A
PRELIMINARY REPORT
ON

PEER DRIVEN CARPOOL SERVICE WITH BLOCKCHAIN

Submitted to the
Savitribai Phule Pune University, Pune
In partial fulfillment of the Requirements for the award of the Degree of
BACHELOR OF ENGINEERING (COMPUTER ENGINEERING)

SUBMITTED BY

Lokhande Gauri	72154781J
Nagare Manasvi	72154792D
Narkhede Rasika	72154794L
Nimkar Krupa	72154798C

Department of Computer Engineering

GOKHALE EDUCATION SOCIETY'S
R H SAPAT COLLEGE OF ENGINEERING, MANAGEMENT, STUDIES AND RESEARCH, NASHIK - 422005
SAVITRIBAI PHULE PUNE UNIVERSITY

(2023-24)

CERTIFICATE

This is to certify that the seminar report entitled
"PEER-DRIVEN CARPOOL SERVICE WITH BLOCKCHAIN"
Submitted by

Lokhande Gauri	72154781J
Nagare Manasvi	72154792D
Narkhede Rasika	72154794L
Nimkar Krupa	72154798C

is a bonafide student of this institute and the work has been carried out by him/her under the supervision of Dr.S.R.Lahane sir and it is approved for the partial fulfillment of the requirement of Savitribai Phule Pune University, for the award of the degree of Bachelor of Engineering (Computer Engineering).

Dr. S.R.Lahane
Project Guide
Department of Computer Engineering

Dr. D.V.Patil
HOD
Department of Computer Engineering

External Examiner

Dr.P.C.Kulkarni
Principal,
Gokhale Edu. Society's
R H Sapat COEMSR, Nashik

Place:
Date:

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to all those who made the completion of this group project possible. Without their support, guidance, and assistance, this endeavor would not have been successful.

First and foremost, we would like to thank our mentor Dr. S. R. Lahane for their invaluable guidance throughout the project. Their expertise and feedback were instrumental in shaping our work and ensuring its quality. We are also grateful to our fellow team members for their dedication, hard work, and collaboration. Each member played a crucial role in their respective tasks, and together we achieved our project goals.

Additionally, we extend our thanks to the Computer Department of our College for providing us with access to resources and facilities that were essential to our research and data collection. Furthermore, we appreciate the support from our friends and family for their patience, encouragement, and understanding during the demanding periods of this project.

Last but not least, we want to express our gratitude to the academic community and all the authors whose work we consulted for our literature review. Their research and publications were invaluable in shaping our understanding of the subject matter.

In conclusion, we are thankful to everyone who played a part in this project, directly or indirectly. Your support and contributions have been instrumental, and we couldn't have done it without you. Thank you all for being a part of this journey.

Mr. R.S.Jagale
Project Coordinator

Dr. D.V.Patil
HOD

ABSTRACT

The rapid urbanization and increasing congestion on roadways have necessitated the development of innovative solutions to address transportation challenges. This project proposes the implementation of a Carpool Service with Blockchain technology to enhance the efficiency, transparency, and security of carpooling systems.

The proposed system leverages the decentralized and tamper-resistant nature of blockchain to establish a trustable and transparent platform for coordinating and managing carpooling services. Smart contracts, self-executing contracts with the terms of the agreement directly written into code, will be employed to automate key processes, such as ride matching, payment transactions, and reputation scoring. Participants in the carpool service, including drivers and passengers, will benefit from enhanced security and privacy through the use of cryptographic techniques. The blockchain ledger will provide an immutable record of all transactions and interactions, fostering accountability and reducing the risk of fraud or disputes. Furthermore, the integration of a cryptocurrency-based payment system will streamline financial transactions, eliminating the need for traditional payment methods and reducing transaction costs.

The project aims to create a seamless and cost-effective carpooling experience for users, encouraging the adoption of sustainable and shared transportation practices. By combining the efficiency of blockchain technology with the collaborative nature of carpooling, this project strives to contribute to the creation of a more sustainable, affordable, and reliable transportation solution for urban communities. The implementation of a Carpool Service with Blockchain has the potential to revolutionize the way individuals share rides, fostering a more connected and eco-friendly future. In summary, the "Carpool Service with Blockchain" project is a comprehensive initiative that extends beyond the basics, incorporating advanced features to create a secure, efficient, and user-centric carpooling solution. Through the integration of these elements, the project aims to redefine urban mobility, promoting sustainable and collaborative transportation practices.

Contents

List of Figures

Chapter 1

PEER DRIVEN CARPOOL SERVICE WITH BLOCKCHAIN

1.1 INTRODUCTION

The ultra-fast moving cars required in today's hectic world are a result of the quickly expanding advancements in vehicular technologies. Every identity should be able to travel with ease and confidence. The trace system must be extremely efficient with time. Every one of these needs converges at the rate of the car that is directly correlated with traffic congestion on the road. Research has shown that during the past few years, there has been a sharp growth in the number of vehicles per person worldwide. Causing traffic to move more slowly as a result of the limited road capacity.

It is hard to achieve fairness in the currently running systems and also there exists no common platform in this regard where the rider can go for a comparison and select the driver accordingly. Also, the fairness in the procedure is trusted to be implemented by the underlying service providers, which is not verifiable from the rider end. This kind of trust based system creates a point of dissatisfaction as the inner computations strategies are not clearly known to them. Hence, the car sharing to be widely appreciated among all the class of people, we have appreciated the role of decentralized peer to peer network of blockchain Block as the backbone of the architecture.

1.2 MOTIVATION

The motivation behind Block is firstly to ensure the payment fairness where the breakup of the fare i,e. the cost of the ride for a particular path is computable by any peer of the network with the path details. Secondly, we present the ride fairness where in the case of any dispute, addressed by the rider, the malicious driver or the malicious rider (in case of false allegation) will be penalized Block collaborates with the Road Side Units (RSUs) to achieve fairness in this respect. We believe that our work will be useful to ride sharing services like BlaBlaCar, Ola and Uber.

1.3 OBJECTIVES OF WORK

The persons who are using their private car will switch to the car hiring methodologies, only if they and the procedures are simple, explainable, cost efficient and fair in all respect. Here, we have presented Block, an architecture which follows these four principles and reflects a robust end to end solution. The procedure of hiring a car involves a negotiation for a fare against a ride chosen by the rider, a fair payment mechanism explainable to all and a decentralized system that ensures fair, trusted, cost effective transactions. The system which is being followed currently is the application based car sharing, where there exist many centralized servers monitoring every aspects of the ride mentioned above in different platforms of applications.

Chapter 2

RELATED WORK

2.1 BLOCKCHAIN

At the moment, blockchain applications are being concentrated on multiple fronts. The application in the finance sector, where decentralized cryptocurrency is the major draw, is the most fundamental and straightforward sort. One well-known cryptocurrency that has spurred a revival in the finance industry is Bitcoin. The most well praised blockchain development to date was the release of the Bitcoin whitepaper in 2008. We may now transact with electronically coded addresses using Litecoin, Namecoin, Peercoin, and Dogecoin in addition to Bitcoin. In a traditional system, trust in a transaction is ensured by a middleman such as a banker or remittance business.

Supply chain management, healthcare, Internet of Things (IoT) with optimized blockchain, data sharing for medical in cloud environments, and personal data sharing are just a few of the applications that have benefited greatly from block chain technology. using apps like file sharing using Filecoin, digital certificates, music and other content, and more, it has completely changed the sharing economy. Transferring confidence from centralized service providers to peers who maintain the blockchain is the fundamental goal of using it. The system is safe as long as most of the peers are trustworthy. The blockchain's achievement of system fairness is demonstrated in.

2.2 ETHEREUM

Anyone can create and use decentralized blockchain applications using Ethereum, an open programmable blockchain platform. It is an international collaborative effort that is open-source. Ethereum is made to be flexible and adaptive. Developing new applications on the Ethereum platform is simple. Ethereum encompasses a peer-to-peer network protocol, just like any other blockchain. Numerous nodes linked to the network update and maintain the Ethereum blockchain database. The EVM is run on every single node in the network, carrying out identical instructions. There are two types of accounts:

1. Externally Owned Accounts (EOAs), which are controlled by private keys.
2. Contract Accounts, which are controlled by their contract code and can only be activated by an EOA

2.3 ETHEREUM VIRTUAL MACHINE

By offering a run-time environment for smart contracts, the Ethereum Virtual Machine (EVM) is a quasi-Turing complete machine that carries out the Ethereum blockchain's execution paradigm. A virtual byte-array RAM (ROM) serves as the memory representation, and smart contracts are used to construct the algorithm. One parameter that limits computations is termed gas. The gas cost is calculated for each action in Ethereum before a smart contract is carried out and paid for as those operations are carried out. If the software has a lower specified gas limit than necessary, several operations cannot be carried out.

Simple stack-based design characterizes the EVM. The language used for computation on the EVM is called stack-based bytecode. The machine's word size, or 32 bytes, is 256 bits, which is also the size of a stack item. Every item on the EVM stack has a 256-bit size. Leading zeros are used to pad data items that are smaller than 256 bits in size. The maximum size of the stack is 1024. The EVM uses a straightforward word-addressed byte array as its memory model. This indicates that memory is an array of bytes, with a memory address assigned to each byte. The EVM uses a word-addressable word array as its storage model.Storage is kept as a component of the system state and is non-volatile in contrast to memory, which is volatile. Every place in memory and storage is initially well-designed as zero.

2.4 SECURING THE ETHEREUM BLOCKCHAIN WITH EVM

1. The EVM imposes the following set of restrictions to secure the state of the system.
2. Every computational step taken in process of executing a program must be paid for upfront, thereby preventing Denial-of-Service (DoS) attacks.
3. Programs may only interact with each other by transmitting a single arbitrary length byte array but they do not have access to each others state.
4. An EVM program can only access and modify its own internal state and may trigger the execution of other EVM programs, but not allowed to do anything else.
5. Program execution is completely deterministic and produces identical state transitions for any similar type of implementation starting from an identical state

2.5 SMART CONTRACT

On a blockchain network, a smart contract is an electronic contract that takes effect automatically when certain requirements are met. The terms and conditions is written in blockchain-specific programming languages such as Solidity.One can also look at smart contracts as blockchain applications that enable all parties to carry out their part of a transaction. Apps powered by smart contracts are frequently referred to as "Decentralized Applications" or "DApps."

While the idea of blockchain is largely perceived as Bitcoin's underlying tech driver, it has, since then, grown into a force to reckon with. Using smart contracts, a manufacturer requiring raw materials can establish payments, and the supplier can schedule shipments. Then, based on the contract between the two organizations, payments can be automatically transferred to the seller upon dispatch or delivery.

2.6 TYPES OF SMART CONTRACT

When it comes to the types of smart contracts, they are classified into three categories - Legal contracts, Decentralized Autonomous Organizations or DAOs, and Logic contracts.

1. Smart legal contract: Smart contracts are guaranteed by law. They adhere to the structure of legal contracts: "If this happens, and then this will happen." As smart contracts reside on blockchain and are unchangeable, judicial or legal smart contracts offer greater transparency than traditional documents among contracting entities.

2. Decentralized autonomous organizations: DAOs are democratic groups governed by a smart contract that confers them with voting rights. A DAO serves as a blockchain-governed organization with a shared objective that is collectively controlled. No executive or president exists. Instead, blockchain-based tenets embedded within the contract's code regulate how the organization functions and funds are allocated.

3. Application logic contracts: ALCs, or application logic contracts, consist of application-based code that typically remains synced with various other blockchain contracts. It enables interactions between various devices, like the Internet of Things (IoT) or blockchain integration. Unlike the other types of smart contracts, these are not signed between humans or organizations but between machines and other contracts.

Chapter 3

LITERATURE SURVEY

1. Sarvesh Wadi, "P2P Ride-Sharing using Blockchain Technology", September 2022

   Ride-sharing is a service that allows drivers to share trips with other riders, which contributes to the enticing benefits of shared travel costs and traffic congestion. Peer-to-Peer ridesharing technology deploys blockchain to deal with the problems faced by traditional centralized ride-sharing applications. Blockchain is a blooming technology and decentralization of systems can be powered by it. Proposed ride sharing system will be deployed on blockchain and will be decentralized in a true sense. This application allows drivers to provide ride-sharing services without relying on a third party. This paper focuses on developing a peer-to-peer hasslefree, less error prone ride-sharing application powered by blockchain technology. The Android Application consists of multiple classes activities to support the functionality of the project as a whole.Most of the existing ride sharing applications like Uber, Ola deploy centralized 3rd party based systems to provide ride sharing services to customers. Login and Registration service is deployed with a simple User interface and google firebase authentication service is used at the backend to store and authorize login credentials of the users.

2. V. Buterin. Ethereum, April 2020, "A NEXT GENERATION SMART CONTRACT and DECENTRALIZED APPLICATION PLATFORM"

   The Ethereum protocol was originally conceived as an upgraded version of a cryptocurrency, providing advanced features such as on-blockchain escrow, withdrawal limits and financial contracts, gambling markets and the like via a highly generalized programming language. The Ethereum protocol would not "support" any of the applications directly, but the existence of a Turing-complete programming language means that arbitrary contracts can theoretically be created for any transaction type or application. What is more interesting about Ethereum, however, is that the Ethereum protocol moves far beyond just currency. Protocols and decentralized applications around decentralized file storage, decentralized computation and decentralized prediction markets, among dozens of other such concepts, have the potential to substantially increase the efficiency of the computational industry, and provide a massive boost to other peer-to-peer protocols by adding for the first time an economic layer. Finally, there is also a substantial array of applications that have nothing to do with money at all.

3. S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system", January 2019 SSRN Electronic Journal

A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

4. T. Report, "Filecoin: A cryptocurrency operated file storage network", Dec 2018

Filecoin is a distributed electronic currency similar to Bitcoin. Unlike Bitcoin's computation-only proof-of-work, Filecoin's proof-of-work function includes a proof-of-retrievability component, which requires nodes to prove they store a particular file. The Filecoin network forms an entirely distributed file storage system, whose nodes are incentivized to store as much of the entire network's data as they can. Like Bitcoin, Filecoin implements a transaction ledger via a blockchain, in which each block must be accompanied by a proof-of-work based on a cryptographic hash function. The proof-of-work parameter is dynamically adjusted so that one block occurs roughly every ten minutes, as in Bitcoin. Further, as in most blockchain-based constructions, clients should only consider transactions to be committed after a sequence of several valid blocks, where the number of blocks is determined by the assumed restrictions on the power of the adversary (e.g., that no adversary can control a majority of the network's computation power).

5. MovieBloc, "Decentralized Movie and Content Distribution ", White Paper-Ver. 1.14 / APR. 2019

This document is intended to convey specific information about the platform being planned and developed by the MovieBloc team. This document is for informational purposes only. Nothing herein should be construed as indicating the accuracy or reliability of the information contained herein. Although the information contained in this document has been taken from the source materials that the MovieBloc team believes to be reliable and believable, the MovieBloc team does not endorse the accuracy or suitability of such information. In other words, the MovieBloc team will not be held liable for any loss or damage caused by information associated with us or the MovieBloc platform. The information contained in this document is furnished only at the present time and is subject to change without notice, and the MovieBloc team is not obligated to revise, modify or update this document.

6. **Yong Yuan, Fei-Yue Wang, "Towards blockchain-based intelligent transportation systems" , 2016 IEEE 19 th International Conference on Intelligent Transportation Systems (ITSC)**

Blockchain, widely known as one of the disruptive technologies emerged in recent years, is experiencing rapid development and has the full potential of revolutionizing the increasingly centralized intelligent transportation systems (ITS) in applications. Blockchain can be utilized to establish a secured, trusted and decentralized autonomous ITS ecosystem, creating better usage of the legacy ITS infrastructure and resources, especially effective for crowd sourcing technology. This paper conducts a preliminary study of Blockchain-based ITS (B 2 ITS). We outline an ITS-oriented, seven-layer conceptual model for blockchain, and on this basis address the key research issues in B 2 ITS. We consider that blockchain is one of the secured and trusted architectures for building the newly developed parallel transportation management systems (PtMS) , and thereby discuss the relationship between B 2 ITS and PtMS. Finally, we present a case study for blockchain-based real time ride-sharing services. In our viewpoint, B 2 ITS represents the future trend of ITS research and practice, and this paper is aimed at stimulating further effort and providing helpful guidance and reference for future research works.

Chapter 4

EXISTING SYSTEM

4.1 INTRODUCTION

Existing System is an online marketplace for carpooling headquartered in Paris. Its website and mobile apps connect drivers and passengers willing to travel together between cities and share the cost of the journey, in exchange for a commission of between 18 percent and 21 percent. It also operates, an intercity bus service. The platform has 26 million active members and is available Europe and Latin America. There are many popular ridesharing platform that connects drivers with empty seats in their vehicles with passengers traveling in the same direction. It was founded in France in 2006 by Frédéric Mazzella, Francis Nappez, and Nicolas Brusson.

4.2 TECHNOLOGY USED

Existing System uses 24 technology products and services including HTML5 ,Google Analytics and Google Fonts, according to G2 Stack. There are actively using 81 technologies for its website, according to BuiltWith. These include Viewport Meta ,IPhone / Mobile Compatible and SPF.

1. **Programming Languages: The success behind the best driving direction apps is the right programming language. If you want your carpooling app to be successful in no time you must choose programming languages such as:**
   (a) **Swift**
   (b) **React Native**
   (c) **Flutter**
   (d) **Java**
2. **Backend Framework: For backend framework, the first choice of Android app developers is Node. JS with Express.js, Ruby on Rails, Django, or Laravel for building server-side logic and APIs**
3. **Primary Database: If you want to make an app like BlaBlacar use databases like PostgreSQL, MySQL, or MongoDB for storing user data, ride information, preferences, and other app-related data.**

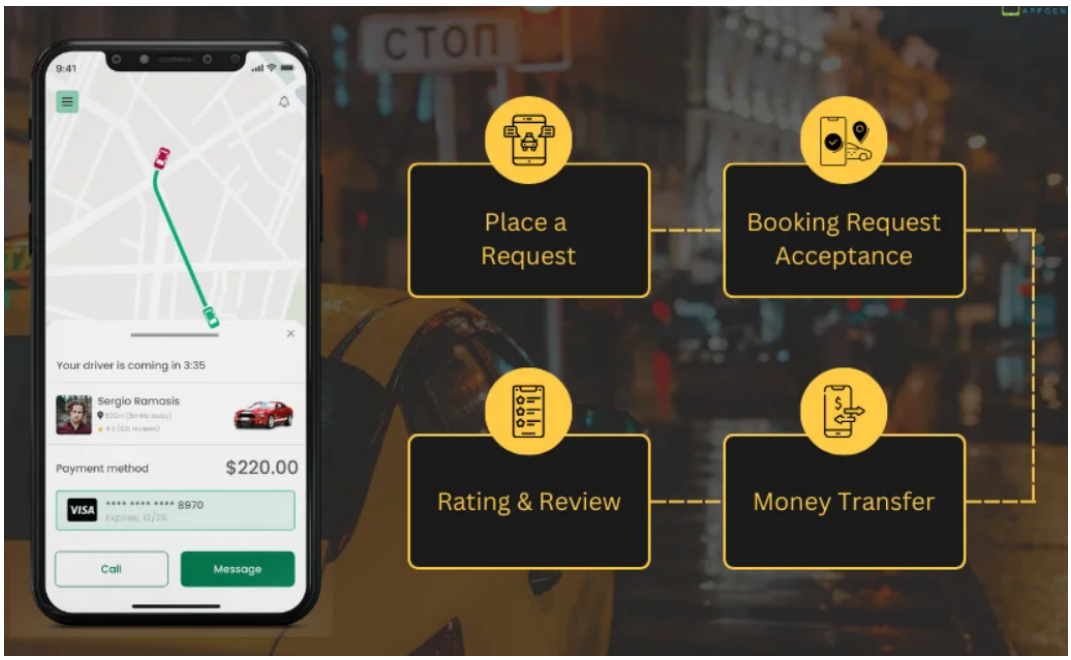


Figure 4.1: Existing System

Chapter 5

PROPOSED MODEL

The Related work described in Chapter 2 assumes the existence of Blockchain as the backbone of the model. For the purpose of security analysis of this architecture, we have assumed that blockchain is secured i,e. the consensus algorithms are working efficiently and the immutability properties hold. Given the blockchain is secured, the money locked in blockchain is secured hence the payment involved in the system is secured. The definition of the reputation score as a state variable in blockchain makes any changes in it noticeable by all the peers. Hence any unauthorized changes will be marked. Similarly we have defined the status of Rider and Driver as the state variable where the opposite party can check ones present status from blockchain. This makes the communications secured by choosing appropriate conditions for each procedure.

1. **Preposition 2: Block achieves the payment fairness and ride fairness assuming the blockchain, RSU and Route database RD are secure and the interaction with these entities are secure.**

2. **Proof:The sub-procedure Calculate FARE as discussed in Procedure 18 computes the linear path traveled and the corresponding fare by multiplying the cumulative path by a non negative fare component. The algorithm takes the route r which is a set of locations, forming the path the rider wants to travel. for each consecutive entries in r, DISTANCE calculates the euclidean distance. We have added a fixed component of fare with this to cater the expenses of cost of establishment, toll taxes, maintenance cost etc. Given a path, computation of fare is available to all the peers in the network. Hence, the fare can be verified by all. So there exists a linearly proportional component as well as fixed component in the fare distribution. This ensures payment fairness.**

   The scheme offers the rider to come up with an objection before triggering the Complete procedure. We have considered both the cases where the driver is the malicious and tries to follow a route beyond contract or a malicious rider tries to cheat the driver by accusing him with a false objection. Complain procedure as discussed in 7 handles these situations and penalizes the malicious party. Data (LT) as the proof of wrong route, supplied by the rider with the objection, is considered for the further analysis.

5.1 ER DIAGRAM OF CARPOOLING SYSTEM

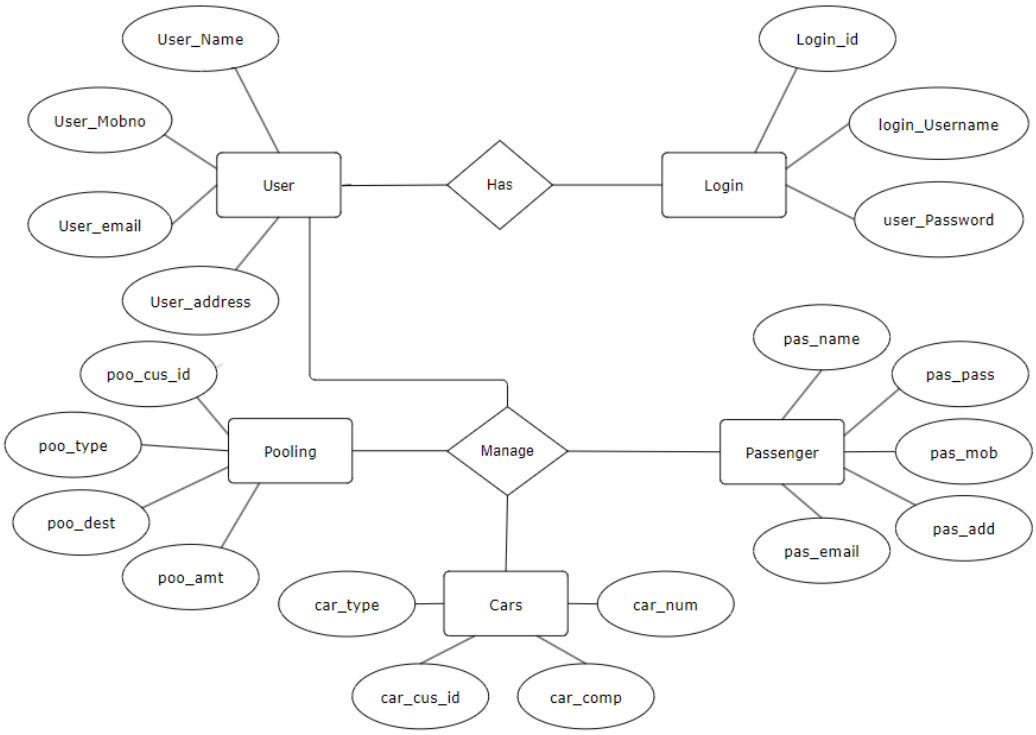


Figure 5.1: ER Diagram of Carpooling System

This ER (Entity Relationship) Diagram represents the model of Car Pooling System Entity. The entity relationship diagram of Car Pooling System shows all the visual instrument of database tables and the relation between Pooling, Car Routes, Cars, Drivers, etc. It used structure data and to define the relationships between structured data groups of Car Pooling System functionalities. The main entities of Car Pooling System are Cars, Pooling, Passenger, Car Routes, Bill and Drivers.

Car Pooling System Entities and Their Attributes:

1. **Cars Entity: Attributes of Cars are CarId, CarCustomerId, CarNumber, CarCompany, CarType.**
2. **Pooling Entity: Attributes of Pooling are PoolingId, PoolingCustomerId, PoolingDriverId, PoolingAmount, PoolingPayment, PoolingType.**
3. **Passenger Entity: Attributes of Passenger are PassengerId, PassengerName, PassengerMob, PassengerEmail, PassengerAddress.**
4. **Car Routes Entity: Attributes of Car Routes are CarRouteID, CarRouteName, CarRouteType, CarRouteDescription.**
5. **Bill Entity: Attributes of Bill are BillID, BillCustomerId, BillNumber, BillDescription.**

Description of Car Pooling System Database:

1. **The details of Cars is store into the Cars tables respective with all tables.**
2. **Each entity (Drivers, Passenger, Bill, Pooling, Cars) contains primary key and unique keys.**
3. **The entity Passenger, Bill has binded with Cars, pooling entities with foreign key.**
4. **There is one-to-one and one-to-many relationships available between Bill, Car Routes, Drivers, Cars.**
5. **All the entities Cars, Bill, Passenger, Drivers are normalized and reduced duplicacy of records**
6. **We have implemented indexing on each tables of Car Pooling System tables for fast query execution.**

5.2 UML DIAGRAM OF CARPOOLING SYSTEM

1. **Use Case Diagram of Car Pooling System:**

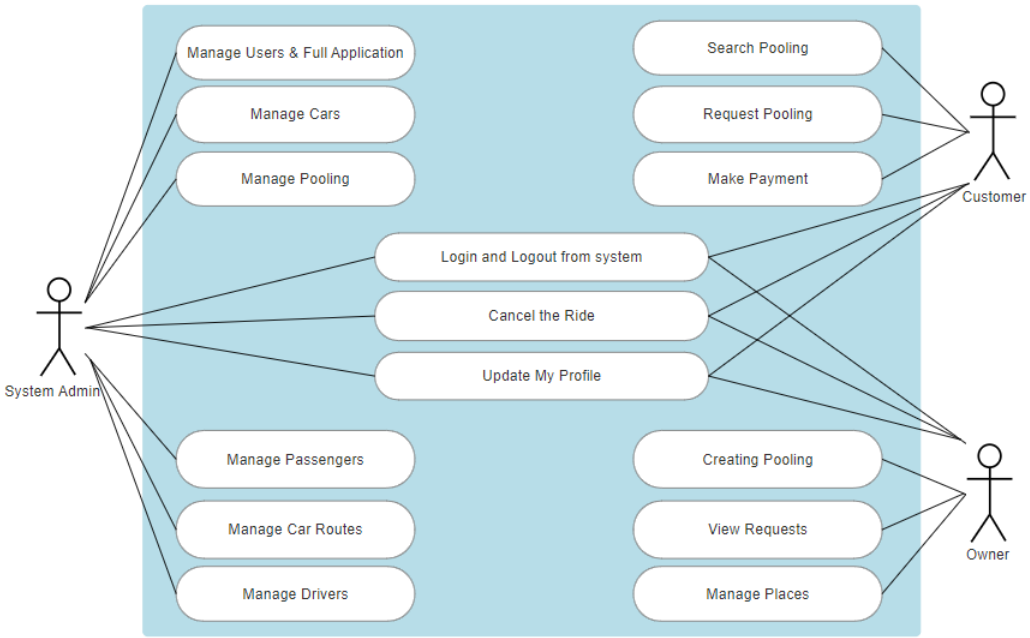


Figure 5.2: Use Case Diagram

This Use Case Diagram is a graphic depiction of the interactions among the elements of Car Pooling System. It represents the methodology used in system analysis to identify, clarify, and organize system requirements of Car Pooling System.

The main actors of Car Pooling System in this Use Case Diagram are: System Admin, Customer, Owner, who perform the different type of use cases such as Manage Cars, Manage Pooling, Manage Passenger, Manage Car Routes, Manage Drivers, Manage Users and Full Car Pooling System Operations. Major elements of the UML use case diagram of Car Pooling System are shown on the picture below.

The relationships between and among the actors and the use cases of Car Pooling System:

(a) **System Admin Entity: Use cases of Super Admin are Manage Cars, Manage Pooling, Manage Passenger, Manage Car Routes, Manage Drivers, Manage Users and Full Car Pooling System Operations.**

(b) **Customer Entity: Use cases of Customer are Search Pooling, Request Pooling, Make Payment**

(c) **Owner Entity: Use cases of Owner are Creating Pooling, View Requests, Manage Places**

2. Activity UML Diagram of Car Pooling System:

This is the Activity UML diagram of Car Pooling System which shows the flows between the activity of Drivers, Passengar, Car Routes, Cars, Pooling. The main activity involved in this UML Activity Diagram of Car Pooling System are as follows:

(a) **Drivers Activity**

(b) **Passengar Activity**

(c) **Car Routes Activity**

(d) **Cars Activity**

(e) **Pooling Activity**

Activity diagrams represent workflows in an graphical way. They can be used to describe business workflow or the operational workflow of any component in a system. Sometimes activity diagrams are used as an alternative to State machine diagrams. Check out this wiki article to learn about symbols and usage of activity diagrams.

Feature of the Activity UML Diagram of Carpooling System:

(a) **Admin User can search Drivers, view discription of selected drivers, add Drivers, update Drivers and delete Drivers.**

(b) **Its shows the activity flow of editing, adding and updating of Passenger.**

(c) **User will be able to search and generate report of Car Routes, Cars, Pooling**

(d) **All objects such as(Drivers, Passenger, Pooling) are interlinked.**

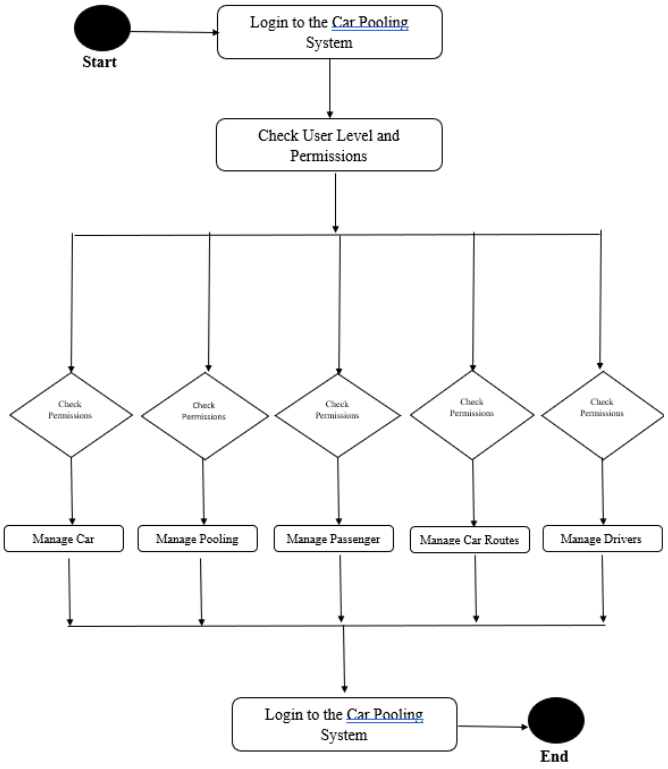


Figure 5.3: Activity UML Diagram

3. Class Diagram of the Carpooling System:

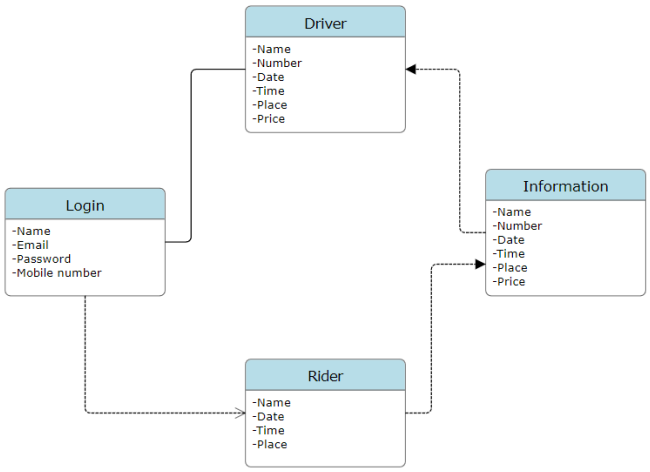


Figure 5.4: Class Diagram

The purpose of a class diagram is to depict the classes within a model. In an object oriented application, classes have attributes (member variables), operations (member functions) and relationships with other classes. The UML class diagram can depict all these things quite easily. The fundamental element of the class diagram is an icon the represents a class. This icon is shown in the figure 5.4.

4. **Component Diagram of the Carpooling System:**

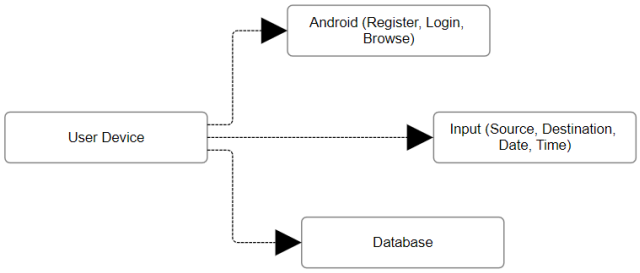


Figure 5.5: Component Diagram

A component diagram displays the structural relationship of components of a software system. These are mostly used when working with complex systems that has many components. Components communicate with each other using interfaces. The interfaces are linked using connectors. Figure 5.5 shows a component diagram.

5. **Deployment Diagram of the Carpooling System:**

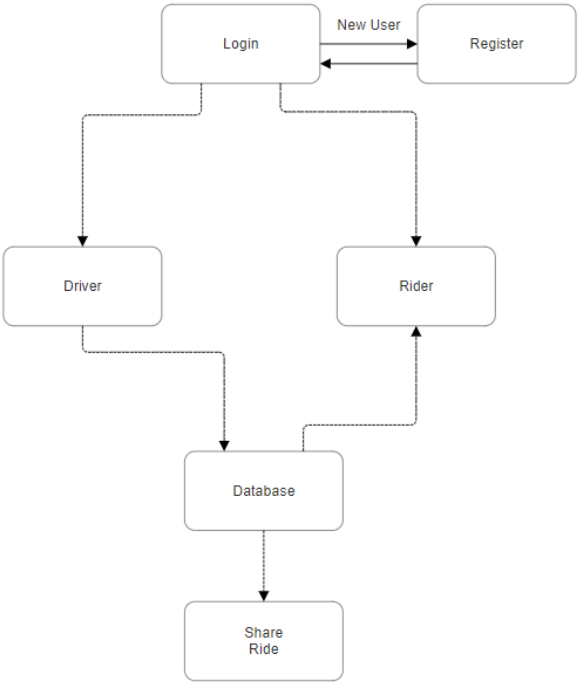


Figure 5.6: Deployment Diagram

A deployment diagrams shows the hardware of your system and the software in those hardware. Deployment diagrams are useful when your software solution is deployed across multiple machines with each having a unique configuration. Figure 5.6 is an example deployment diagram. UML Class Diagram with Relationships.

5.3 DFD DIAGRAM OF CARPOOLING SYSTEM

1. Zero Level DFD of Carpooling System:

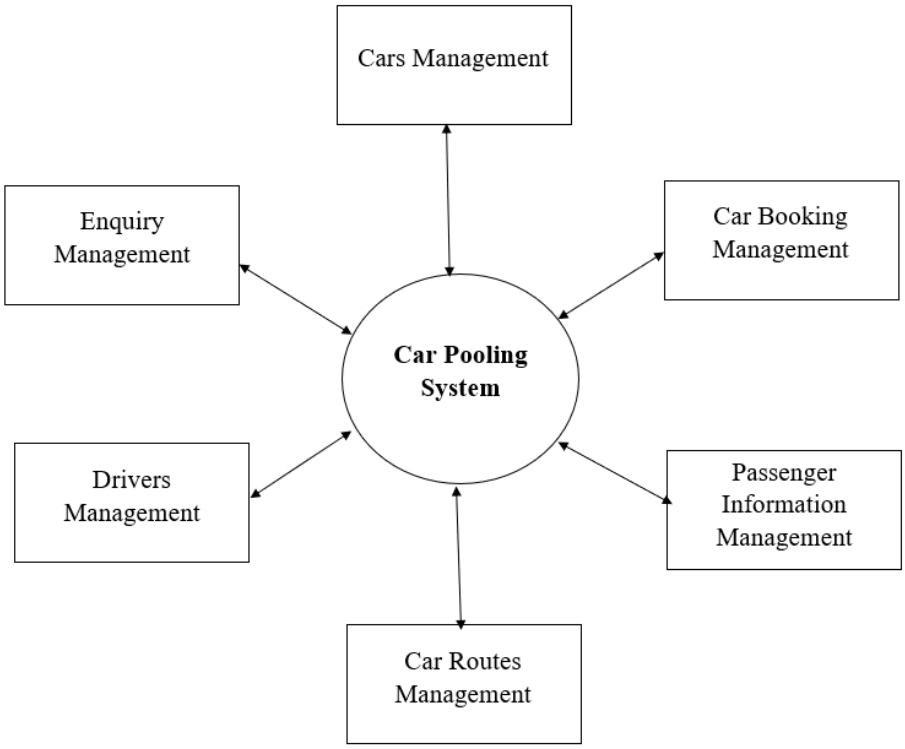


Figure 5.7: Zero level DFD Diagram

This is the Zero Level DFD of Car Pooling System, where we have elaborate the high level process of Car. It's a basic overview of the whole Car Pooling System or process being analyzed or modeled. It's designed to be an at-a-glance view of Drivers, Enquiry and Login showing the system as a single high-level process, with its relationship to external entities of Cars, Car booking and Passenger Information.

2. First Level DFD of Carpooling System:

First Level DFD (1st Level) of Car Pooling System shows how the system is divided into sub-systems (processes), each of which deals with one or more of the data flows to or from an external agent, and which together provide all of the functionality of the Car Pooling System as a whole.

3. Second Level DFD of Carpooling System:

DFD Level 2 then goes one step deeper into parts of level 1 of Car. It may require more functionalities of Car to reach the necessary level of detail about the Car functioning. First Level DFD (1st Level) of Car Pooling System shows how the system is divided into sub-systems (processes). The 2nd Level DFD contains more details of Login, Enquiry, Drivers, Car Routes, Passenger Information, Car Booking, Cars.

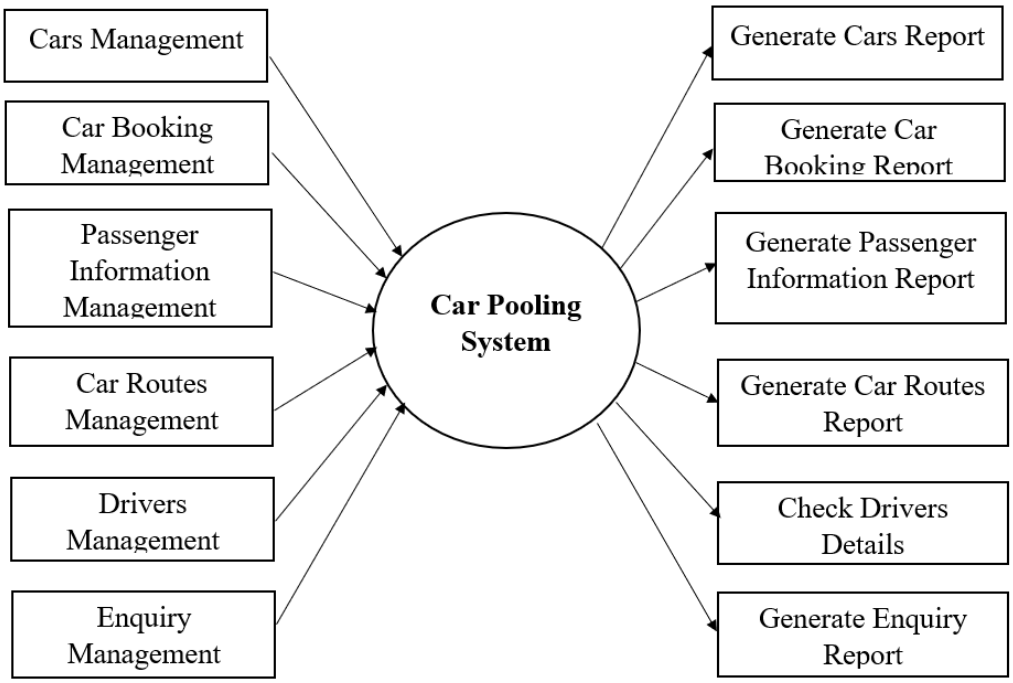


Figure 5.8: First level DFD Diagram

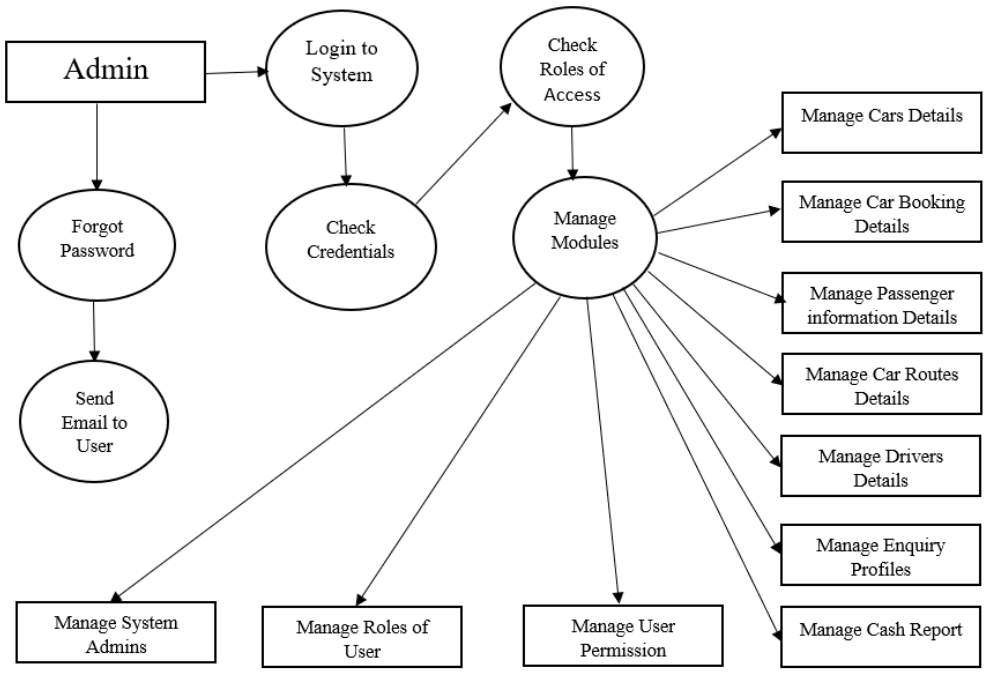


Figure 5.9: Second level DFD Diagram

Chapter 6

MATHEMATICAL MODEL

Mapping Function f(x)=y	x(Input)	y(Output)
f1(d)=f d -user data F -Home page	d	F
F2(F)=F'/R F'-Data record created	F	F'/R
F3(F')=dbs dbs- data in database	F'	dbs
F4(dbs,R)=m m = dbs(list of matching)	dbs,R	m
F5(m,Ip3)=F" F" = ML model processing fare recommendation	m,Ip3	F"
F6(F")=Su Su - Final available list	F"	Su

Mathematical model of our project describes complete working of the Rideshare app. Here f(x) denotes the function of x, where we have used f1, f2, f3, f4, f5 as functions with different inputs resulting in required outputs.

At the start of the app the user will be asked to login, this action is our function 1 'f1' with input 'd' as username and password and 'F' as output that is the Home page of the app. This can be denoted as F1(d)→F.

'F2' is the function in which the user creates a listing for the rideshare. This function takes in username and password (output of the previous function f1) as input and displays the listing as output. This can be viewed as F2(F)→F'/R where F' is the data listing.

After the user creates the listing, the ride schedule is then stored in the database. The function 'F3' takes F' as input and gives 'dbs' as output, denoted by F3(F') →dbs. Now the rider searches his/her destination and the function ' F4' looks for the required details in the database resulting in a match found or not.

This can be described as F4(dbs)→m where m is the result of a match found or not.

The next function 'F5' will be fare recommendation which will be done over the constraints like place, time, traffic congestion at that specific time. All the processing will happen using a machine learning model. This process is denoted by F5(m)→F".

The final part of our app is to find the optimal listing. The data listed by the driver will be compared to data requirements of the rider. This is the function 'F6' denoted by F6(F") → Su

1. **System Architecture**

   System S is defined as collection of following set: S = Ip, Op, Ss, Su, Fi, A

Mapping Function f(x)	X	Y
f1(Ip1)→ Op1	Ip1	Op1
F2(Ip2) → Op2	Ip2	Op2
F3(Ip2) → Op2	Op2	Op2
F4(Ip2) → Op2	Ip2	Op2
F5(Ip2) → Op2	Ip3	Op3
F6(Op3) → Op3	Op3	Op3

2. **Objects:**

   (a) **Input1: Ip1 = Username, Password**

   (b) **Input2: Ip2= Enter driver data (date, time, price)/Enter rider data(date, time)**

   (c) **Input3: Ip3= Enter destination**

   (a) **Output1: Op1 = Homepage (driver/rider)**

   (b) **Output2: Op2 = Data record created**

   (c) **Output3: Op3 = ML model processing for price recommendation+ List of available drivers**

Chapter 7

TECHNOLOGY USED

7.1 FRONTEND

1. **HTML (Hyper Text Markup Language)**

   HTML is the standard markup language used to create and design web pages. It provides a structure for content on the internet, allowing developers to organize and format text, images, links, and other media elements.

   (a) **Features of HTML:**

   i. **It is easy to learn and easy to use.**
   ii. **It is platform-independent.**
   iii. **Images, videos, and audio can be added to a web page.**
   iv. **Hypertext can be added to the text.**
   v. **It is a markup language.**

   (b) **HTML Page Structure: The basic structure of an HTML page is shown in fig.7.1. It contains the essential building-block elements (i.e. doctype declaration, HTML, head, title, and body elements) upon which all web pages are created.**

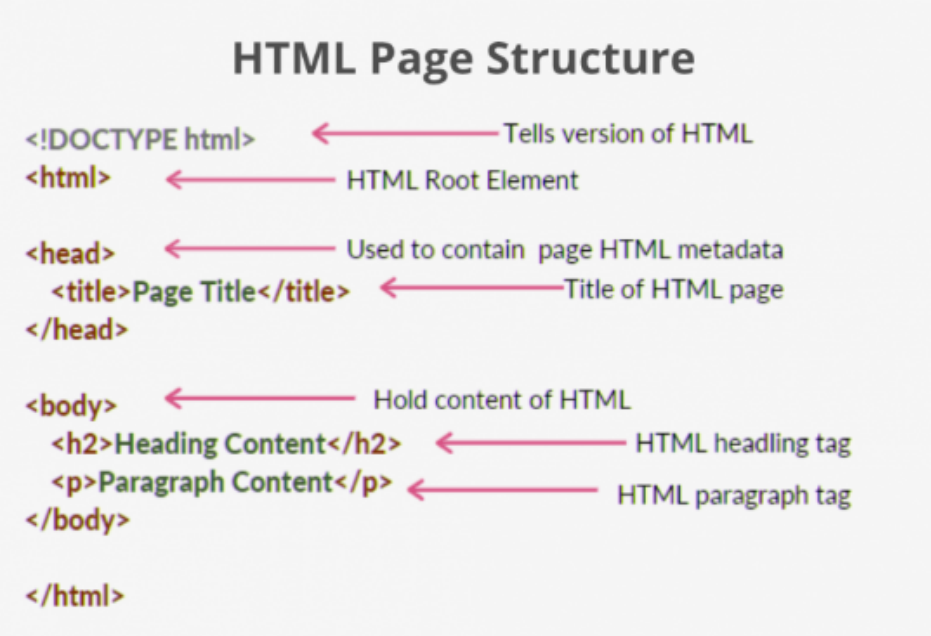


Figure 7.1: Basic Structure of an HTML

2. **CSS (Cascading Style Sheets)**

   Cascading Style Sheets (CSS) is a style sheet language used for specifying the presentation and styling of a document written in a markup language such as HTML or XML (including XML dialects such as SVG, MathML or XHTML). CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.

   The three types of CSS are external, internal, and inline. External CSS is a file that HTML files will link to. Internal CSS is specified at the beginning of an HTML document. Inline CSS is written for a specific element in the HTML document.

3. **JavaScript**

   JavaScript is a lightweight, cross-platform, single-threaded, and interpreted compiled programming language. It is also known as the scripting language for Webpages. It is well-known for the development of web pages, and many non-browser environments also use it.

   JavaScript is a weakly typed language (dynamically typed). JavaScript can be used for Client-side developments as well as Server-side developments. JavaScript is both an imperative and declarative type of language. JavaScript contains a standard library of objects, like Array, Date, and Math, and a core set of language elements like operators, control structures, and statements.

   (a) **Client-side: It supplies objects to control a browser and its Document Object Model (DOM). Like if client-side extensions allow an application to place elements on an HTML form and respond to user events such as mouse clicks, form input, and page navigation. Useful libraries for the client side are AngularJS and so many others.**

   (b) **Server-side: It supplies objects relevant to running JavaScript on a server. For if the server-side extensions allow an application to communicate with a database, and provide continuity of information from one invocation to another of the application, or perform file manipulations on a server. The useful framework which is the most famous these days is node.js.**

   (c) **Imperative language: In this type of language we are mostly concerned about how it is to be done. It simply controls the flow of computation. The procedural programming approach, object, oriented approach comes under this as async await we are thinking about what is to be done further after the async call.**

   (d) **Declarative programming: In this type of language we are concerned about how it is to be done, basically here logical computation requires. Her main goal is to describe the desired result without direct dictation on how to get it as the arrow function does.**

   These are just some of the basics of JavaScript. As you delve deeper, you'll discover more advanced topics like closures, prototypes, events, DOM manipulation, Asynchronous Programming, error handling, and more.

7.2 BACKEND

1. Solidity

   Solidity is a brand-new programming language created by Ethereum which is the second-largest market of cryptocurrency by capitalization, released in the year 2015 and led by Christian Reitwiessner.

   Solidity is a high-level programming language designed for implementing smart contracts. It is a statically typed object-oriented(contract-oriented) language. Solidity is highly influenced by Python, C++, and JavaScript which run on the Ethereum Virtual Machine (EVM). Solidity supports complex user-defined programming, libraries, and inheritance. Solidity is the primary language for blockchains running platforms. Solidity can be used to create contracts like voting, blind auctions, crowdfunding, multi-signature wallets, etc.

   (a) Ethereum: Ethereum is a decentralized open-source platform based on the blockchain domain, used to run smart contracts i.e. applications that execute the program exactly as it was programmed without the possibility of any fraud, interference from a third party, censorship, or downtime. It serves as a platform for nearly 2,60,000 different cryptocurrencies. Ether is a cryptocurrency generated by Ethereum miners, used to reward for the computations performed to secure the blockchain.

   (b) Ethereum Virtual Machine(EVM): Ethereum Virtual Machine abbreviated as EVM is a runtime environment for executing smart contracts in ethereum. It focuses widely on providing security and execution of untrusted code using an international network of public nodes. EVM is specialized to prevent Denial-of-service attack and confirms that the program does not have any access to each other's state, also ensures that the communication is established without any potential interference.

   (c) Smart Contract: Smart contracts are high-level program codes that are compiled to EVM byte code and deployed to the ethereum blockchain for further execution. It allows us to perform credible transactions without any interference of the third party, these transactions are trackable and irreversible. Languages used to write smart contracts are Solidity (a language library with similarities to C and JavaScript), Serpent (similar to Python, but deprecated), LLL (a low-level Lisp-like language), and Mutan (Go-based, but deprecated).

2. Ganache

   Ganache is a personal Blockchain network allowing developers to quickly build and test distributed Ethereum and Filecoin applications. It can be used throughout development, providing a secure and predictable environment for developing, deploying, and testing dApps.

   However, Ganache has its limitations. Since it is a private network, it does not accurately mimic the actions of miners on the primary grid. This can cause issues when developers need to test the behavior of smart contracts that depend on miner actions. Ganache provides developers with advanced mining controls and a built-in block explorer, making testing and inspecting smart contracts during development easier.

(a) Ganache and Truffle Suite

Ganache, a crucial component of the Truffle Suite framework, is a high-end development tool that plays a pivotal role in innovative contract development processes. Alongside other elements such as Truffle and Drizzle, it offers a complete package for developing decentralized applications based on the Ethereum Virtual Machine.

(b) Understanding Ganache's Functionality in Blockchain

Ganache is a personal local Blockchain network for developing decentralized applications on Ethereum and Corda. It enables developers to run their projects in a deterministic and safe environment. Moreover, Ganache offers many plausible advantages, making it an ideal choice for smart contract development.

(c) Advantages of Ganache in Smart Contract Development

The most significant advantage of using Ganache in smart contract development is its ease of developing, testing and deploying smart contracts and dApp projects. Its deterministic nature ensures consistent results, reducing the risk of errors in the development process. Furthermore, Ganache offers two different variants, depending on the required functionality.

(d) The Importance of Ganache in Solidity Smart Contract Development

One of the most significant advantages of using Ganache in smart contract development is that it offers a safe and deterministic environment for testing smart contracts and dApp projects. When developing a smart contract, ensuring it works flawlessly in a secure environment before deploying it on the main Ethereum Blockchain or test networks is crucial. Ganache provides a personal Blockchain network to test your smart contracts, which helps you avoid the costs associated with deploying contracts on the leading network.

3. Node.js

Node.js is a cross-platform, open-source JavaScript runtime environment that can run on Windows, Linux, Unix, macOS, and more. Node.js runs on the V8 JavaScript engine, and executes JavaScript code outside a web browser. Node.js lets developers use JavaScript to write command line tools and for server-side scripting. The ability to run JavaScript code on the server is often used to generate dynamic web page content before the page is sent to the user's web browser. Consequently, Node.js represents a "JavaScript everywhere" paradigm, unifying web-application development around a single programming language, as opposed to using different languages for the server- versus client-side programming.

Node.js has an event-driven architecture capable of asynchronous I/O. These design choices aim to optimize throughput and scalability in web applications with many input/output operations, as well as for real-time Web applications (e.g., real-time communication programs and browser games). The Node.js distributed development project was previously governed by the Node.js Foundation, and has now merged with the JS Foundation to form the OpenJS Foundation. OpenJS Foundation is facilitated by the Linux Foundation's Collaborative Projects program.

4. jQuery

jQuery is a lightweight, "write less, do more", JavaScript library. The purpose of jQuery is to make it much easier to use JavaScript on your website. jQuery takes a lot of common tasks that require many lines of JavaScript code to accomplish, and wraps them into methods that you can call with a single line of code. jQuery also simplifies a lot of the complicated things from JavaScript, like AJAX calls and DOM manipulation.

The jQuery library contains the following features:

(a) HTML/DOM manipulation

(b) CSS manipulation

(c) HTML event methods

(d) Effects and animations

(e) AJAX

(f) Utilities

5. Hardhat

Hardhat is an Ethereum software development environment. It consists of several parts that work together to form a comprehensive development environment for editing, compiling, debugging, and deploying your dApps and smart contracts. The primary interface that you use with Hardhat is called Hardhat Runner. It's an adaptable and expandable task runner that assists you in organizing and automating the repetitive chores involved in creating dApps and smart contracts.

The design of Hardhat Runner revolves around the ideas of tasks and plugins. Each time you launch Hardhat via the command line, a task is launched. For instance, the built-in compile job is executed by npx hardhat compile. Tasks have the ability to call one another, defining intricate workflows. Because current tasks can be overridden by users or plugins, those workflows are adaptable and expandable.

Hardhat is utilized in our project through a local installation. In this manner, we may replicate our environment and steer clear of version conflicts in the future. We must first create an empty folder, execute npm init, then adhere to its instructions in order to install it. While yarn is a viable alternative, we prefer to use npm 7 or later due to its ease of use when installing Hardhat plugins.

6. Mongodb

An open source NoSQL database management system is called MongoDB. Conventional relational databases being replaced with NoSQL (not only SQL). NoSQL databases can be handy when handling big, dispersed data sets. MongoDB is a tool for managing, storing, and retrieving document-oriented data. High-volume data storage is facilitated by MongoDB, which enables businesses to store vast volumes of data quickly. Ad hoc queries, indexing, load balancing, aggregation, server-side JavaScript execution, and other features are some of the other reasons why businesses utilize MongoDB.

(a) How does MongoDB work?

Users can construct databases with MongoDB on servers that are part of MongoDB environments. Data is stored in MongoDB as records, which are composed of documents and collections. The user-selected data to be stored in the MongoDB database is included in documents. Documents consist of pairs of fields and values. They serve as MongoDB's fundamental data unit. The papers use a variation known as Binary JSON (BSON), which is related to JavaScript Object Notation (JSON). Using BSON has the advantage of supporting more data types. These documents' fields resemble a relational database's columns. The MongoDB user manual states that values contained can be a wide range of data formats, such as other documents, arrays, and arrays of documents.A primary key will also be included in documents as a means of unique identification. The structure of a document can be altered by adding or removing fields.

(b) MongoDB platforms

MongoDB Inc., the provider, offers both community and commercial versions of the database. The open-source MongoDB Community Edition version is accompanied by MongoDB Enterprise Server, which offers additional security features, an in-memory storage engine, administration and authentication functionalities, and Ops Manager monitoring tools.

MongoDB Compass is a graphical user interface (GUI) that allows users to deal with document structure, perform queries, index data, and more. Users can visualize data and generate reports using SQL queries by connecting their NoSQL database to business intelligence tools with the MongoDB Connector for BI.

7. Express

Express is a Node.js web application framework that is simple to use. It has a quick, simple, and modular design. Express is a project covered by an open-source, free MIT license. It has a vibrant contributor community and is a component of the OpenJS foundation. Express adds routing, middleware, and view functionality to the Node.js http package. Express is a popular tool for developing APIs, particularly RESTful ones. This basic routing example can be expanded upon to interact with actual data. Developing a REST API to store data in a database, like MongoDB, is a typical use case. The various resources that the API exposes as endpoints correspond to database collections.

Does MongoDB support Express?

By installing the MongoDB Node.js driver, we may use MongoDB in your Express application. One Node.js module that helps us work with our data and connect our application to MongoDB is the MongoDB Node.js driver.It is regarded as best practice to separate our routing logic from your data storage logic if we are developing a REST API. For instance, we might have a Product model that houses the data and a Product controller that responds to queries for the /products resource.

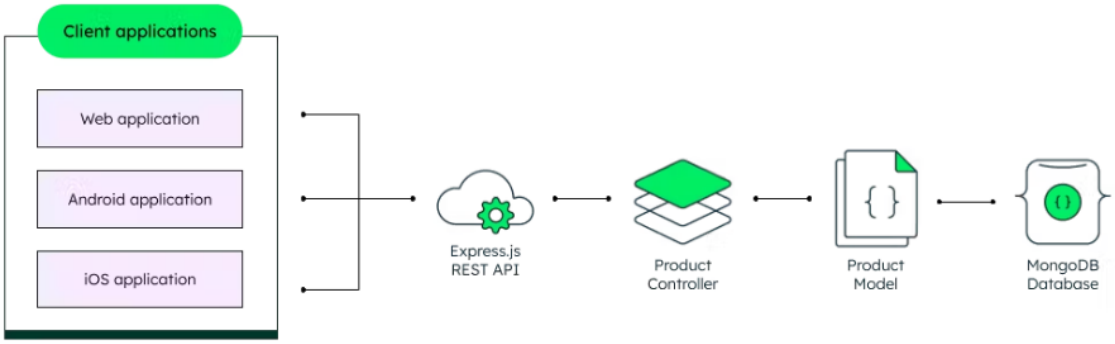


Figure 7.2: MongoDb Supports Express [5]

8. **Google Map API**

Developers can incorporate Google Maps into mobile applications and websites, as well as get data from Google Maps, using the Google Maps Platform, which consists of a collection of APIs and SDKs.There are several offerings.The Maps JavaScript API lets you customize maps with your own content and imagery for display on web pages and mobile devices.Google APIs give you programmatic access to Google Maps, Google Drive, YouTube, and many other Google products. To make coding against these APIs easier, Google provides client libraries that can reduce the amount of code you need to write and make your code more robust. Like many other Google web applications, Google Maps uses JavaScript extensively. The site also uses protocol buffers for data transfer rather than JSON, for performance reasons.The version of Google Street View for classic Google Maps required Adobe Flash.

Depending on your needs, you may find yourself using one or a combination of these APIs and SDKs:

Maps:

(a) Maps JavaScript API
(b) Maps SDK for Android
(c) Maps SDK for iOS
(d) Map Tiles API
(e) Aerial View API
(f) Maps Static API
(g) Street View Static API
(h) Maps URLs
(i) Maps Embed API

Routes:

(a) Routes API
(b) Roads API
(c) Directions API
(d) Distance Matrix API

Places:

(a) Places API
(b) Places SDK for Android
(c) Places SDK for iOS
(d) Places Library, Maps JavaScript API
(e) Address Validation API
(f) Geocoding API
(g) Geolocation API
(h) Time Zone API

Environment:

(a) Air Quality API
(b) Pollen API
(c) Solar API

Chapter 8

ADVANTAGES OF BLOCKCHAIN

Using blockchain technology in a ride-sharing app can offer several significant advantages:

8.1 Decentralization

1. Reduced Central Authority:Traditional ride-sharing platforms rely on a central authority to match riders with drivers and handle transactions. Blockchain enables a decentralized platform where these activities can be managed by a distributed network, reducing the need for a central intermediary.
2. Trust: Decentralization can enhance trust among users as the system operates transparently without a central point of control that might act in its own interest.

8.2 Transparency and Trust

1. Immutable Records: All transactions are recorded on the blockchain, providing an immutable and transparent ledger. This can help in resolving disputes, as both parties can independently verify the transaction history.
2. Reputation Systems: Drivers and riders can build a verifiable reputation on the blockchain, making it difficult to fake reviews or ratings.

8.3 Cost Efficiency

1. Lower Fees: By removing intermediaries, blockchain can reduce transaction fees, making the service cheaper for both riders and drivers.
2. Direct Payments: Smart contracts can automate payments directly from riders to drivers without the need for a third-party payment processor.

8.4 Security

1. Enhanced Security: Blockchain technology provides robust security features, protecting against fraud and hacking. Each transaction is encrypted and linked to the previous transaction, making it highly secure.

2. Data Privacy: Users have control over their data, and sensitive information can be shared securely without risk of exposure.

8.5 Smart Contracts

1. Automated Agreements: Smart contracts can automate the execution of agreements between riders and drivers, ensuring that payments are only made when certain conditions are met (e.g., ride completion).
2. Dispute Resolution: Smart contracts can include predefined conditions for resolving disputes, reducing the need for human intervention.

8.6 Tokenization and Incentives

1. Incentives: Platforms can introduce tokens as a form of currency within the ecosystem, rewarding users for good behavior or loyalty. These tokens can be used for rides or exchanged for other cryptocurrencies or fiat money.
2. Investment Opportunities: Users can potentially invest in the platform itself by holding tokens, aligning their interests with the success of the platform.

8.7 Global Reach

1. Cross-Border Transactions: Blockchain facilitates seamless cross-border transactions without the complications of currency exchange rates or international banking fees.
2. Universal Access: The decentralized nature of blockchain can make the service available in regions where traditional banking and financial services are not well-established.

8.8 Enhanced User Experience

1. Unified User Interface: With blockchain, users can have a unified interface for various ride-sharing services, potentially integrating with other blockchain-based services like payments, insurance, and more.
2. Dynamic Pricing: Smart contracts can enable dynamic pricing models that are fair and transparent, adjusting prices based on supply and demand without the need for manual intervention.

8.9 Ownership and Governance

1. User Ownership: Decentralized platforms can distribute ownership and control among users, creating a more democratic and user-focused service.
2. Community Governance: Blockchain platforms can implement decentralized governance models where users have a say in the decision-making process, enhancing user satisfaction and platform adaptability.

Chapter 9

SOFTWARE TESTING

Software program testing is the procedure of comparing a software program object to come across variations among given input and predicted output. Also to evaluate the characteristics of a software program item. Software testing is a method that ought to be accomplished at some stage in the development process. In different phrases software program testing is a verification and validation method.

9.1 VERIFICATION

Verification is the process to make certain the product satisfies the conditions imposed on the start of the development phase. In different phrases, to make sure the product behaves the manner we want it to.Verification focuses on answering the question: "Are we building the product right?" It is concerned with the conformance of the software product to its specifications and requirements. Verification is often contrasted with validation, which is the process of evaluating the software to ensure that it meets the user's needs and expectations.

9.2 VALIDATION

Validation is the technique to make certain the product satisfies the required requirements on the give up of the development phase. In different phrases, to make sure the product is built as consistent with client necessities. Validation is essential because it ensures that the software meets the user's needs and expectations and functions correctly in its intended environment. It helps mitigate the risk of delivering a product that does not satisfy the user requirements or fails to perform as expected.

9.3 BASICS OF SOFTWARE TESTING

There are two basics of software testing: black-box testing and white-box testing.

1. Black-box Testing: Black box testing only focuses on the output generated and ignores the internal structure of the system.
2. White-box Testing: White Box testing or the structural testing focuses on the internal structure of the system and is always used for validation purposes.

9.4 TEST CASES

Test Case ID	**1**
Test Case Description	**Run Xamarin code on virtual machine**
Steps	**1. Debug and Compile the code. 2. Run the code.**
Test Case Result	**Application should be successful installed.**
Action Result	**Application successfully installed and started.**
Status	**Pass**

Test Case ID	**2**
Test Case Description	**Applications home page should get displayed on start-up with option of login and registration.**
Steps	**1. Open application 2. Register 3.Login**
Test Case Result	**Application should be successfully started and Register Form should be accessible.**
Action Result	**Application accepted registration data and able to login successfully.**
Status	**Pass**

Test Case ID	**3**
Test Case Description	**Selection between driver and rider**
Steps	**1. Login 2. Select driver or rider**
Test Case Result	**Display the respective page for driver and rider.**
Action Result	**Both pages are displayed after each selection test.**
Status	**Pass**

Test Case ID	**4**
Test Case Description	**Processing of data for driver page**
Steps	**1. Select as driver** **2. Enter the required data like time, place, and price.** **3. Submit the data** **4. Get recommended price**
Test Case Result	**Data is stored in database and listing is created.** **Recommended price is generated.**
Action Result	**Data successfully stored and listed**
Status	**Pass**

Test Case ID	**5**
Test Case Description	**Processing of data for rider page**
Steps	**1. Select as rider** **2. Enter the required data like time and place.** **3. Submit the data**
Test Case Result	**Data is compared with the available rider data in database.**
Action Result	**Data compared and match was found for the ride.**
Status	**Pass**

Chapter 10

RESULT

10.1 CODE

1. Home Page

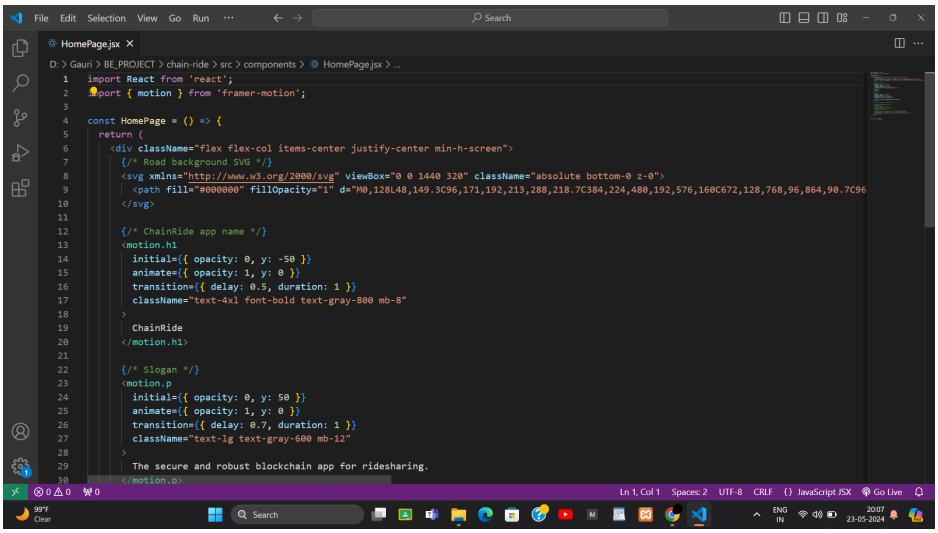


Figure 10.1: Screenshot of code of Home Page

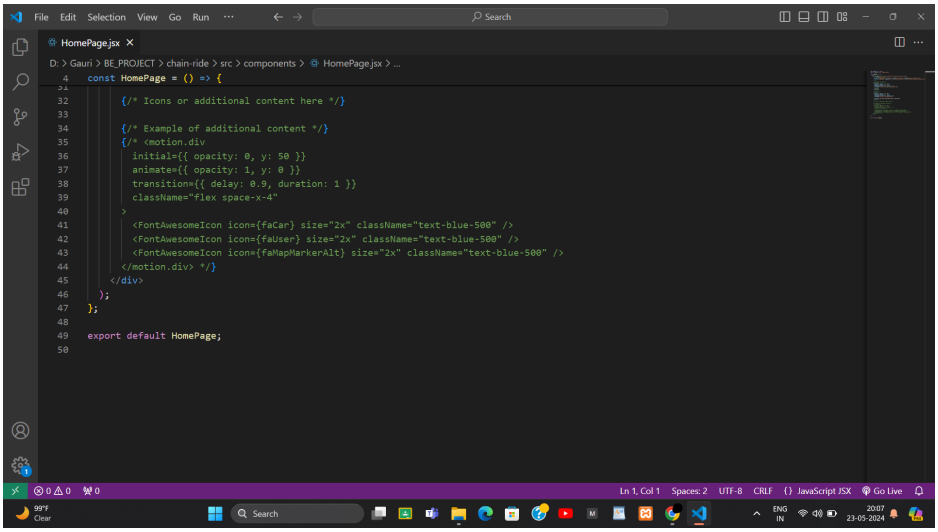
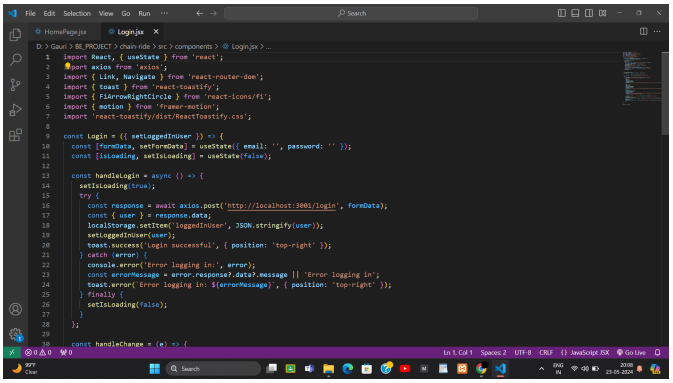


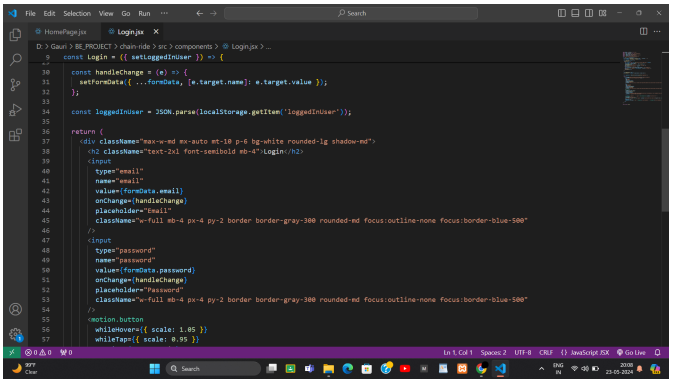
Figure 10.2: Screenshot of code of Home Page

2. Login Page



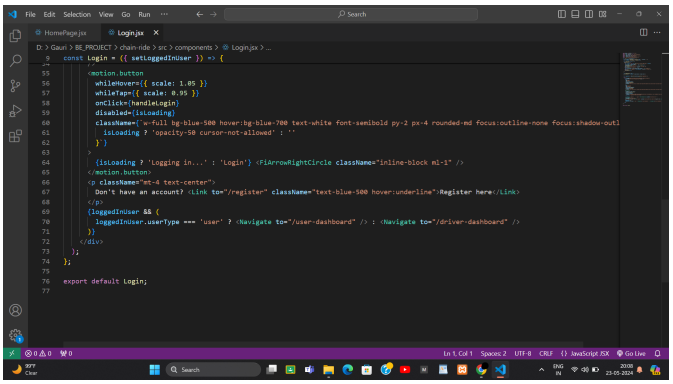
```jsx
import React, { useState } from 'react';
import axios from 'axios';
import { Link, Navigate } from 'react-router-dom';
import { toast } from 'react-toastify';
import { FiArrowRightCircle } from 'react-icons/fi';
import { motion } from 'framer-motion';
import 'react-toastify/dist/ReactToastify.css';

const Login = ({ setLoggedInUser }) => {
  const [formData, setFormData] = useState({ email: '', password: '' });
  const [isLoading, setIsLoading] = useState(false);

  const handleLogin = async () => {
    setIsLoading(true);
    try {
      const response = await axios.post('http://localhost:3001/login', formData);
      const { user } = response.data;
      localStorage.setItem('loggedInUser', JSON.stringify(user));
      setLoggedInUser(user);
      toast.success('Login successful', { position: 'top-right' });
    } catch (error) {
      console.error('Error logging in:', error);
      const errorMessage = error.response?.data?.message || 'Error logging in';
      toast.error(`Error logging in: ${errorMessage}`, { position: 'top-right' });
    } finally {
      setIsLoading(false);
    }
  };

  const handleChange = (e) => {
```

Figure 10.3: Screenshot of code of Login Page



```jsx
const Login = ({ setLoggedInUser }) => {

  const handleChange = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };

  const loggedInUser = JSON.parse(localStorage.getItem('loggedInUser'));

  return (
    <div className="max-w-md mx-auto mt-10 p-6 bg-white rounded-lg shadow-md">
      <h2 className="text-2xl font-semibold mb-4">Login</h2>
      <input
        type="email"
        name="email"
        value={formData.email}
        onChange={handleChange}
        placeholder="Email"
        className="w-full mb-4 px-4 py-2 border border-gray-300 rounded-md focus:outline-none focus:border-blue-500"
      />
      <input
        type="password"
        name="password"
        value={formData.password}
        onChange={handleChange}
        placeholder="Password"
        className="w-full mb-4 px-4 py-2 border border-gray-300 rounded-md focus:outline-none focus:border-blue-500"
      />
      <motion.button
        whileHover={{ scale: 1.05 }}
        whileTap={{ scale: 0.95 }}
```

Figure 10.4: Screenshot of code of Login Page



```jsx
const Login = ({ setLoggedInUser }) => {
      <motion.button
        whileHover={{ scale: 1.05 }}
        whileTap={{ scale: 0.95 }}
        onClick={handleLogin}
        disabled={isLoading}
        className={`w-full bg-blue-500 hover:bg-blue-700 text-white font-semibold py-2 px-4 rounded-md focus:outline-none focus:shadow-outl
          isLoading ? 'opacity-50 cursor-not-allowed' : ''
        }`}
      >
        {isLoading ? 'Logging in...' : 'Login'} <FiArrowRightCircle className="inline-block ml-1" />
      </motion.button>
      <p className="mt-4 text-center">
        Don't have an account? <Link to="/register" className="text-blue-500 hover:underline">Register here</Link>
      </p>
      {loggedInUser && (
        loggedInUser.userType === 'user' ? <Navigate to="/user-dashboard" /> : <Navigate to="/driver-dashboard" />
      )}
    </div>
  );
};

export default Login;
```

Figure 10.5: Screenshot of code of Login Page

3. Map Page

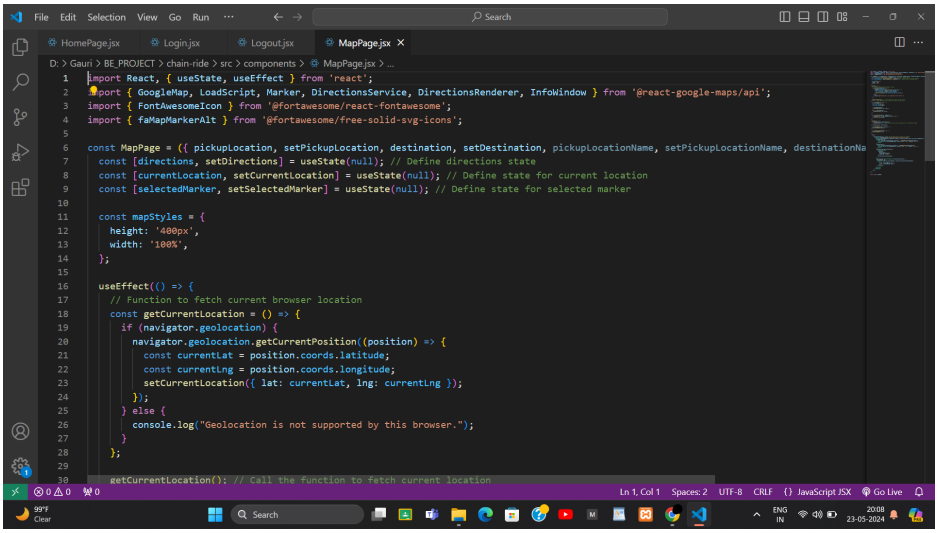


Figure 10.6: Screenshot of code of Map Page

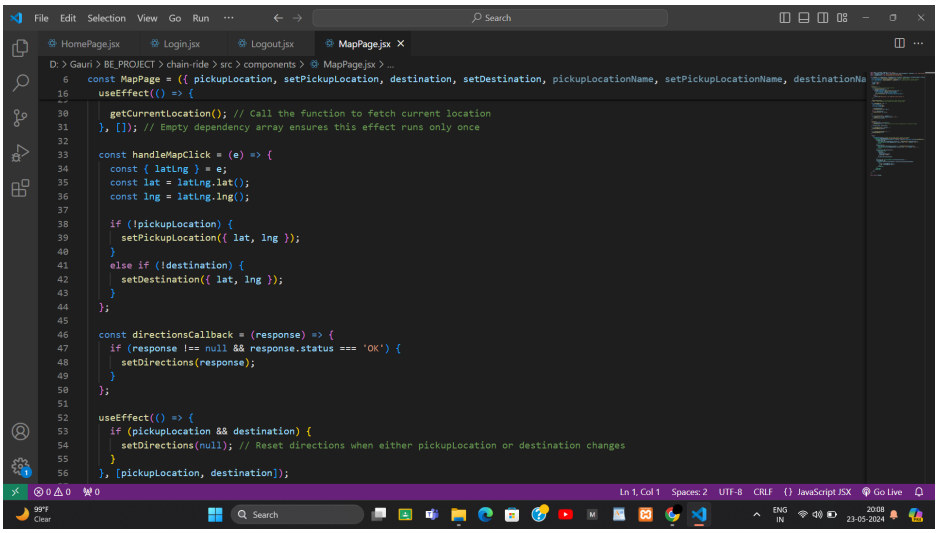


Figure 10.7: Screenshot of code of Map Page

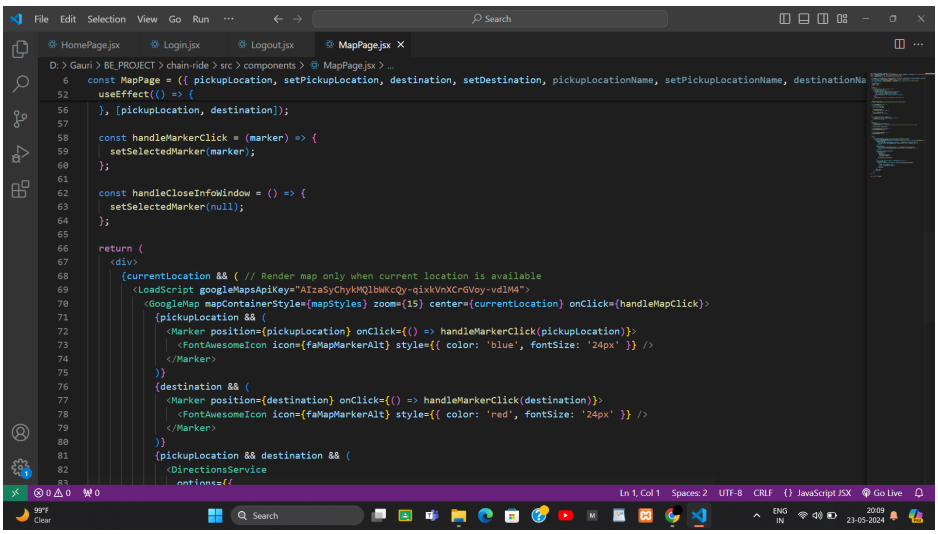


Figure 10.8: Screenshot of code of Home Page

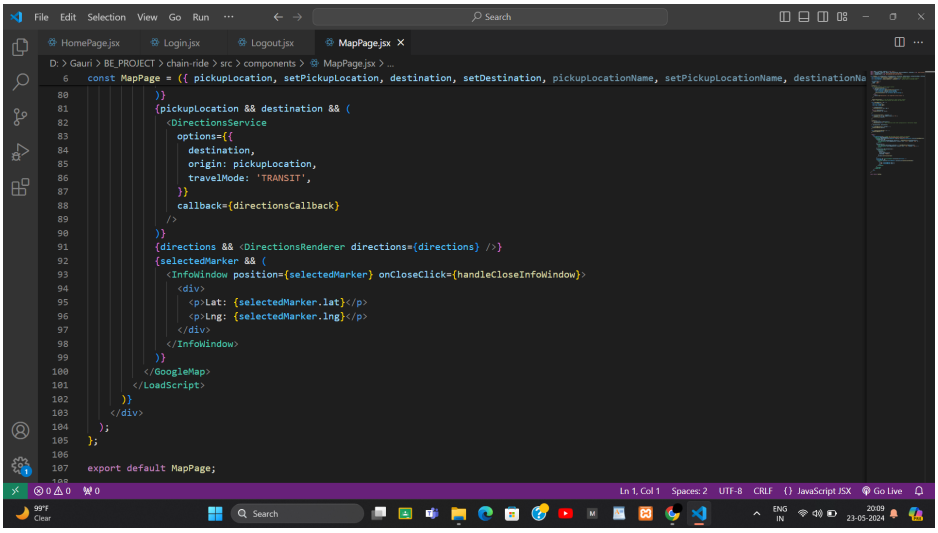


Figure 10.9: Screenshot of code of Home Page

4. Profile Page

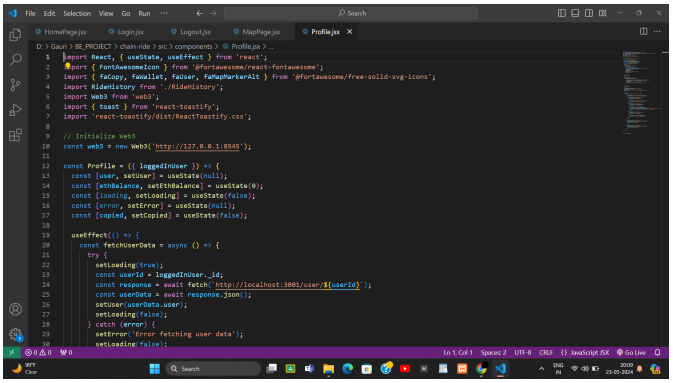


Figure 10.10: Screenshot of code of Profile Page

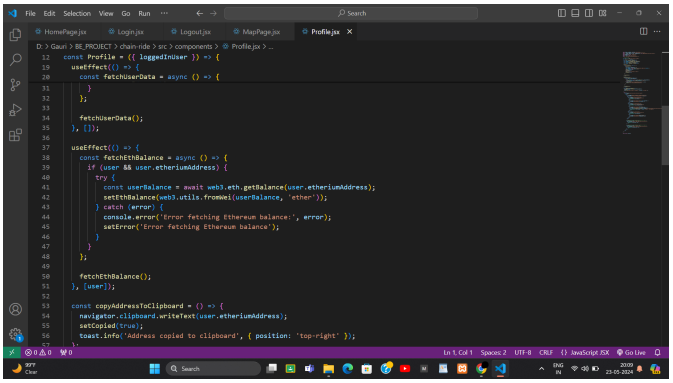


Figure 10.11: Screenshot of code of Profile Page

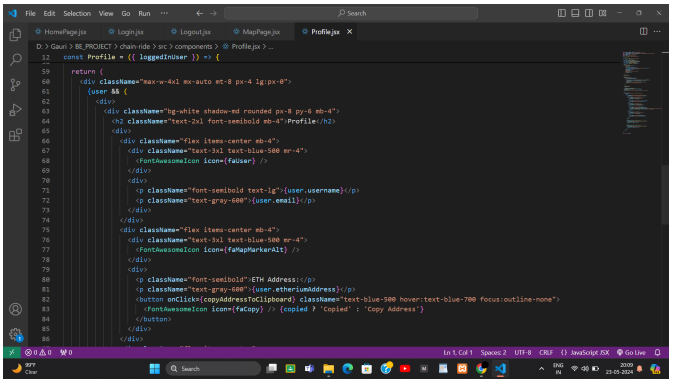


Figure 10.12: Screenshot of code of Profile Page

5. Registration Page

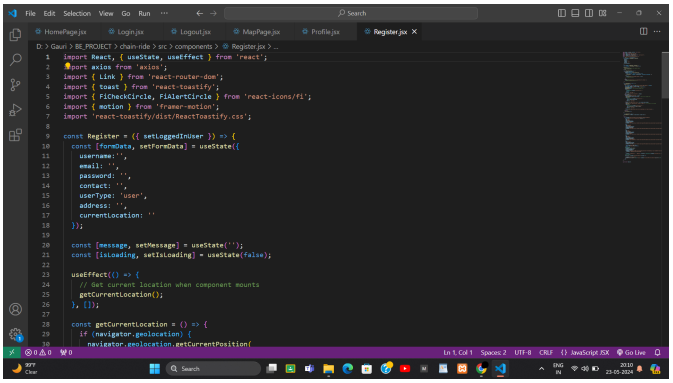


Figure 10.13: Screenshot of code of Registration Page

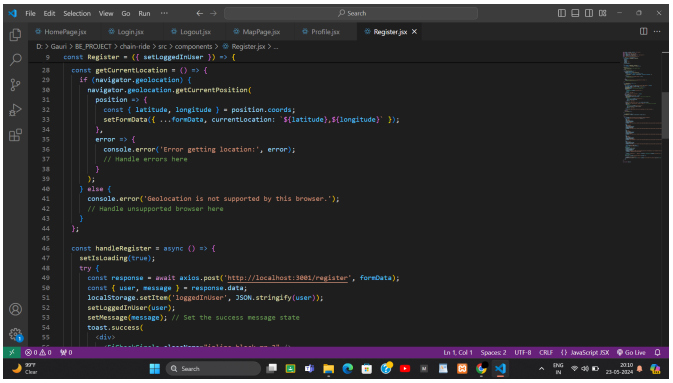


Figure 10.14: Screenshot of code of Registration Page

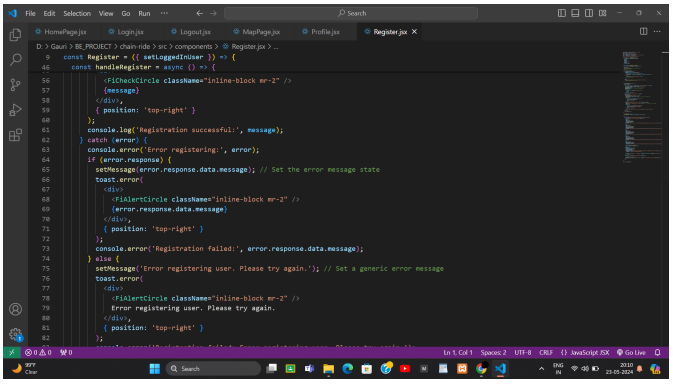


Figure 10.15: Screenshot of code of Registration Page

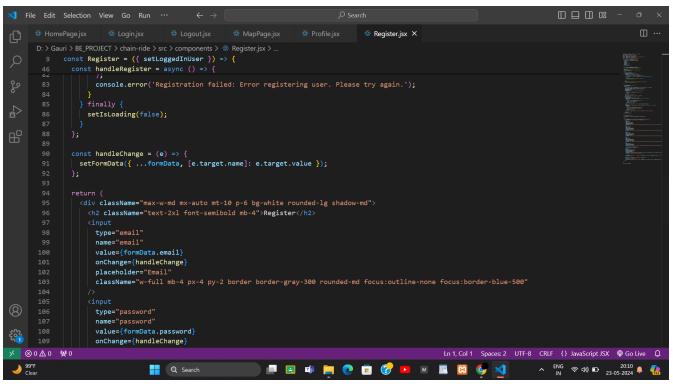


Figure 10.16: Screenshot of code of Registration Page

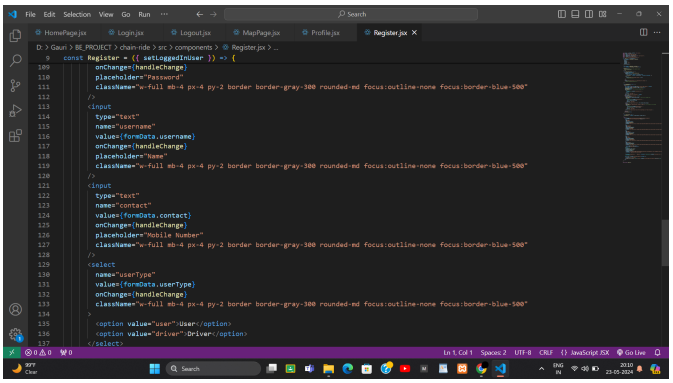


Figure 10.17: Screenshot of code of Registration Page

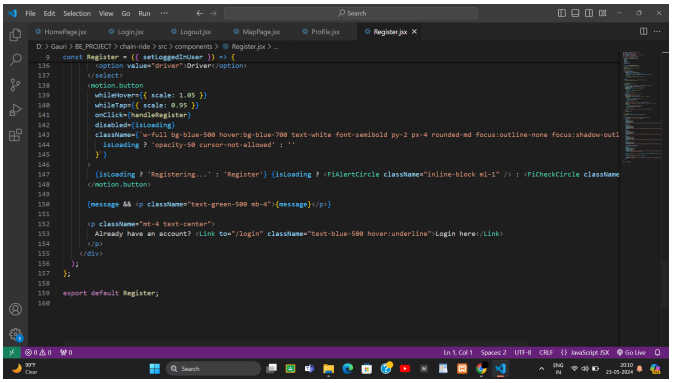
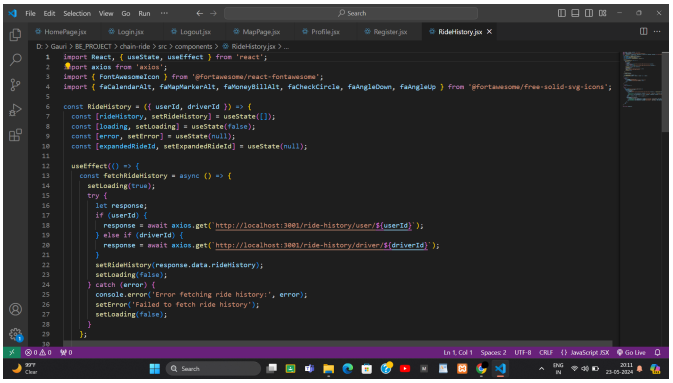


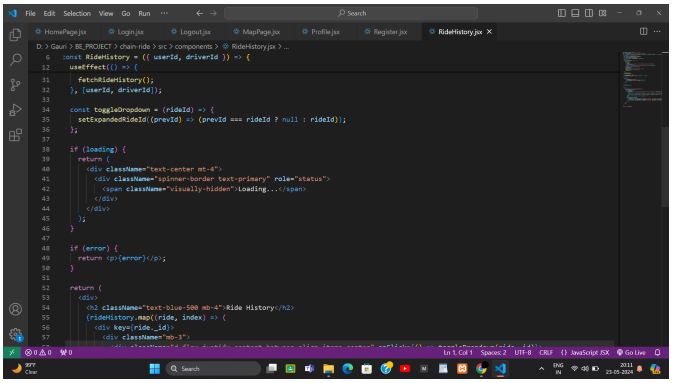
Figure 10.18: Screenshot of code of Registration Page

6. Ride History



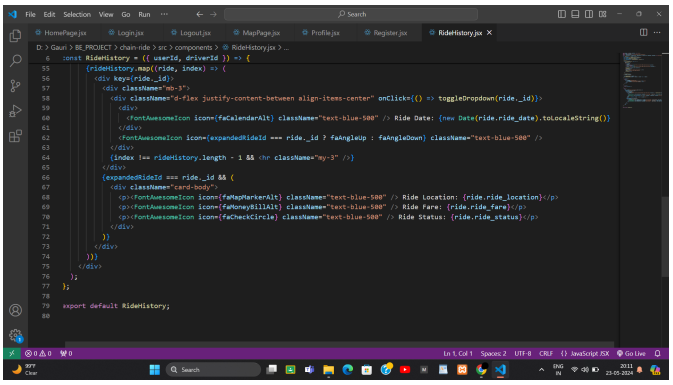
```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
import { faCalendarAlt, faMapMarkerAlt, faMoneyBillAlt, faCheckCircle, faAngleDown, faAngleUp } from '@fortawesome/free-solid-svg-icons';

const RideHistory = ({ userId, driverId }) => {
  const [rideHistory, setRideHistory] = useState([]);
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState(null);
  const [expandedRideId, setExpandedRideId] = useState(null);

  useEffect(() => {
    const fetchRideHistory = async () => {
      setLoading(true);
      try {
        let response;
        if (userId) {
          response = await axios.get(`http://localhost:3001/ride-history/user/${userId}`);
        } else if (driverId) {
          response = await axios.get(`http://localhost:3001/ride-history/driver/${driverId}`);
        }
        setRideHistory(response.data.rideHistory);
        setLoading(false);
      } catch (error) {
        console.error('Error fetching ride history:', error);
        setError('Failed to fetch ride history');
        setLoading(false);
      }
    };
```

Figure 10.19: Screenshot of code of Ride History



```
const RideHistory = ({ userId, driverId }) => {
  useEffect(() => {
    fetchRideHistory();
  }, [userId, driverId]);

  const toggleDropdown = (rideId) => {
    setExpandedRideId(prevId => (prevId === rideId ? null : rideId));
  };

  if (loading) {
    return (
      <div className="text-center mt-4">
        <div class="spinner-border text-primary" role="status">
          <span class="visually-hidden">Loading...</span>
        </div>
      </div>
    );
  }

  if (error) {
    return <p>{error}</p>;
  }

  return (
    <div>
      <h2 className="text-blue-500 mb-4">Ride History</h2>
      {rideHistory.map((ride, index) => (
        <div key={ride._id}>
          <div className="mb-3">
```

Figure 10.20: Screenshot of code of Ride History



```
const RideHistory = ({ userId, driverId }) => {
      {rideHistory.map((ride, index) => (
        <div key={ride._id}>
          <div className="mb-3">
            <div className="d-flex justify-content-between align-items-center" onClick={() => toggleDropdown(ride._id)}>
              <div>
                <FontAwesomeIcon icon={faCalendarAlt} className="text-blue-500" /> Ride Date: {new Date(ride.ride_date).toLocaleString()}
              </div>
              <FontAwesomeIcon icon={expandedRideId === ride._id ? faAngleUp : faAngleDown} className="text-blue-500" />
            </div>
            {index !== rideHistory.length - 1 && <hr className="my-3" />}
          </div>
          {expandedRideId === ride._id && (
            <div className="card-body">
              <p><FontAwesomeIcon icon={faMapMarkerAlt} className="text-blue-500" /> Ride Location: {ride.ride_location}</p>
              <p><FontAwesomeIcon icon={faMoneyBillAlt} className="text-blue-500" /> Ride Fare: {ride.ride_fare}</p>
              <p><FontAwesomeIcon icon={faCheckCircle} className="text-blue-500" /> Ride Status: {ride.ride_status}</p>
            </div>
          )}
        </div>
      ))}
    </div>
  );
};

export default RideHistory;
```

Figure 10.21: Screenshot of code of Ride History

7. Ride Status Page

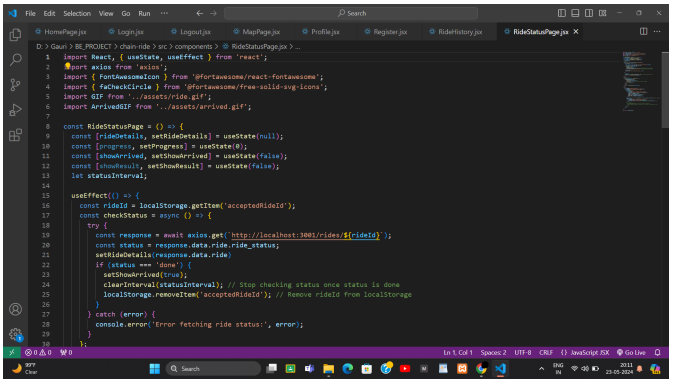


Figure 10.22: Screenshot of code of Ride Status Page

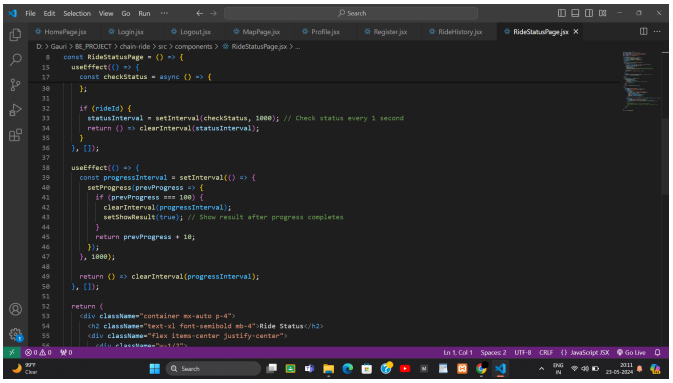


Figure 10.23: Screenshot of code of Ride Status Page

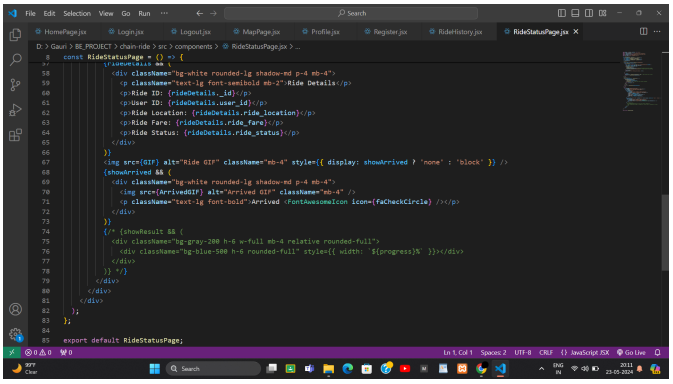


Figure 10.24: Screenshot of code of Ride Status Page

8. User Dashboard

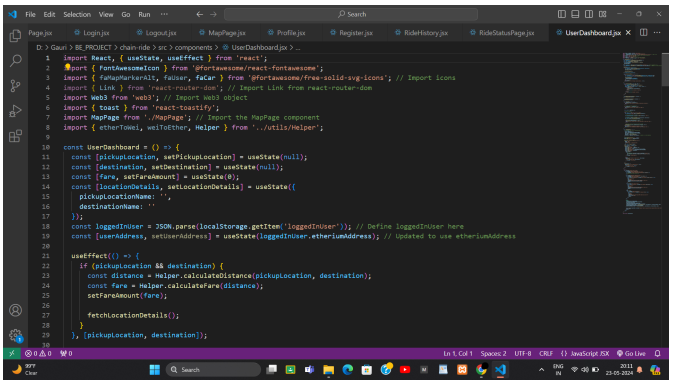


Figure 10.25: Screenshot of code of User Dashboard

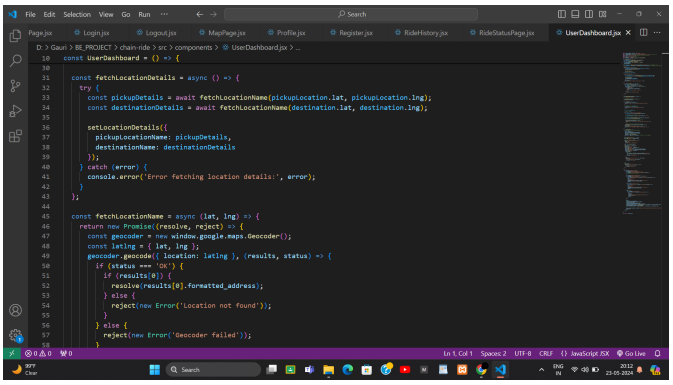


Figure 10.26: Screenshot of code of User Dashboard

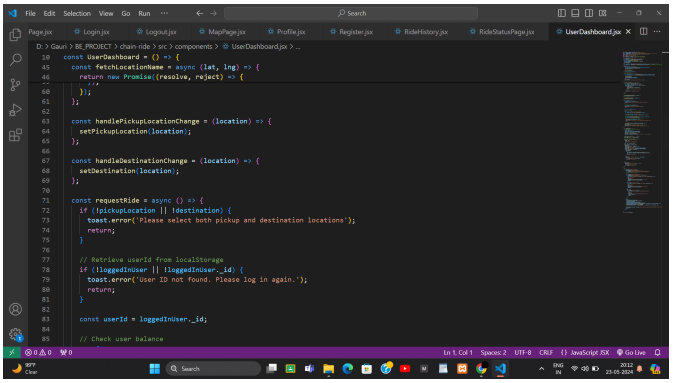


Figure 10.27: Screenshot of code of User Dashboard

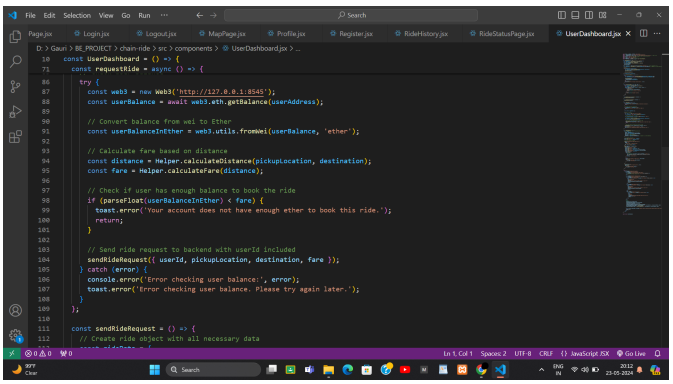


Figure 10.28: Screenshot of code of User Dashboard

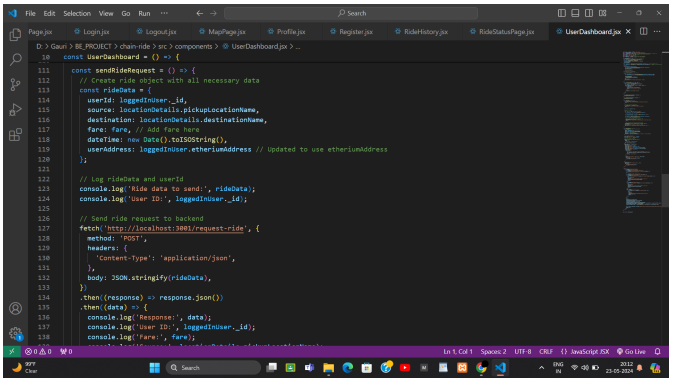


Figure 10.29: Screenshot of code of User Dashboard

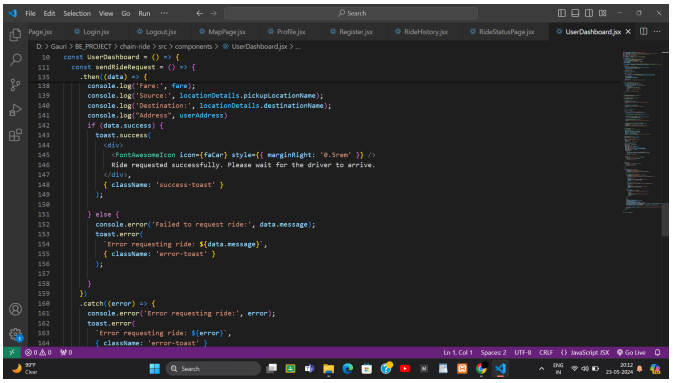
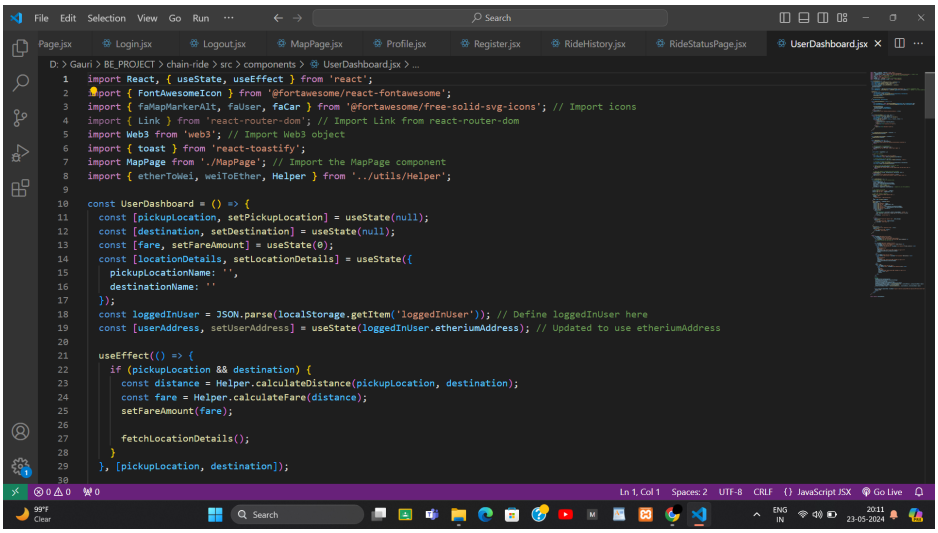


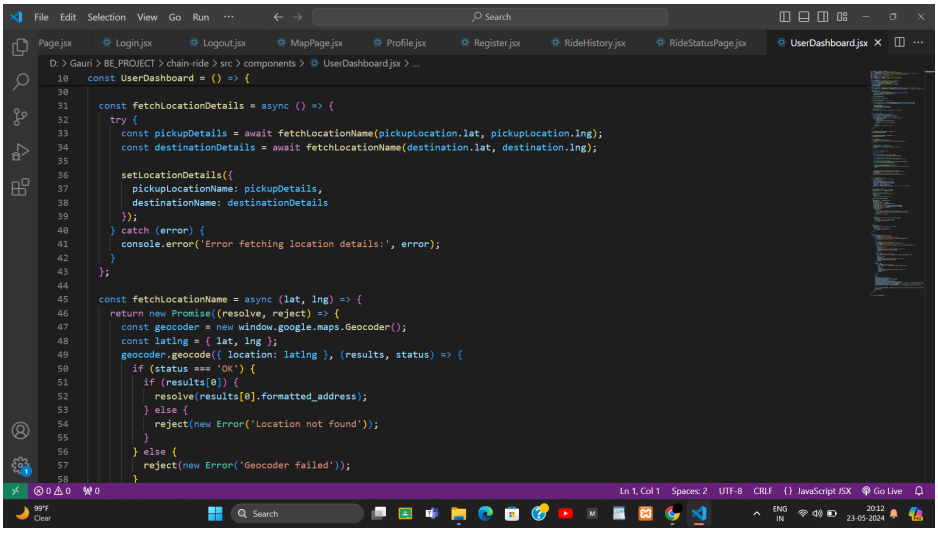
Figure 10.30: Screenshot of code of User Dashboard

9. Driver Dashboard



```
import React, { useState, useEffect } from 'react';
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
import { faMapMarkerAlt, faUser, faCar } from '@fortawesome/free-solid-svg-icons'; // Import icons
import { Link } from 'react-router-dom'; // Import Link from react-router-dom
import Web3 from 'web3'; // Import Web3 object
import { toast } from 'react-toastify';
import MapPage from './MapPage'; // Import the MapPage component
import { etherToWei, weiToEther, Helper } from '../utils/Helper';

const UserDashboard = () => {
  const [pickupLocation, setPickupLocation] = useState(null);
  const [destination, setDestination] = useState(null);
  const [fare, setFareAmount] = useState(0);
  const [locationDetails, setLocationDetails] = useState({
    pickupLocationName: '',
    destinationName: ''
  });
  const loggedInUser = JSON.parse(localStorage.getItem('loggedInUser')); // Define loggedInUser here
  const [userAddress, setUserAddress] = useState(loggedInUser.etheriumAddress); // Updated to use etheriumAddress

  useEffect(() => {
    if (pickupLocation && destination) {
      const distance = Helper.calculateDistance(pickupLocation, destination);
      const fare = Helper.calculateFare(distance);
      setFareAmount(fare);

      fetchLocationDetails();
    }
  }, [pickupLocation, destination]);
```

Figure 10.31: Screenshot of code of Driver Dashboard



```
const UserDashboard = () => {

  const fetchLocationDetails = async () => {
    try {
      const pickupDetails = await fetchLocationName(pickupLocation.lat, pickupLocation.lng);
      const destinationDetails = await fetchLocationName(destination.lat, destination.lng);

      setLocationDetails({
        pickupLocationName: pickupDetails,
        destinationName: destinationDetails
      });
    } catch (error) {
      console.error('Error fetching location details:', error);
    }
  };

  const fetchLocationName = async (lat, lng) => {
    return new Promise((resolve, reject) => {
      const geocoder = new window.google.maps.Geocoder();
      const latlng = { lat, lng };
      geocoder.geocode({ location: latlng }, (results, status) => {
        if (status === 'OK') {
          if (results[0]) {
            resolve(results[0].formatted_address);
          } else {
            reject(new Error('Location not found'));
          }
        } else {
          reject(new Error('Geocoder failed'));
        }
```

Figure 10.32: Screenshot of code of Driver Dashboard

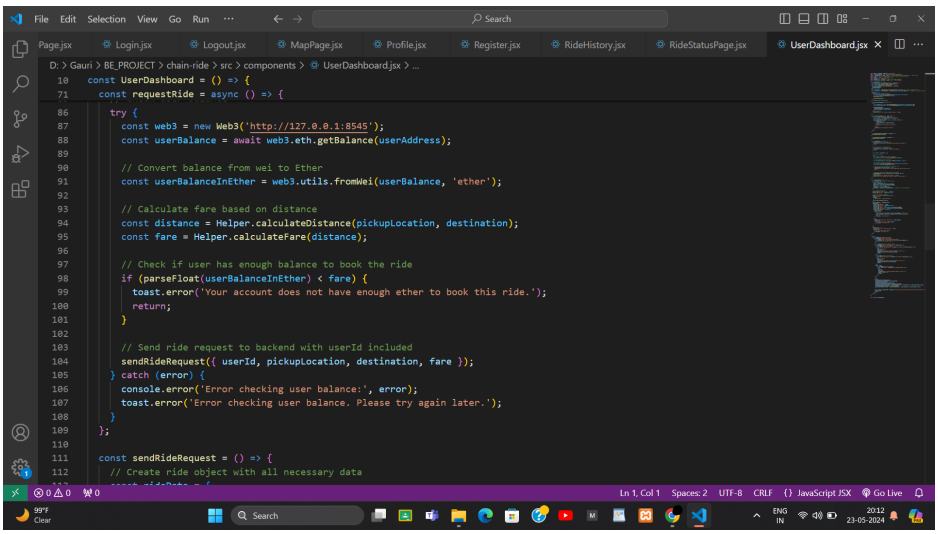


Figure 10.33: Screenshot of code of Driver Dashboard

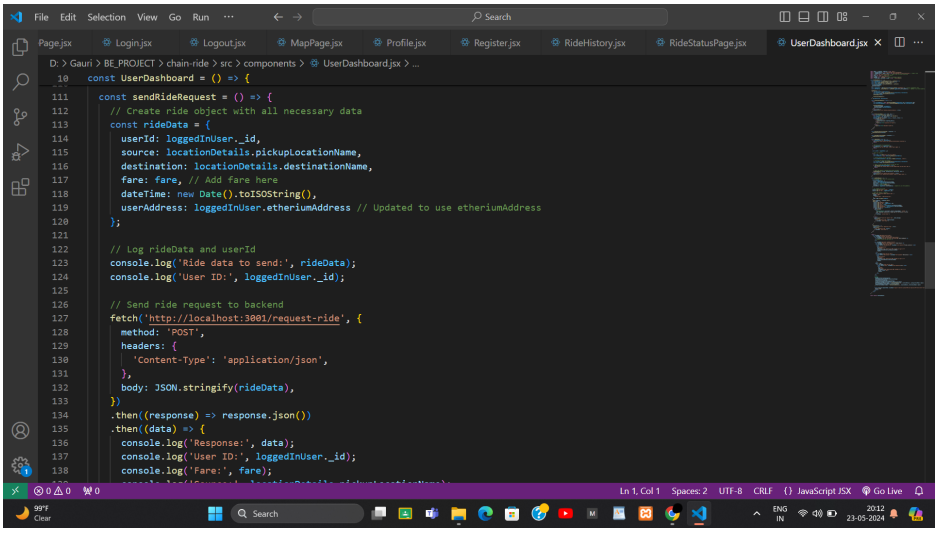


Figure 10.34: Screenshot of code of Driver Dashboard

10. Solidity Code

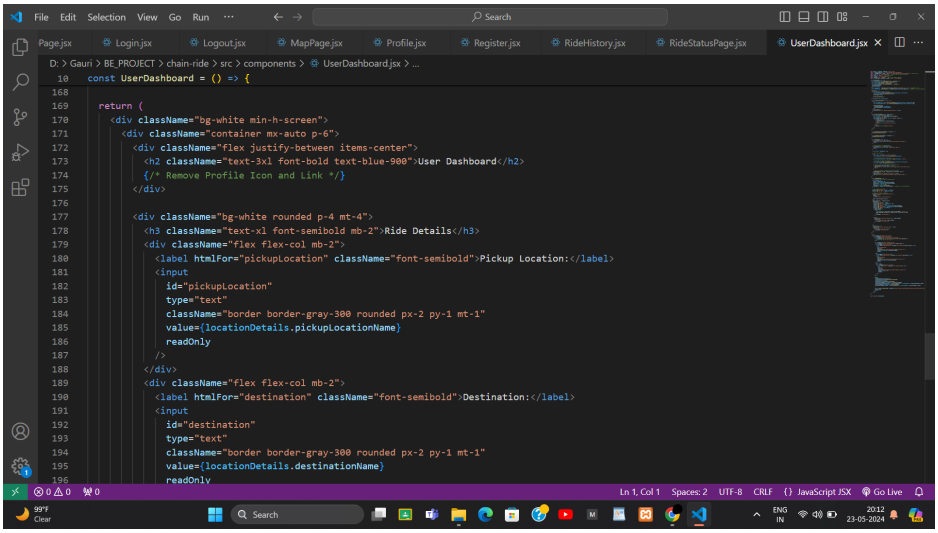


Figure 10.35: Screenshot of code of Ride Share

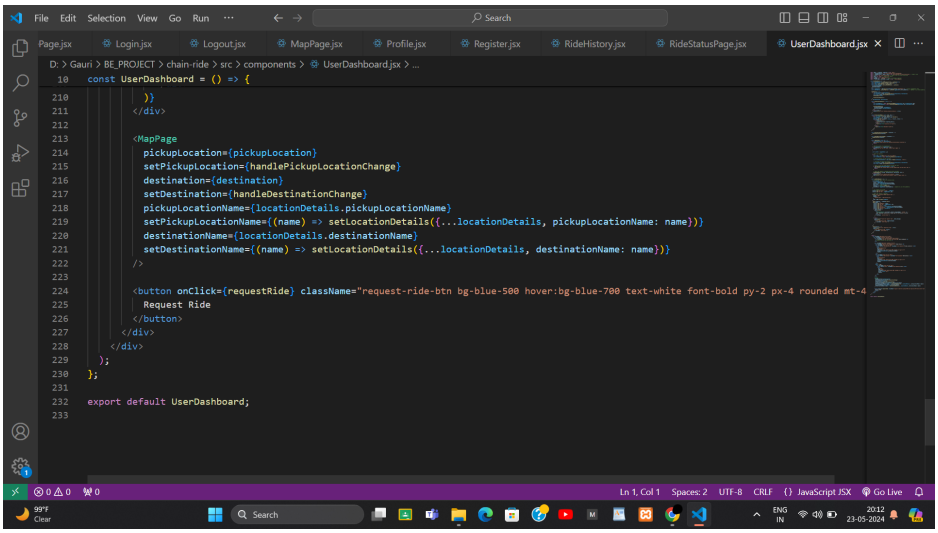


Figure 10.36: Screenshot of code of Login

10.2 OUTPUT

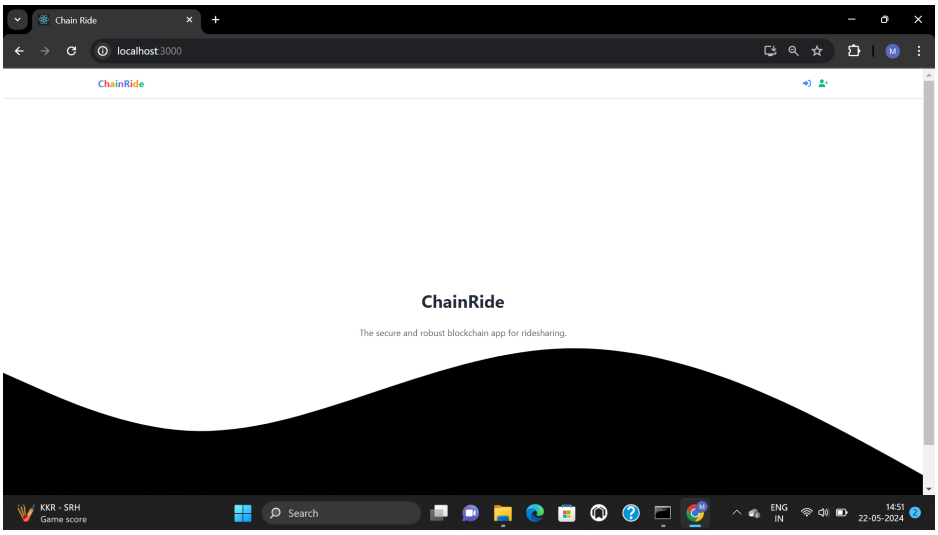


Figure 10.37: Home Page

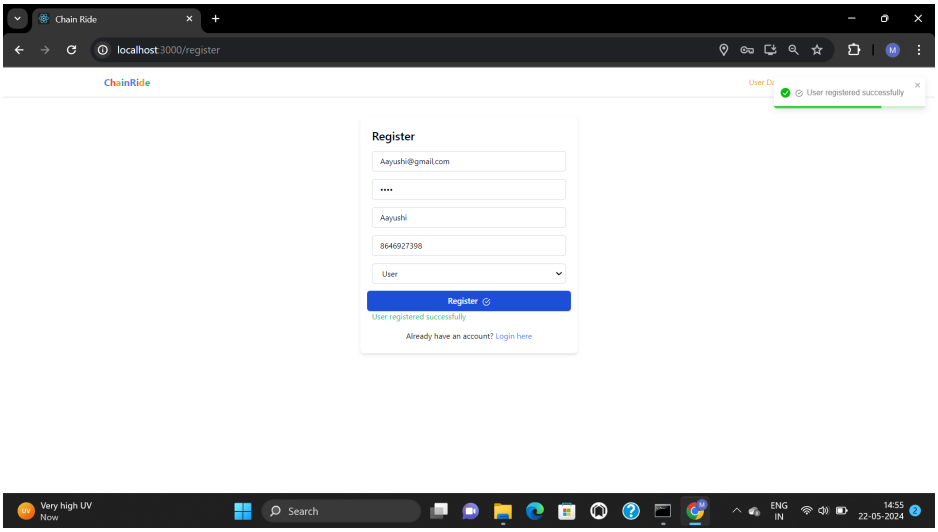


Figure 10.38: Registration Page

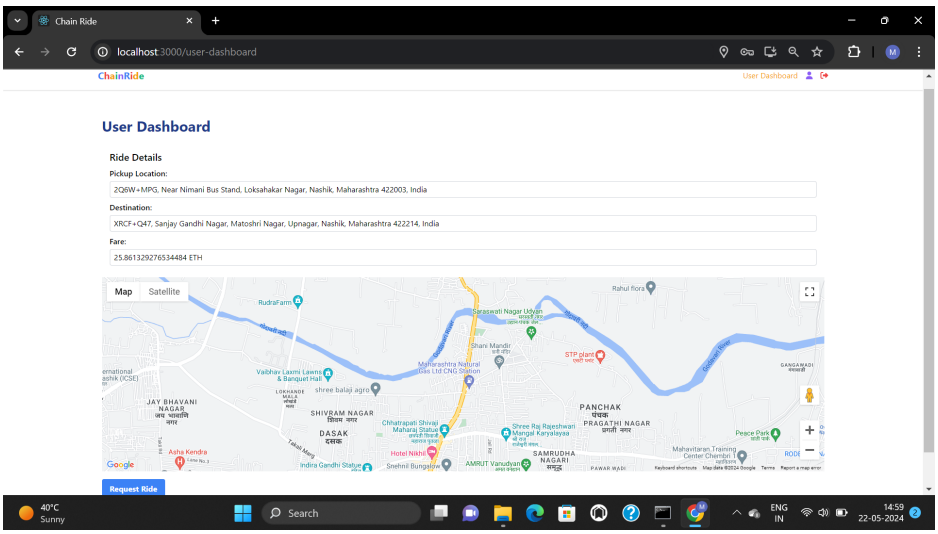


Figure 10.39: User Dashboard

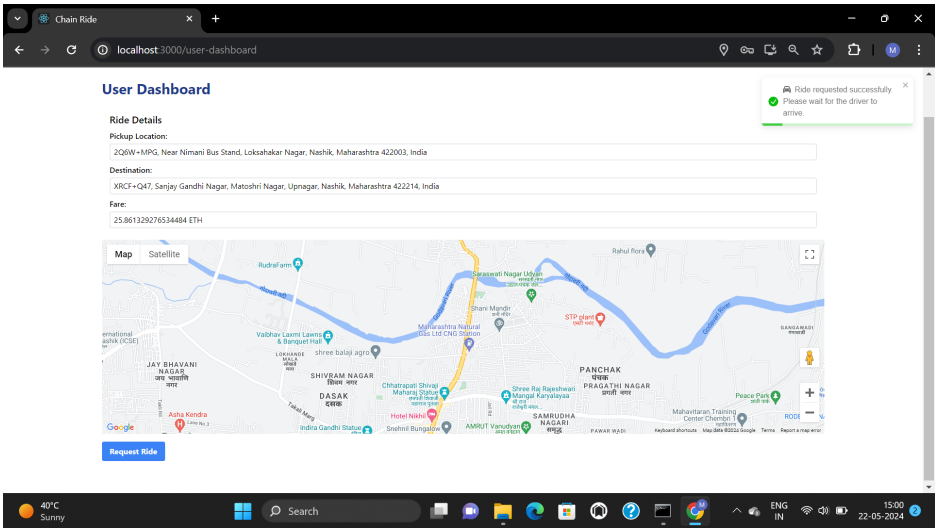


Figure 10.40: User Dashboard

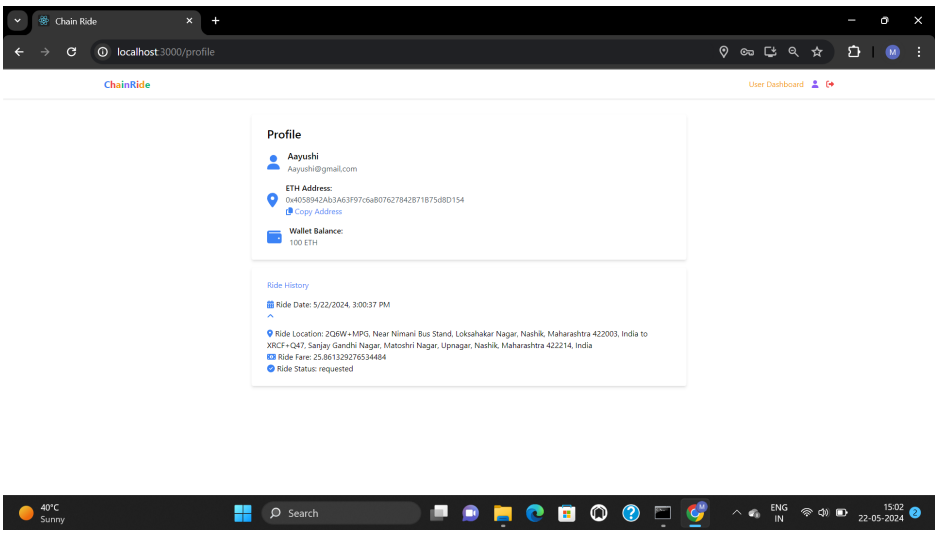


Figure 10.41: Profile Page

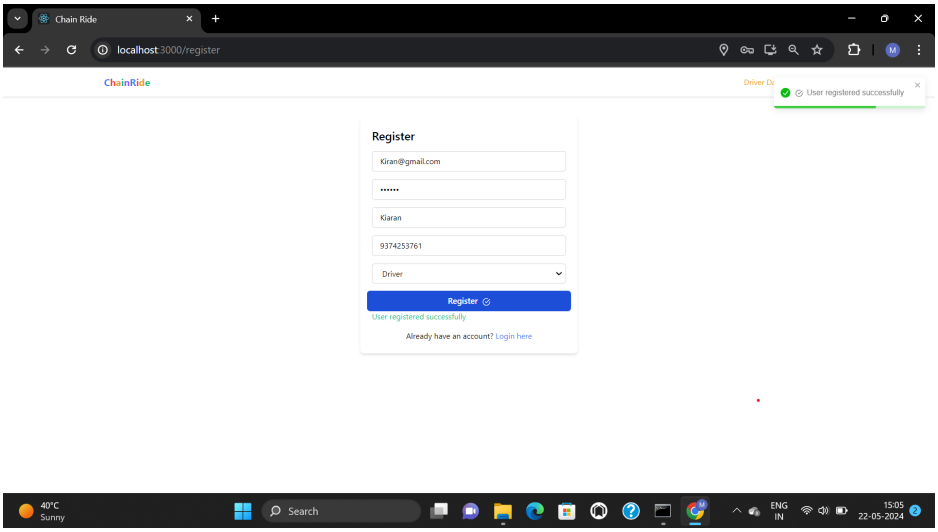


Figure 10.42: Registration Page

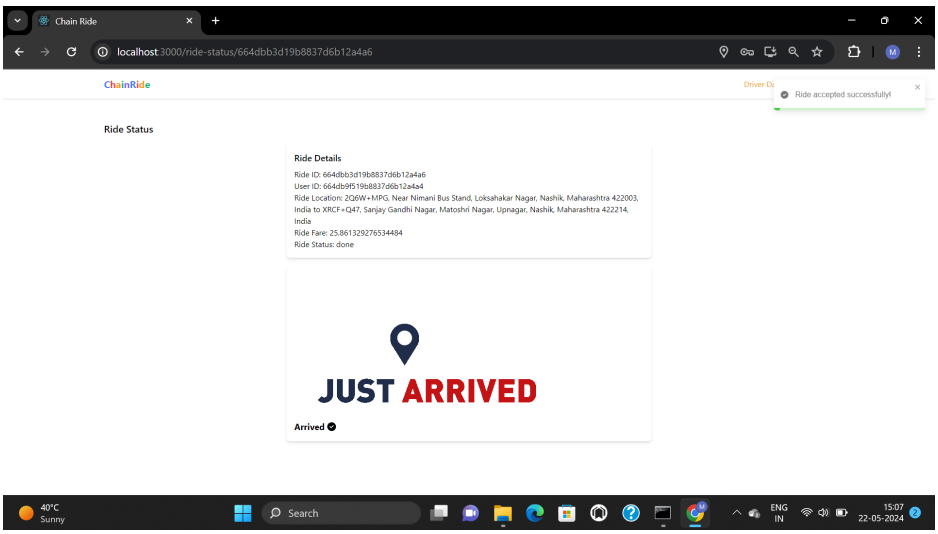


Figure 10.43: Ride Details

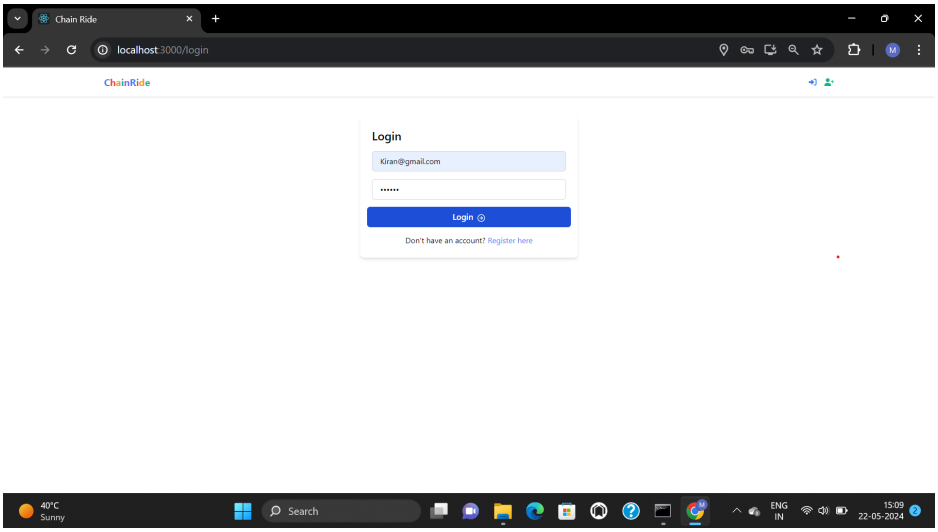


Figure 10.44: Login Page

Chapter 11

CONCLUSION AND FUTURE SCOPE

The working and system architecture of our peer to peer ridesharing application.This project finds an alternative to public transportation ,replacing it by peer to peer ride sharing. Our application tackles various issues involved in public transportation by using shortest path algorithms , p2p communication, optimal routing and machine learning based price recommendation. Similarly by addition of functions like payment gateway, realtime notification service this application can be revised. There can be many more functions like this which we hope to add in Car Chain Connect.

This restricts the users from being anonymous. An anonymous user shall be allowed to change its identity before the connected network without changing or resetting its gained reputation score and wallet values. The requirement is fulfilled with this architecture. A heavy system with a large number of peers will be prone to reach consensus slower than the required pace. In that case, that validator set can be selected as a subset of all the peers within a fixed region based on the contracted path. This will make the process faster as in this case we will be considering only a small subset of the peers.

Similarly, when the rider wishes to join a ride, he may get to see available only those drivers who are within a fixed radius of the riders start location. The rider may be ordered several fares for his opted path by the drivers and the rider may select any one of them based on the fare ordered and the reputation score gained by the driver. Hence, the views of the riders can be restricted to only nearest drivers with good rating based on the riders own rating. The problem of dispute resolution is hard in all practical cases as the fixing a route is only a coarse adjustment of a path. The ne tuning has to be done during the driving. So, mathematically it is hard to the difference between the ne tuning and detouring. Hence it is convenient to neighborhood of the contracted path and allow this ne tuning. The locus of neighborhood as described in this paper forms an ellipse contracted or chopped from both side along the path segment end

Chapter 12

REFERENCES

1. Sarvesh Wadi, "P2P Ride-Sharing using Blockchain Technology", September 2022
2. V. Buterin. Ethereum, April 2020, "A NEXT GENERATION SMART CONTRACT and DECENTRALIZED APPLICATION PLATFORM"
3. S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system", January 2019, SSRN Electronic Journal.
4. T. Report, "Filecoin: A cryptocurrency operated file storage network.", Dec 2018
5. MovieBloc, "Decentralized Movie and Content Distribution ", White Paper-Ver. 1.14 / APR. 2019.
6. Yong Yuan, Fei-Yue Wang, "Towards blockchain-based intelligent transportation systems" , 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC).
7. Surbhi Dhar, Sandra Arun, Vivek Dubey, Nilesh Kulal, "App for Ride Sharing",International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056 Volume: 07 Issue: 03, Mar 2020.
8. M. Baza, N. Lasla, M. M. Mahmoud ,G. Srivastava, M. Abdallah, "B-Ride: Ride Sharing with Privacy-preservation, Trust and Fair Payment atop Public Blockchain", IEEE Transactions on Network Science and Engineering.
9. Cai W., Wang Z., Ernst J. B., Hong Z., Feng C., Leung V. C. M. , "Decentralized Applications: The Blockchain-Empowered Software System", IEEE Access.
10. Panayiotis Christodoulou, Klitos Christodoulou, Andreou Andreas, "A Decentralized Application for Logistics: Using Blockchain in Real-World Applications", 2018 Cyprus Review, 30(2), 181-193.
11. David Schuff, Robert St Louis, "Centralization vs. Decentralization of Application Software", Communications of the ACM, June 2001, Vol. 44 No. 6, Pages 88-94.
12. A. Dorri, M. Steger, S. S. Kanhere, and R. Jurdak, "Blockchain: A distributed solution to automotive security and privacy", IEEE Communications Magazine, vol. 55, no. 12, pp. 119-125,2017.

13. HireGo, "HireGo - Decentralised Shared Mobility Platform," tech. rep., 2018.

14. T. Copel, Noam and Ater, "DAV White Paper," tech. rep., 2017.

15. Helbiz, "Helbiz Mobility System." Accessed: 2018-12-07.

16. F. Vogelsteller and V. Buterin, "Erc-20 token standard," Ethereum Foundation (Stiftung Ethereum), Zug, Switzerland, 2015.

17. FFQuest, "FFQuest - Blockchain marketplace for Car rental, Ride- sharing and Parking." Accessed: 2019-04-15.

18. WONO, "WONO White Paper," tech. rep., 2018.

19. W. Entriken, D. Shirley, J. Evans, and N. Sachs, "Non-Fungible Token Standard, document ERC-721, Sep. 2018." Accessed: 2018-12-07.

20. "Pinata - Add Files To IPFS Effortlessly." Accessed: 2019-04-15.