A Project Report

on

FOLA: A Decentralized Cab Rental Service

Submitted in partial fulfillment of the

requirement for the award of the degree of

Bachelor of Technology in Computer Science and Engineering

Under The Supervision of
Dr. Prashant Johri
Professor
Department of Computer Science and Engineering

Submitted By

18SCSE1050017 – CHHAVI TUTEJA

18SCSE1070012 – NITESH SAXENA

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GALGOTIAS UNIVERSITY, GREATER NOIDA, INDIA

DECEMBER - 2021

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA

CANDIDATE'S DECLARATION

I/We hereby certify that the work which is being presented in the project, entitled **"FOLA: A Decentralized Cab Rental Service"** in partial fulfillment of the requirements for the award of the **BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING** submitted in the **School of Computing Science and Engineering** of Galgotias University, Greater Noida, is an original work carried out during the period of **JULY-2021 to DECEMBER-2021**, under the supervision of **Dr. Prashant Johri, Professor**, **Department of Computer Science and Engineering** of School of Computing Science and Engineering, Galgotias University, Greater Noida

The matter presented in the project has not been submitted by me/us for the award of any other degree of this or any other places.

18SCSE1050017 - CHHAVI TUTEJA

18SCSE1070012 – NITESH SAXENA

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Supervisor

(Dr. Prashant Johri, Professor)

CERTIFICATE

The Final Thesis/Project/ Dissertation Viva-Voce examination of **18SCSE1050017 – CHHAVI TUTEJA, 18SCSE1070012 - NITESH SAXENA** has been held on _____and his/her work is recommended for the award of **BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING**.

Signature of Examiner(s) **Signature of Supervisor(s)**

Signature of Project Coordinator **Signature of Dean**

Date:

Place:

ABSTRACT

The introduction of car rental services via mobile applications has brought the ease of transportation to the customers. However, the pricing is quite controversial. Neither the customers nor the drivers get benefitted from this. Also, the need to trust the middleman for the availing services is a security concern among many people. There is a conflict in categorization of drivers as employees or service providers.

This led us to the idea of a Decentralized Cab Rental System. A decentralized cab system eliminates all the prime ill concerns of the traditional cab rental systems. It is decentralized, i.e., there is no central authority or middleman governing and interfering between the transacting parties. The fares can directly be transferred to the drivers from the customers with minimal charges. This system is based on Blockchain Technology. We'll use Solidity, JavaScript. For front end React Native and Socket.io. For Backend: MongoDB, Express.js, Ethereum Blockchain.

The **output** will be a car rental application that's based on Blockchain along with a research paper on "Blockchain Technology: A case study on its Decentralized Use".

Using the previously mentioned technologies, we're drafting a system that would prove to be beneficial not only for the users but also for the ones who are actually providing the services. The problem with categorization of drivers as Employees or service providers will now be removed. There is no need to trust a third party via our proposed system. We'll look for ways to publish this software and bring further reforms according to the requirements of the then time.

Table of Contents

List of Figures

Acronyms

FOLA	Free OLA
ETH	Ether
EVM	Ethereum Virtual Machine
dApp	Decentralized Application

CHAPTER-1

Introduction

1.1 Introduction

The introduction of car rental services via mobile applications has brought the ease of transportation to the customers. They don't have to walk or wait or find taxis, and can easily go about their way simply by pressing a few buttons on their mobile phones which come handy. Travelling from one place to another has now become hassle-free. However, with all the perks, come some disadvantages. The pricing is one of them.

One of the disadvantages of the now available cab rental services is that they offer variable pricing. In the rush hour the prices increase while in the night or early morning they stoop low. In festivals the prices are hiked in comparison to the non-festive days. They increase and decrease the prices according to the demands.

Another disadvantage for the driver is that it becomes inconvenient. Most of the times when a driver wants to be taking fares and making money, but there are less passenger compare to drivers and it is frustrating for drivers who wants to work, but not getting any passenger. There are expenses that are not reimbursed for example wear and tear on the car. Drivers are held responsible for all car expenses. There is a conflict in categorizing driver as the actual service provider rather than an employee who's working and will get due pay.

This led us to come up with the idea of creating a decentralized cab rental system which allows customers to directly come in contact and exchange services with money directly. In such a system no middleman is needed to regulate and govern the exchange between the transacting parties. Both the customers and actual service providers aka the drivers are benefitted in this manner. This system is based on the Blockchain Technology which offers benefits like traceability via timestamps and immutable transaction information, more security and hence reliable, decentralized as well as efficient and scalable.

1.2 Formulation of Problem

With all the disadvantages encountered in the present system as mentioned above there is a need to improvise the solution. A system that is decentralized, i.e., that does not involve a central governing system that improvises the rates according to the demands poses a good solution for the first problem of variable rates. This way the drivers can charge fees according to their services.

The second problem was the conflict in categorization of the drivers as employees who work for an organization or as the actual service providers. In such cab rental systems, usually the drivers are responsible for the maintenance and upkeep of the cabs. They have to pay due taxes with employee cut. This can be resolved if the drivers got all that they deserve according to the services they offer rather than the minimal employee leverage that they get.

The third and another major concern could be the need to put your trust on a third party to avail the services rather than transacting with the service provider. This might pose a security threat

For this we propose a decentralized cab rental system which is based on Blockchain Technology.

1.3 Merits of the Proposed System

There has been a paradigm shift to the emerging Blockchain technology due to the various benefits that it offers.

1. Full Control
One of the most significant advantages of decentralization is that it allows users to be in full control of their transactions. Therefore, they can start a transaction whenever they want and without the need to authorize it from a central authority. In other words, the verification process is independent from third parties.

2. Data cannot be altered nor deleted
Blockchain's data structure is an "add only" system, which means data cannot be modified or destroyed; any attempt to do so will become immediately obvious within the network because of the work the consensus algorithm does to make sure the data integrity is maintained. Moreover, Timestamps provide transparency.

3. Security
Tying into the point above, decentralized networks are exceptionally secure because of how they handle data and transactions. These networks use cryptography and hashing to create blocks, but also to ensure that the data ledgers are secure. The data in each block requires data from the block before and after it in the chain so that it can use cryptography to validate the data; the more transactions that occur on the network, the more blocks are created, and the more secure the data becomes: there is truly no weak link in the chain. While it's inaccurate to claim a blockchain ledger could never be hacked, it is accurate to assert that doing so would be impractical since it would require altering data in thousands if not millions of blocks while avoiding detection.

4. Lack of Censorship
In a centralized system, more information can be censored, while the decentralized paradigm leads to less censorship because no central authority controls the data. Take Twitter and Facebook, for example. Whether you believe these accusations hold water or not is up to you, but both social media giants are routinely accused of censoring accounts and content, which is ultimately easy to do since they are centralized platforms.

5. Open Development
Decentralized networks, by their nature of operation, support open development. By creating an environment of open development, the networks can add services, tools, and even products on top of the network itself. Linux, for example, is open-source and has an ecosystem allowing anyone to improve it. In contrast, centralized networks are closed source, which limits development.
The decentralized model is here to stay, and it has the potential to overturn the centralized model as

the de facto approach. Not only have we seen the rise of bitcoin and other cryptocurrencies, but some governments are considering what the future of voting may look like and whether the decentralized approach is viable, while the energy sector and Environmental, Social, and Governance are also considered the future on blockchain. In sum, this is an exciting time for decentralization and blockchain, the technology that makes it all possible.

CHAPTER-2

Literature Survey

In the existing system, the prices are based on supply and demand. In some areas the same distance fares are higher than the other areas which are considerably remote. There are expenses that are not reimbursed for example wear and tear on the car. Drivers are held responsible for all car expenses.

There is a need for an electronic system that allows two parties to transact digitally without the need of a third party organization. This system should be based on proof rather than trust. And a viable solution for this is offered by the inculcation of blockchain technology in our system. Blockchain is a shared, immutable ledger that facilitates the process of recording transactions and tracking assets in a business network.

Blockchain could be regarded as a public ledger, in which all committed transactions are stored in a chain of blocks. This chain continuously grows when new blocks are appended to it. Each block points to the previous block except the first block called Genesis Block. It offers enhanced security by creating a record that can't be altered and is encrypted end-to-end, blockchain helps prevent fraud and unauthorized activity.

It offers Greater transparency because blockchain uses a distributed ledger, transactions and data are recorded identically in multiple locations. All network participants with permissioned access see the same information at the same time, providing full transparency. Instant Traceability can be achieved because each block in blockchain holds unmodifiable information about all the transactions along with the timestamp for each, this is an added advantage. Each transaction can easily be traced.

CHAPTER-3

Functionality/Working of the Project

FOLA looks to become a decentralized sharing economy platform that combines the growing trend of peer-to-peer businesses with long-distance/inter-city traveling. Through a web application, FOLA connects Riders, those who would like a convenient way of getting from one city to another without driving, with our Drivers who happen to be going along that route and have extra seats in their vehicles. From a business standpoint, a distinct advantage that FOLA has over other solutions lies in its ability to maximize economic efficiency. The platform provides value to customers who now have a reliable and low-cost alternative for traveling, as well as commuters/drivers who are able to cut down or recoup the cost of gas for a journey they were planning on regardless.

But where does the blockchain and decentralization come into play? All data concerning delivery transactions are stored on the Ethereum blockchain network. In a nutshell, Ethereum gives you the power to trust. It enables the development of systems of automated and executable agreements facilitated by smart contracts that ensure that all counterparties are treated fairly throughout a transaction. Decentralization allows us to further maximize economic efficiency by replacing any profit-seeking intermediaries with smart contracts established directly between parties (Rider and Driver). This is unlike services like Uber and Lyft, which take a cut of the payment from the user of the service to the provider to compensate them for facilitating the transaction. We believe this reinforces our mission for creating positive change in a non-profit manner.

In addition, since Ethereum tracks every transaction non-repudiably on the blockchain, there is always a trusted record. This allows for better transparency and more trust between Riders and Drivers in this open development platform. Lastly, with on-chain data concerning the reputation of both Riders and Drivers and the success of their prior "transactions", the use of blockchain technology promises to help facilitate the decentralized credit scoring system of the future.

Installation Instructions

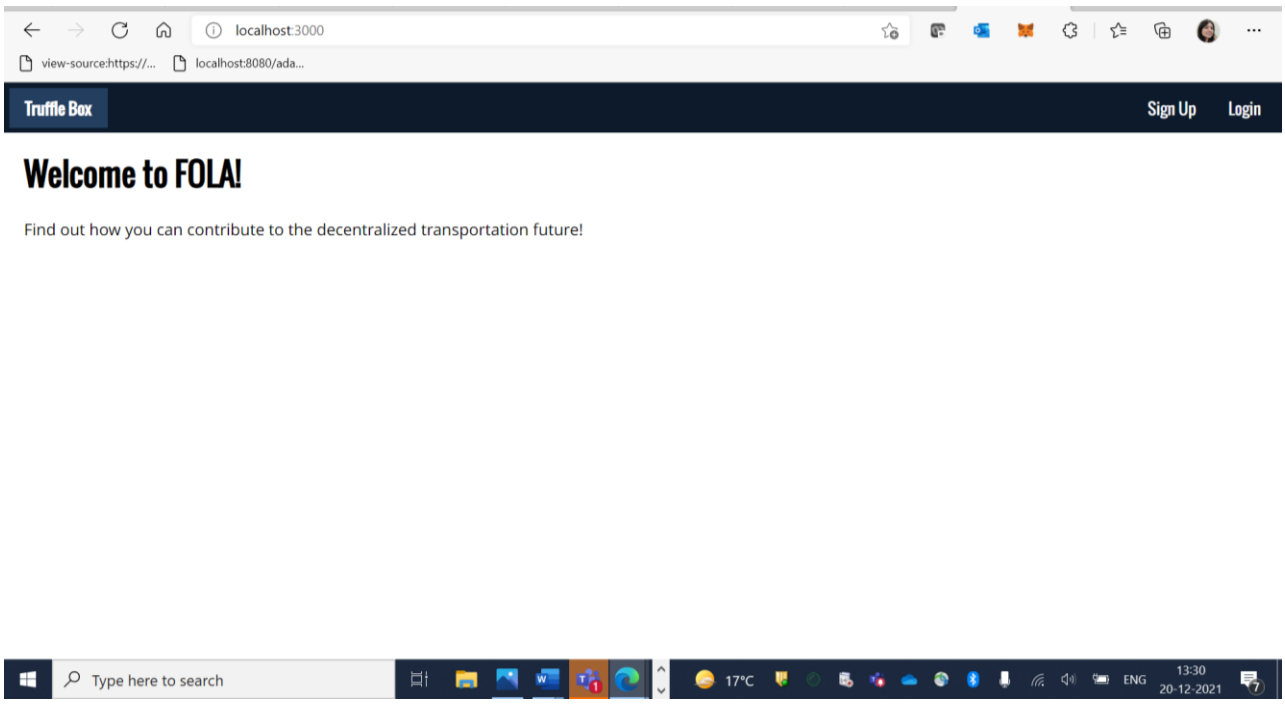
1. Install Node.js & npm
2. Download Ganache
3. Install Truffle

Instructions to run

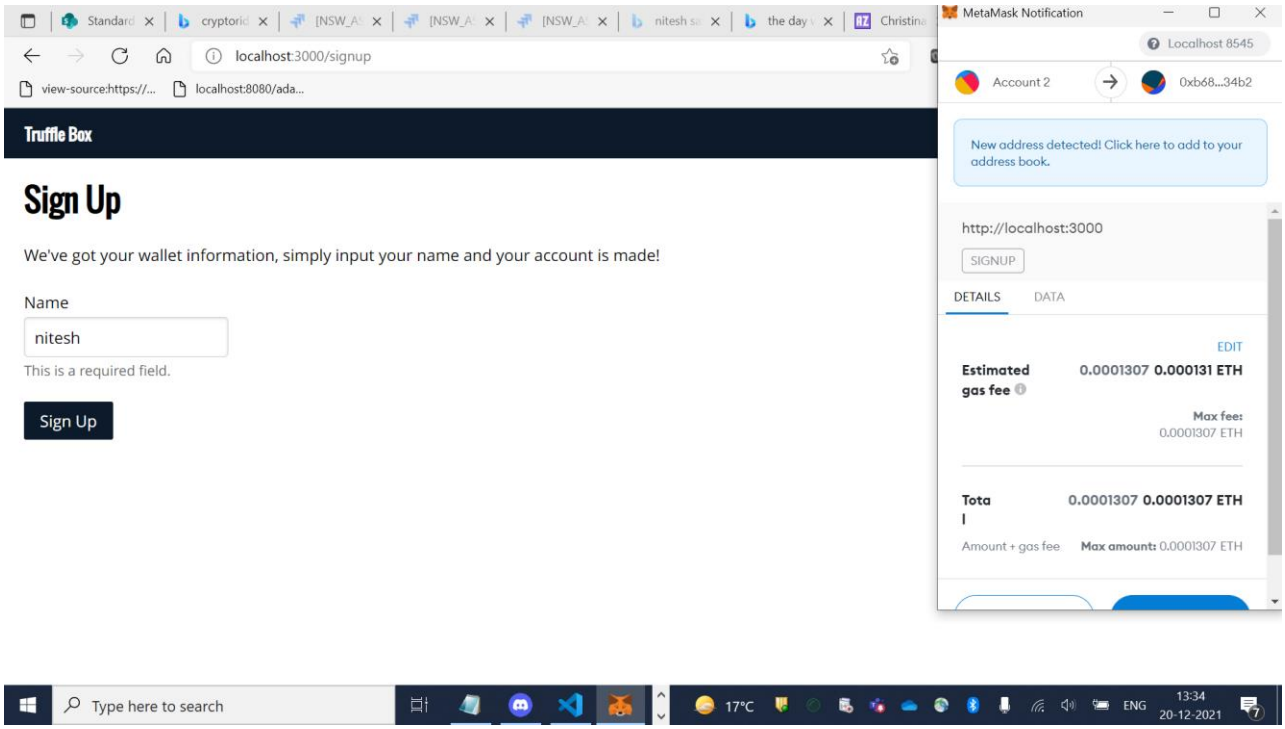
1. truffle compile
2. truffle deploy
3. npm start
4. Open in localhost in browser

Screenshots of the Application:

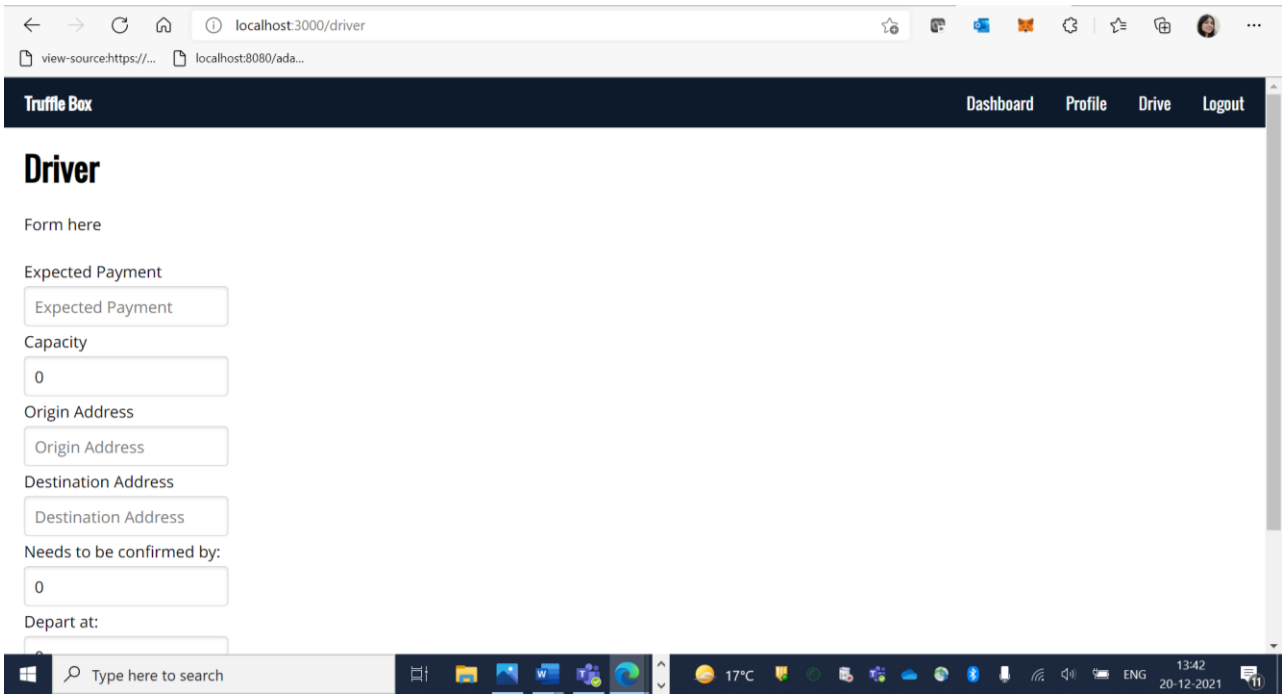
1. Home Page



2. SignUp



3. Create a Drive



SOURCE CODE

1. File : authentication.sol

```
pragma solidity ^0.4.2;

import './zeppelin/lifecycle/Killable.sol';

contract Authentication is Killable {
  struct User {
    bytes32 name;
  }

  mapping (address => User) public users;

  uint private id; // Stores user id temporarily

  modifier onlyExistingUser {
    // Check if user exists or terminate

    require(!(users[msg.sender].name == 0x0));
    _;
  }

  modifier onlyValidName(bytes32 name) {
    // Only valid names allowed

    require(!(name == 0x0));
    _;
  }

  function login() constant
  public
  onlyExistingUser
  returns (bytes32) {
    return (users[msg.sender].name);
  }

  function signup(bytes32 name)
  public
  payable
  onlyValidName(name)
  returns (bytes32) {
    // Check if user exists.
    // If yes, return user name.
    // If no, check if name was sent.
    // If yes, create and return user.

    if (users[msg.sender].name == 0x0)
    {
       users[msg.sender].name = name;

       return (users[msg.sender].name);
    }

    return (users[msg.sender].name);
```

```
    }

    function update(bytes32 name)
    public
    payable
    onlyValidName(name)
    onlyExistingUser
    returns (bytes32) {
      // Update user name.

      if (users[msg.sender].name != 0x0)
      {
          users[msg.sender].name = name;

          return (users[msg.sender].name);
      }
    }
  }
```

File : rideshare.sol

```
pragma solidity ^0.4.2;

import './zeppelin/lifecycle/Killable.sol';

contract Rideshare is Killable {
  struct Passenger {
    uint price;
    string state; // initial, driverConfirmed, passengerConfirmed, enRoute, completion, canceled
  }

  struct Ride {
    address driver;
    uint drivingCost;
    uint capacity;
    string originAddress;
    string destAddress;
    uint createdAt;
    uint confirmedAt;
    uint departAt;
    mapping (address => Passenger) passengers;
    address[] passengerAccts;
  }

  Ride[] public rides;
  uint public rideCount;

  mapping (address => uint) reputation;

  // for now, only drivers can create Rides
  function createRide(uint _driverCost, uint _capacity, string _originAddress, string _destAddress, uint _confirmedAt,
uint _departAt) {
    address[] memory _passengerAccts;
    rides.push(Ride(msg.sender, _driverCost, _capacity, _originAddress, _destAddress, block.timestamp, _confirmedAt,
```

```
_departAt, _passengerAccts));
  }

  function checkBalance() public view returns (uint){
    return address(this).balance;
  }

  // called by passenger
  function joinRide(uint rideNumber) public payable {
    Ride curRide = rides[rideNumber];
    require(msg.value == curRide.drivingCost);

    var passenger = curRide.passengers[msg.sender];

    passenger.price = msg.value;
    passenger.state = "initial";

    rides[rideNumber].passengerAccts.push(msg.sender) -1; //***
  }

  function getPassengers(uint rideNumber) view public returns(address[]) {
    return rides[rideNumber].passengerAccts;
  }

  function getPassengerRideState(uint rideNumber, address passenger) view public returns(string) {
    return rides[rideNumber].passengers[passenger].state;
  }

  function getRide(uint rideNumber) public view returns (
    address _driver,
    uint _drivingCost,
    uint _capacity,
    string _originAddress,
    string _destAddress,
    uint _createdAt,
    uint _confirmedAt,
    uint _departAt
  ) {
    Ride ride = rides[rideNumber];
    return (
      ride.driver,
      ride.drivingCost,
      ride.capacity,
      ride.originAddress,
      ride.destAddress,
      ride.createdAt,
      ride.confirmedAt,
      ride.departAt
    );
  }

  function getRideCount() public constant returns(uint) {
    return rides.length;
  }

  function passengerInRide(uint rideNumber, address passengerAcct) returns (bool) {
    Ride curRide = rides[rideNumber];
```

```
  for(uint i = 0; i < curRide.passengerAccts.length; i++) {
    if (curRide.passengerAccts[i] == passengerAcct) {
      return true;
    }
  }
  return false;
}

function cancelRide(uint rideNumber) {
  Ride curRide = rides[rideNumber];
  require(block.timestamp < curRide.confirmedAt);
  if (msg.sender == curRide.driver) {
    for (uint i = 0; i < curRide.passengerAccts.length; i++) {
      curRide.passengerAccts[i].transfer(curRide.passengers[curRide.passengerAccts[i]].price);
    }
  } else if (passengerInRide(rideNumber, msg.sender)) {
    msg.sender.transfer(curRide.passengers[msg.sender].price);
  }
}

// called by passenger
function confirmDriverMet(uint rideNumber) {
  require(passengerInRide(rideNumber, msg.sender));
  Ride curRide = rides[rideNumber];
  if (keccak256(curRide.passengers[msg.sender].state) == keccak256("passengersConfirmed")) {
    curRide.passengers[msg.sender].state = "enRoute";
  } else {
    curRide.passengers[msg.sender].state = "driverConfirmed";
  }
}

// called by driver
function confirmPassengersMet(uint rideNumber, address[] passengerAddresses) {
  Ride curRide = rides[rideNumber];
  require(msg.sender == curRide.driver);
  for(uint i=0; i < passengerAddresses.length; i++) {
    string curState = curRide.passengers[passengerAddresses[i]].state;
    if (keccak256(curRide.passengers[passengerAddresses[i]].state) == keccak256("driverConfirmed")) {
      curRide.passengers[passengerAddresses[i]].state = "enRoute";
    } else {
      curRide.passengers[passengerAddresses[i]].state = "passengersConfirmed";
    }
  }
  // require(rides[rideNumber].state == "confirmed");
}

function enRouteList(uint rideNumber) view public returns(address[]) {
  Ride curRide = rides[rideNumber];
  address[] addressesEnRoute;
  for(uint i = 0; i < curRide.passengerAccts.length; i++) {
    if (keccak256(curRide.passengers[curRide.passengerAccts[i]].state) == keccak256("enRoute")) {
      addressesEnRoute.push(curRide.passengerAccts[i]);
    }
  }
}

// called by passenger
```

```
  function arrived(uint rideNumber) {
    require(passengerInRide(rideNumber, msg.sender));
    Ride curRide = rides[rideNumber];
    curRide.driver.transfer(curRide.passengers[msg.sender].price);
    curRide.passengers[msg.sender].state = "completion";
  }
}
```

File: dashboard.js

```
import React, { Component } from 'react'
import RideList from './RideList'

class Dashboard extends Component {
  constructor(props, { authData }) {
    super(props)
    authData = this.props
  }

  render() {
    return(
      <main className="container">
        <div className="pure-g">
          <div className="pure-u-1-1">
            <h1>Dashboard</h1>
            <RideList/>
          </div>
        </div>
      </main>
    )
  }
}

export default Dashboard
```

File : ridelist.js

```
import React, { Component } from 'react'
import RideshareContract from '../../../build/contracts/Rideshare.json'
import store from '../../store'
import JoinRideContainer from '../../rideshare/ui/joinride/JoinRideContainer'
```

```
import { Link } from 'react-router'

const contract = require('truffle-contract')

class RideList extends Component {
  constructor(props) {
    super(props)
    this.state = {
      rideshares: [],
      passengers: [],
      rideshareLoading: true
    };
    this.getRides = this.getRides.bind(this);
    this.rideshareButton = this.rideshareButton.bind(this);
  }

  componentDidMount() {
    this.getRides();
  }

  getRides() {
    let web3 = store.getState().web3.web3Instance

    const rideshare = contract(RideshareContract)
    rideshare.setProvider(web3.currentProvider)

    // Declaring this for later so we can chain functions on Authentication.
    var rideshareInstance

    var _this = this;

    // Get current ethereum wallet.
    web3.eth.getCoinbase((error, coinbase) => {
      // Log errors, if any.
      if (error) {
        console.error(error);
      }

      rideshare.deployed().then(function(instance) {
        rideshareInstance = instance

        rideshareInstance.getRideCount.call()
        .then(function(result) {
          console.log('get rideshare count')
          console.log(result)
          let rideshareCount = result["c"][0];
```

```
        for (let i = 0; i < rideshareCount; i++) {
          rideshareInstance.getRide.call(i)
          .then(function(result) {
            // If no error, login user.
            console.log('getridesharecount')
            console.log(result)
            var tempArr = _this.state.rideshares;
            let tempRideshares = tempArr.concat([result]);
            _this.setState({rideshares: tempRideshares})
            console.log('test2');
            console.log(_this.state.rideshares);
            // debugger
            // return result;
            // return dispatch(loginUser())
          })
          rideshareInstance.getPassengers.call(i)
          .then(function(result) {
            var tempArr = _this.state.passengers;
            let tempPassengers = tempArr.concat([result]);
            _this.setState({passengers: tempPassengers})
          })
          // Attempt to sign up user.
          .catch(function(result) {
            // If error...
          })
        }
        _this.setState({rideshareLoading: false})
      })
    })
  })
}

rideshareButton(condition, bigNum,i) {
  let web3 = store.getState().web3.web3Instance
  console.log('passengers');
  console.log(this.state.passengers);
  if (condition) {
    return (
      <span>Leave</span>
    )
  } else {
    return (
      <JoinRideContainer ride_number={i} payment={web3.fromWei(bigNum, "ether"
).toNumber()}/>
    )
```

```
    }
  }

  render() {
    let web3 = store.getState().web3.web3Instance

    if (this.state.rideshareLoading) {
      return (
        <p>Loading</p>
      )
    } else {
      return(
        <main className="container">
          <div className="pure-g">
            <div className="pure-u-1-1">
              {this.state.rideshares.map((ride, i) => {
                console.log(ride);
                return (
                  <p>{ride[0]}, {web3.fromWei(ride[1], "ether" ).toNumber()},
{ride[2]["c"][0]}, {ride[3]}, {ride[4]},
                  {this.rideshareButton(this.state.passengers[i].indexOf(web3.eth.
accounts[0]) > -1, ride[1],i)}
                  <Link to={`/details/${i}`}>Details</Link>
                  </p>
                )
              })}
            </div>
          </div>
        </main>
      )
    }
  }
}

export default RideList
```

File: details.js

```
import React, { Component } from 'react'
import { Link } from 'react-router'
import RideDetails from './RideDetails'

class Details extends Component {
```

```
  constructor(props) {
    super(props)
  }

  render() {
    console.log(this.props);
    return(
      <main className="container">
        <div className="pure-g">
          <div className="pure-u-1-1">
            <RideDetails rideId={this.props.params.id}/>
          </div>
        </div>
      </main>
    )
  }
}

export default Details
```

File : RideDetails.js

```
import React, { Component } from 'react'
import RideshareContract from '../../../build/contracts/Rideshare.json'
import store from '../../store'
import JoinRideContainer from '../../rideshare/ui/joinride/JoinRideContainer'
import ConfirmDriverMetContainer from
'../../rideshare/ui/confirmDriverMet/ConfirmDriverMetContainer'
import ConfirmPassengersMetContainer from
'../../rideshare/ui/confirmPassengersMet/ConfirmPassengersMetContainer'
import ArrivedContainer from '../../rideshare/ui/arrived/ArrivedContainer'
import { Link } from 'react-router'

const contract = require('truffle-contract')

class RideDetails extends Component {
  constructor(props) {
    super(props)
    this.state = {
      ride: [],
      passenger: [],
      isPassenger: false,
      isDriver: false,
```

```
      passengerState: '',
      passengerStates: [],
      rideshareLoading: true,
      passengerLoaded: false,
    };
    this.getRides = this.getRides.bind(this);
  }

  componentDidMount() {
    this.getRides();
  }

  getRides() {
    let web3 = store.getState().web3.web3Instance

    const rideshare = contract(RideshareContract)
    rideshare.setProvider(web3.currentProvider)

    // Declaring this for later so we can chain functions on Authentication.
    var rideshareInstance

    var _this = this;

    // Get current ethereum wallet.
    web3.eth.getCoinbase((error, coinbase) => {
      // Log errors, if any.
      if (error) {
        console.error(error);
      }

      rideshare.deployed().then(function(instance) {
        rideshareInstance = instance

        rideshareInstance.getPassengers(_this.props.rideId)
        .then(function(result) {
          _this.setState({passenger: result})
          if (_this.state.passenger.indexOf(web3.eth.accounts[0]) > -1) {
            _this.setState({isPassenger: true});
            rideshareInstance.getPassengerRideState(_this.props.rideId,
web3.eth.accounts[0])
            .then(function(result) {
              _this.setState({passengerState: result})
            })
          }
          console.log(result);
          rideshareInstance.getRide(_this.props.rideId)
```

```
          .then(function(result) {
            _this.setState({ride: result})
            if (result[0] == web3.eth.accounts[0]) {
              for (var i = 0; i < _this.state.passenger.length; i++) {
                rideshareInstance.getPassengerRideState(_this.props.rideId,
_this.state.passenger[i])
                .then(function(result) {
                  var tempArr = _this.state.passengerStates;
                  let tempPassengerStates = tempArr.concat([result]);
                  _this.setState({passengerStates: tempPassengerStates});
                })
                _this.forceUpdate();
              }
            }
            _this.setState({passengerLoaded: true})
            console.log(result);
          })
        })
      })
    })
  }

  render() {
    let web3 = store.getState().web3.web3Instance

    if (this.state.passengerLoaded == false) {
      return (
        <p>Loading...</p>
      )
    } else {
      let rideId = this.props.rideId;
      let ride = this.state.ride;
      let isPassenger = (this.state.passenger.indexOf(web3.eth.accounts[0]) > -1);
      let isDriver = (ride[0] == web3.eth.accounts[0]);
      let passengerState = this.state.passengerState;
      let passengerStates = this.state.passengerStates;
      let passenger = this.state.passenger;

      if (isPassenger) {
        let confirmDriverMet;
        if (passengerState == "initial") {
          confirmDriverMet = <ConfirmDriverMetContainer ride_number={rideId}/>;
        } else if (passengerState == "enRoute") {
          confirmDriverMet = <ArrivedContainer ride_number={rideId} />
        }
```

```
        return(
          <main className="container">
            <div className="pure-g">
              <div className="pure-u-1-1">
                <p>You are a passenger</p>

                <p>Current state: {passengerState}</p>

                {confirmDriverMet}
              </div>
            </div>
          </main>
        )
      } else if (isDriver) {
        if (passengerStates.length == 0) {
          return (<p>Loading...</p>)
        } else {
          return(
            <main className="container">
              <div className="pure-g">
                <div className="pure-u-1-1">
                  <p>You are a driver</p>
                  <p>Passenger States: </p>

                  {passengerStates.length == 0 ? '' :
passengerStates.map((passengerState, i) => {
                    return (
                      <p>{passenger[i]} {passengerState}</p>
                    )
                  })}

                  <ConfirmPassengersMetContainer ride_number={rideId} />
                </div>
              </div>
            </main>
          )
        }
      } else {
        return(
          <main className="container">
            <div className="pure-g">
              <div className="pure-u-1-1">
                <p>Would you like to join this ride?</p>

                <JoinRideContainer ride_number={rideId}
payment={web3.fromWei(ride[1], "ether" ).toNumber()}/>
```

```
            </div>
          </div>
        </main>
      )
    }
  }
}

export default RideDetails
```

File : Driver.js

```
import React, { Component } from 'react'
import { Link } from 'react-router'
import CreateRideContainer from
'../../rideshare/ui/createride/CreateRideContainer'

class Driver extends Component {
  render() {
    return(
      <main className="container">
        <div className="pure-g">
          <div className="pure-u-1-1">
            <h1>Driver</h1>
            <p>Form here</p>
            <CreateRideContainer/>
          </div>
        </div>
      </main>
    )
  }
}

export default Driver
```

File : Home.js

```
import React, { Component } from 'react'
import { Link } from 'react-router'

class Home extends Component {
  render() {
    return(
      <main className="container">
        <div className="pure-g">
          <div className="pure-u-1-1">
            <h1>Good to Go!</h1>
            <p><Link to={`/dashboard`}>Dashboard</Link></p>
            <p>Your Truffle Box is installed and ready.</p>
            <h2>Smart Contract Authentication</h2>
            <p>This particular box comes with autentication via a smart contract
built-in.</p>
            <p>In the upper-right corner, you'll see a login button. Click it to
login with with the Authentication smart contract. If there is no user information
for the given address, you'll be redirected to sign up. There are two
authenticated routes: "/dashboard", which displays the user's name once
authenticated; and "/profile", which allows a user to update their name.</p>
            <h3>Redirect Path</h3>
            <p>This example redirects home ("/") when trying to access an
authenticated route without first authenticating. You can change this path in the
failureRedriectUrl property of the UserIsAuthenticated wrapper on <strong>line
9</strong> of util/wrappers.js.</p>
            <h3>Accessing User Data</h3>
            <p>Once authenticated, any component can access the user's data by
assigning the authData object to a component's props.<br/><code>{"// In
component's render function."}<br/>{"const { authData } =
this.props"}<br/><br/>{"// Use in component."}<br/>{"Hello {
this.props.authData.name }!"}</code></p>
            <h3>Further Reading</h3>
            <p>The React/Redux portions of the authentication fuctionality are
provided by <a href="https://github.com/mjrussell/redux-auth-wrapper"
target="_blank">mjrussell/redux-auth-wrapper</a>.</p>
          </div>
        </div>
      </main>
    )
  }
}

export default Home
```

File : Payment.js

```
import React, { Component } from 'react'

class Payment extends Component {
  constructor(props) {
    super(props)
  }

  render() {
    return(
      <main className="container">
        <div className="pure-g">
          <div className="pure-u-1-1">
            <h1>Dashboard</h1>
            <p><strong>Congratulations.</strong> If you're seeing this page,
you've logged in with your own smart contract successfully.</p>
          </div>
        </div>
      </main>
    )
  }
}

export default Payment
```

File : ConfirmPassengersMet.js

```
import React, { Component } from 'react'

class ConfirmPassengersMet extends Component {
  constructor(props) {
    super(props)

    this.state = {
      gps_location: '',
      passengers: ''
    }
  }
```

```
  onGpsLocationChange(event) {
    this.setState({ gps_location: event.target.value })
  }

  onPassengersChange(event) {
    this.setState({ passengers: event.target.value })
  }

  handleSubmit(event) {
    event.preventDefault()

    this.props.onConfirmPassengersMetFormSubmit(this.props.ride_number,
this.state.passengers, this.state.gps_location)
  }

  render() {
    return(
      <form className="pure-form pure-form-stacked"
onSubmit={this.handleSubmit.bind(this)}>
        <fieldset>
          <label htmlFor="name">GPS Location</label>
          <input id="gps_location" type="text" value={this.state.gps_location}
onChange={this.onGpsLocationChange.bind(this)} placeholder="GPS Location" />

          <label htmlFor="name">Passengers</label>
          <textarea id="passengers" type="text" value={this.state.passengers}
onChange={this.onPassengersChange.bind(this)} placeholder="Passengers" />

          <br />

          <button type="submit" className="pure-button pure-button-
primary">Confirm Passengers Met</button>
        </fieldset>
      </form>
    )
  }
}

export default ConfirmPassengersMet
```

File : ConfirmPassengersMetActions.js

```
import RideshareContract from '../../../../build/contracts/Rideshare.json'
// import { loginUser } from '../loginbutton/LoginButtonActions'
import { browserHistory } from 'react-router'
import store from '../../../store'

const contract = require('truffle-contract')

export function confirmPassengersMet(ride_number, passengers, gps_location) {
  let web3 = store.getState().web3.web3Instance

  // Double-check web3's status.
  if (typeof web3 !== 'undefined') {

    return function(dispatch) {
      // Using truffle-contract we create the authentication object.
      const rideshare = contract(RideshareContract)
      rideshare.setProvider(web3.currentProvider)

      // Declaring this for later so we can chain functions on Authentication.
      var rideshareInstance

      // Get current ethereum wallet.
      web3.eth.getCoinbase((error, coinbase) => {
        // Log errors, if any.
        if (error) {
          console.error(error);
        }

        rideshare.deployed().then(function(instance) {
          rideshareInstance = instance

          let new_passengers = passengers.split(",");

          console.log(new_passengers);

          // console.log('shipping cost');
          // console.log(shipping_cost);
          // console.log(shipping_cost * 10^18);
          // console.log(parseInt(shipping_cost * Math.pow(10,18)))

          // Attempt to sign up user.
          rideshareInstance.confirmPassengersMet(ride_number, new_passengers,
{from: coinbase})
          .then(function(result) {
            // If no error, login user.
            return browserHistory.push('/dashboard')
```

```
          })
          .catch(function(result) {
            // If error...
          })
          rideshareInstance.checkBalance().then(function(result){
            console.log("current Balance: "+result);
          })
        })
      })
    }
  } else {
    console.error('Web3 is not initialized.');
  }
}
```

File : ConfirmPassengersMetContainer.js

```
import { connect } from 'react-redux'
import ConfirmPassengersMet from './ConfirmPassengersMet'
import { confirmPassengersMet } from './ConfirmPassengersMetActions'

const mapStateToProps = (state, ownProps) => {
  return {}
}

const mapDispatchToProps = (dispatch) => {
  return {
    onConfirmPassengersMetFormSubmit: (ride_number, passengers, gps_location) => {
      dispatch(confirmPassengersMet(ride_number, passengers, gps_location))
    }
  }
}

const ConfirmPassengersMetContainer = connect(
  mapStateToProps,
  mapDispatchToProps
)(ConfirmPassengersMet)

export default ConfirmPassengersMetContainer
```

File : CreateRide.js

```
import React, { Component } from 'react'

class CreateRide extends Component {
  constructor(props) {
    super(props)

    this.state = {
      expected_payment: '',
      capacity: 0,
      origin_address: '',
      destination_address: '',
      confirmed_at: 0,
      depart_at: 0
    }
  }

  onExpectedPaymentChange(event) {
    this.setState({ expected_payment: event.target.value })
  }

  onCapacityChange(event) {
    this.setState({ capacity: event.target.value })
  }

  onOriginAddressChange(event) {
    this.setState({ origin_address: event.target.value })
  }

  onDestinationAddressChange(event) {
    this.setState({ destination_address: event.target.value })
  }

  onConfirmedAtChange(event) {
    this.setState({ confirmed_at: event.target.value })
  }

  onDepartAtChange(event) {
    this.setState({ depart_at: event.target.value })
  }
```

```
  handleSubmit(event) {
    event.preventDefault()

    this.props.onCreateRideFormSubmit(this.state.expected_payment,
this.state.capacity, this.state.origin_address, this.state.destination_address,
this.state.confirmed_at, this.state.depart_at)
  }

  render() {
    return(
      <form className="pure-form pure-form-stacked"
onSubmit={this.handleSubmit.bind(this)}>
        <fieldset>
          <label htmlFor="name">Expected Payment</label>
          <input id="expected_payment" type="text"
value={this.state.expected_payment}
onChange={this.onExpectedPaymentChange.bind(this)} placeholder="Expected Payment"
/>

          <label htmlFor="name">Capacity</label>
          <input id="capacity" type="text" value={this.state.capacity}
onChange={this.onCapacityChange.bind(this)} placeholder="Capacity" />

          <label htmlFor="name">Origin Address</label>
          <input id="origin_address" type="text" value={this.state.origin_address}
onChange={this.onOriginAddressChange.bind(this)} placeholder="Origin Address" />

          <label htmlFor="name">Destination Address</label>
          <input id="destination_address" type="text"
value={this.state.destination_address}
onChange={this.onDestinationAddressChange.bind(this)} placeholder="Destination
Address" />

          <label htmlFor="name">Needs to be confirmed by:</label>
          <input id="confirmed_at" type="text" value={this.state.confirmed_at}
onChange={this.onConfirmedAtChange.bind(this)} placeholder="Confirmed At" />

          <label htmlFor="name">Depart at:</label>
          <input id="depart_at" type="text" value={this.state.depart_at}
onChange={this.onDepartAtChange.bind(this)} placeholder="Depart At" />

          <br />

          <button type="submit" className="pure-button pure-button-primary">Create
Rideshare</button>
        </fieldset>
```

```
      </form>
    )
  }
}

export default CreateRide
```

File : CreateRideActions.js

```
import RideshareContract from '../../../../build/contracts/Rideshare.json'
// import { loginUser } from '../loginbutton/LoginButtonActions'
import { browserHistory } from 'react-router'
import store from '../../../store'

const contract = require('truffle-contract')

export function createRide(expected_payment, capacity, origin_address,
destination_address, confirmed_at, depart_at) {
  let web3 = store.getState().web3.web3Instance

  // Double-check web3's status.
  if (typeof web3 !== 'undefined') {

    return function(dispatch) {
      // Using truffle-contract we create the authentication object.
      const rideshare = contract(RideshareContract)
      rideshare.setProvider(web3.currentProvider)

      // Declaring this for later so we can chain functions on Authentication.
      var rideshareInstance

      // Get current ethereum wallet.
      web3.eth.getCoinbase((error, coinbase) => {
        // Log errors, if any.
        if (error) {
          console.error(error);
        }
```

```
        rideshare.deployed().then(function(instance) {
          rideshareInstance = instance

          // console.log('shipping cost');
          // console.log(shipping_cost);
          // console.log(shipping_cost * 10^18);
          // console.log(parseInt(shipping_cost * Math.pow(10,18)))
          console.log(expected_payment);
          console.log(parseInt(expected_payment * Math.pow(10,18)));

          // Attempt to sign up user.
          rideshareInstance.createRide(parseInt(expected_payment *
Math.pow(10,18)), capacity, origin_address, destination_address, confirmed_at,
depart_at, {from: coinbase})
          .then(function(result) {
            // If no error, login user.
            return browserHistory.push('/dashboard')
          })
          .catch(function(result) {
            // If error...
          })
        })
      })
    }
  } else {
    console.error('Web3 is not initialized.');
  }
}
```

File : CreateRideContainer.js

```
import { connect } from 'react-redux'
import CreateRide from './CreateRide'
import { createRide } from './CreateRideActions'

const mapStateToProps = (state, ownProps) => {
  return {}
}

const mapDispatchToProps = (dispatch) => {
  return {
```

```
    onCreateRideFormSubmit: (expected_payment, capacity, origin_address,
destination_address, confirmed_at, depart_at) => {
      dispatch(createRide(expected_payment, capacity, origin_address,
destination_address, confirmed_at, depart_at))
    }
  }
}

const CreateRideContainer = connect(
  mapStateToProps,
  mapDispatchToProps
)(CreateRide)

export default CreateRideContainer
```

File : joinRide.js

```
import React, { Component } from 'react'

class JoinRide extends Component {
  constructor(props) {
    super(props)
  }

  handleSubmit(event) {
    this.props.onJoinRideFormSubmit(this.props.ride_number, this.props.payment);
  }

  render() {
    return(
      <button onClick={this.handleSubmit.bind(this)}>Join Ride</button>
    )
  }
}

export default JoinRide
```

File : JoinRideActions.js

```
import RideshareContract from '../../../../build/contracts/Rideshare.json'
// import { loginUser } from '../loginbutton/LoginButtonActions'
import { browserHistory } from 'react-router'
import store from '../../../store'

const contract = require('truffle-contract')

export function joinRide(ride_number, payment) {
  let web3 = store.getState().web3.web3Instance

  // Double-check web3's status.
  if (typeof web3 !== 'undefined') {

    return function(dispatch) {
      // Using truffle-contract we create the authentication object.
      const rideshare = contract(RideshareContract)
      rideshare.setProvider(web3.currentProvider)

      // Declaring this for later so we can chain functions on Authentication.
      var rideshareInstance

      // Get current ethereum wallet.
      web3.eth.getCoinbase((error, coinbase) => {
        // Log errors, if any.
        if (error) {
          console.error(error);
        }

        rideshare.deployed().then(function(instance) {
          rideshareInstance = instance

          // console.log('shipping cost');
          // console.log(shipping_cost);
          // console.log(shipping_cost * 10^18);
          // console.log(parseInt(shipping_cost * Math.pow(10,18)))

          console.log(payment);

          // Attempt to sign up user.
          rideshareInstance.joinRide(ride_number, {value: parseInt(payment *
Math.pow(10,18)), from: coinbase})
```

```
          .then(function(result) {
            // If no error, login user.
            return browserHistory.push('/dashboard')
          })
          .catch(function(result) {
            // If error...
          })
        })
      })
    }
  } else {
    console.error('Web3 is not initialized.');
  }
}
```

File : JoinRideContainer.js

```
import { connect } from 'react-redux'
import JoinRide from './JoinRide'
import { joinRide } from './JoinRideActions'

const mapStateToProps = (state, ownProps) => {
  return {}
}

const mapDispatchToProps = (dispatch) => {
  return {
    onJoinRideFormSubmit: (ride_number, payment) => {
      dispatch(joinRide(ride_number, payment))
    }
  }
}

const JoinRideContainer = connect(
  mapStateToProps,
  mapDispatchToProps
)(JoinRide)

export default JoinRideContainer
```

File : userReducer.js

```
const initialState = {
  data: null
}

const userReducer = (state = initialState, action) => {
  if (action.type === 'USER_LOGGED_IN' || action.type === 'USER_UPDATED')
  {
    return Object.assign({}, state, {
      data: action.payload
    })
  }

  if (action.type === 'USER_LOGGED_OUT')
  {
    return Object.assign({}, state, {
      data: null
    })
  }

  return state
}

export default userReducer
```

File : getWeb3.js

```
import store from '../../store'
import Web3 from 'web3'

export const WEB3_INITIALIZED = 'WEB3_INITIALIZED'
function web3Initialized(results) {
  return {
    type: WEB3_INITIALIZED,
    payload: results
  }
}

let getWeb3 = new Promise(function(resolve, reject) {
```

```
  // Wait for loading completion to avoid race conditions with web3 injection
timing.
  window.addEventListener('load', function(dispatch) {
    var results
    var web3 = window.web3

    // Checking if Web3 has been injected by the browser (Mist/MetaMask)
    if (typeof web3 !== 'undefined') {
      // Use Mist/MetaMask's provider.
      web3 = new Web3(web3.currentProvider)

      results = {
        web3Instance: web3
      }

      console.log('Injected web3 detected.');

      resolve(store.dispatch(web3Initialized(results)))
    } else {

      // Fallback to localhost if no web3 injection. We've configured this to
      // use the development console's port by default.
      var provider = new Web3.providers.HttpProvider('http://127.0.0.1:9545')

      web3 = new Web3(provider)

      results = {
        web3Instance: web3
      }

      console.log('No web3 instance injected, using Local web3.');

      resolve(store.dispatch(web3Initialized(results)))
    }
  })
})

export default getWeb3
```

File : web3Reducer.js

```
const initialState = {
  web3Instance: null
}
```

```
const web3Reducer = (state = initialState, action) => {
  if (action.type === 'WEB3_INITIALIZED')
  {
    return Object.assign({}, state, {
      web3Instance: action.payload.web3Instance
    })
  }

  return state
}

export default web3Reducer
```

File : wrapper.js

```
import { UserAuthWrapper } from 'redux-auth-wrapper'
import { routerActions } from 'react-router-redux'

// Layout Component Wrappers

export const UserIsAuthenticated = UserAuthWrapper({
  authSelector: state => state.user.data,
  redirectAction: routerActions.replace,
  failureRedirectPath: '/', // '/login' by default.
  wrapperDisplayName: 'UserIsAuthenticated'
})

export const UserIsNotAuthenticated = UserAuthWrapper({
  authSelector: state => state.user,
  redirectAction: routerActions.replace,
  failureRedirectPath: (state, ownProps) => ownProps.location.query.redirect ||
'/dashboard',
  wrapperDisplayName: 'UserIsNotAuthenticated',
  predicate: user => user.data === null,
  allowRedirectBack: false
})

// UI Component Wrappers

export const VisibleOnlyAuth = UserAuthWrapper({
  authSelector: state => state.user,
  wrapperDisplayName: 'VisibleOnlyAuth',
  predicate: user => user.data,
  FailureComponent: null
```

```
})

export const HiddenOnlyAuth = UserAuthWrapper({
  authSelector: state => state.user,
  wrapperDisplayName: 'HiddenOnlyAuth',
  predicate: user => user.data === null,
  FailureComponent: null
})
```

File : App.js

```
import React, { Component } from 'react'
import { Link } from 'react-router'
import { HiddenOnlyAuth, VisibleOnlyAuth } from './util/wrappers.js'

// UI Components
import LoginButtonContainer from './user/ui/loginbutton/LoginButtonContainer'
import LogoutButtonContainer from './user/ui/logoutbutton/LogoutButtonContainer'

// Styles
import './css/oswald.css'
import './css/open-sans.css'
import './css/pure-min.css'
import './App.css'

class App extends Component {
  render() {
    const OnlyAuthLinks = VisibleOnlyAuth(() =>
      <span>
        <li className="pure-menu-item">
          <Link to="/dashboard" className="pure-menu-link">Dashboard</Link>
        </li>
        <li className="pure-menu-item">
          <Link to="/profile" className="pure-menu-link">Profile</Link>
        </li>
        <li className="pure-menu-item">
          <Link to="/driver" className="pure-menu-link">Drive</Link>
        </li>
        <LogoutButtonContainer />
      </span>
    )
```

```
    const OnlyGuestLinks = HiddenOnlyAuth(() =>
      <span>
        <li className="pure-menu-item">
          <Link to="/signup" className="pure-menu-link">Sign Up</Link>
        </li>
        <LoginButtonContainer />
      </span>
    )

    return (
      <div className="App">
        <nav className="navbar pure-menu pure-menu-horizontal">
          <ul className="pure-menu-list navbar-right">
            <OnlyGuestLinks />
            <OnlyAuthLinks />
          </ul>
          <Link to="/" className="pure-menu-heading pure-menu-link">Truffle
Box</Link>
        </nav>

        {this.props.children}
      </div>
    );
  }
}

export default App
```

File : index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Router, Route, IndexRoute, browserHistory } from 'react-router'
import { Provider } from 'react-redux'
import { syncHistoryWithStore } from 'react-router-redux'
import { UserIsAuthenticated, UserIsNotAuthenticated } from './util/wrappers.js'
import getWeb3 from './util/web3/getWeb3'

// Layouts
import App from './App'
import Home from './layouts/home/Home'
import Dashboard from './layouts/dashboard/Dashboard'
```

```
import Landing from './layouts/landing/Landing'
import Payment from './layouts/payment/Payment'
import Driver from './layouts/driver/Driver'
import Details from './layouts/details/Details'
import SignUp from './user/layouts/signup/SignUp'
import Profile from './user/layouts/profile/Profile'

// Redux Store
import store from './store'

// Initialize react-router-redux.
const history = syncHistoryWithStore(browserHistory, store)

// Initialize web3 and set in Redux.
getWeb3
.then(results => {
  console.log('Web3 initialized!')
})
.catch(() => {
  console.log('Error in web3 initialization.')
})

ReactDOM.render((
    <Provider store={store}>
      <Router history={history}>
        <Route path="/" component={App}>
          <IndexRoute component={Landing} />
          <Route path="dashboard" component={UserIsAuthenticated(Dashboard)} />
          <Route path="signup" component={UserIsNotAuthenticated(SignUp)} />
          <Route path="profile" component={UserIsAuthenticated(Profile)} />
          <Route path="home" component={UserIsAuthenticated(Home)} />
          <Route path="payment" component={UserIsAuthenticated(Payment)} />
          <Route path="driver" component={UserIsAuthenticated(Driver)} />
          <Route path="/details/:id" component={UserIsAuthenticated(Details)}/>
        </Route>
      </Router>
    </Provider>
  ),
  document.getElementById('root')
)
```

CHAPTER-4

Results and Discussions

The primary purpose of this project is to offer cab rental services to customers offering immense security without the need to be dependent on third party systems, no governing authority is required to administer and look over the transacting parties and the system is particularly transparent. This system is crafted on Ethereum Blockchain. Ethereum is a blockchain platform with its own cryptocurrency, called Ether (ETH) or Ethereum, and its own programming language, called Solidity. Ethereum is a decentralized public ledger for verifying and recording transactions. The network's users can create, publish, monetize, and use applications on the platform, and use its Ether cryptocurrency as payment. This project can be deployed on Polygon Network for now. Polygon is a protocol and a framework for building and connecting Ethereum-compatible blockchain networks. Aggregating scalable solutions on Ethereum supporting a multi-chain Ethereum ecosystem.

CHAPTER-5

Conclusion and Future Scope

Technologies such as blockchain have disrupted the entire fintech market and even though it has created a lot of controversies, the technology is going to get more and more integrated into our lives. When it comes to the matter of tracking financial properties, Blockchain technology has kept its promise as well as has shown consistency. Several financial institutions have invested in this technology after recognizing its potential and beneficial impacts. Because of its transparent ledger system, Blockchain can tackle the flow and dealings of black money flow. Governments are considering it as an option to have more efficient regulations over the countries' economies.

The current system that has been drafted takes in Longitude and Latitudes to track the Pick-up and Drop Locations, in future the **integration of Google Maps** can be done. This would surely be an added advantage, enhancing the user experience and usability.

As of now, FOLA is built on Ethereum Blockchain written in Solidity. To address the issue of Scalability we are deploying the application on Polygon Network but in future, we'll be migrating our codebase to **Rust** so that we can deploy it on **Solana Network.** It is a high performance blockchain that uses a proof of stake consensus mechanism. It has a low barrier to entry along with timestamped transactions to maximize efficiency.