, "A next-generation smart contract and decentralized application platform", Whitepaper, 2014

[9] W. Bolt and M. van Oordt, “On the Value of Virtual Currencies”, *Tech*, Working Paper, 2016.

[10] Brave Software. "Basic Attention Token (BAT): Blockchain-Based Digital Advertising", Whitepaper, 2021.