NEW HORIZON
COLLEGE OF ENGINEERING
Autonomous College Permanently Affiliated to VTU, Approved by AICTE & UGC
Accredited by NAAC with 'A' Grade, Accredited by NBA

A MINI PROJECT REPORT

for

Mini Project I (21CSE38A)

on

HERO HURTLE – Efficient Vehicle Booking

Submitted by

Chris Jordan J
USN: 1NH21CS062, Sem-Sec: 3-A

In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

Academic Year: 2022-23(ODD SEM)

ABSTRACT

The most reputable and renowned name in the travel industry when it comes to cab rental services is Online vehicle booking services. This is one of the most technologically advanced travel businesses, providing vehicle rentals and taxi services across the globe while fully utilizing information technology to increase our level of productivity.

This, however, is simply one component of the services as a whole. In addition, our initiative works hard to provide all of its customers with the greatest services possible, both in terms of people and technology. Additionally, this enterprise features a unique fleet of vehicles, ranging from luxurious to cost-effective auto mobiles.

While here, it also provides all corporate houses with online cab/bike hiring services. And this particular project makes the claim that it offers the best prices available, exactly tailored to the services used, and provides both intercity and intracity taxi services.

All the vehicles involved in this project have the necessary licenses and papers, ensuring that customers won't ever be pressurized for not having the required paperwork.

However, this project has a well-thought-out backup plan in case something goes wrong. Taxi drivers are highly educated, extremely courteous, dependable, and well-trained to deal with sudden breakdowns. The cab service offers all types of vehicles, from luxurious to economical. Furthermore, upholding quality standards is the top concern for this project. Vehicles are constantly inspected and tested for optimal and uninterrupted performance in order to do this. This technology can create itineraries that fit any traveler's preferences and budget thanks to a strategic team of industry pros.

Furthermore, a professional team with well-trained drivers and administrative employees is employed to ensure the highest quality requirements are met.

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task would be impossible without the mention of the people who made it possible, whose constant guidance and encouragement crowned our efforts with success.

I have great pleasure in expressing gratitude to **Dr. Mohan Manghnani**, Chairman, New Horizon Educational Institutions, for providing necessary infrastructure and creating good environment.

I take this opportunity to express my profound gratitude to **Dr. Manjunatha,** Principal, New Horizon College of Engineering, for his constant support and encouragement.

I would like to thank **Dr. Anu**, Professor and Dean-Academics, NHCE, for her valuable guidance.

I would also like to thank **Dr. B. Rajalakshmi**, Professor and HOD, Department of Computer Science and Engineering, for her constant support.

I also express my gratitude to **Dr. D. Roja Ramani**, Associate Professor, Department of Computer Science and Engineering, my mini project reviewer, for constantly monitoring the development of the project and setting up precise deadlines. Her / His valuable suggestions were the motivating factors in completing the work.

Name: CHRIS JORDAN J

USN: 1NH21CS062

CONTENTS

ABSTRACT **I**

ACKNOWLEDGEMENT **II**

LIST OF FIGURES **V**

1. INTRODUCTION

1.1. PROBLEM DEFINITION **1**

1.2. OBJECTIVES **1**

1.3. METHODOLOGY **2**

1.4. EXPECTED OUTCOMES **2**

1.5. HARDWARE AND SOFTWARE REQUIREMENTS **2**

2. DATA STRUCTURES

2.1. SINGLY LINK LIST 3

2.1.a. DOUBLY LINKED LIST **4**

2.1.b. CIRCULAR LNKED **4**

2.2. STACKS **4**

2.3. GRAPHS **5**

2.4. QUEUES **6**

3. DESIGN

3.1. DESIGN GOALS **7**

4. PSEUDOCODE **7**

5. FLOWCHART **8**

6. IMPLEMENTATION

6.1. MAIN FUNCTION **8**

6.2. SERVER FUNCTION **9**

6.3. CLIENT FUNCTION **10**

6.4.1. APPEND RIDES **11**

6.4.2. DISPLAY RIDES **12**

6.4.3. REMOVE RIDES **13**

6.4.4. CHOOSE RIDES **14**

7. RESULTS

7.1. SERVER OPTION **18**

7.2. INPUTTING A NEW RIDER **19**

7.3. CLIENT OPTION **19**

7.4. UI (INDIVIDUAL) **20**

8. CONCLUSION **20**

9. REFERENCES **21**

LIST OF FIGURES

Figure No	Figure Description	Page No
2.1.1	SINGLY LINKED LIST	3
2.2.1	STACKS	5
2.3.1	GRAPHS	6
2.4.1	QUEUE	6
5.1	FLOWCHART	10

CHAPTER 1

INTRODUCTION

1.1 PROBLEM DEFINITION

The goal of the HERO HURTLE project is to provide an online system for vehicle reservations. The current system is time-consuming, partially manual, and non-standardized in terms of full returns. Customers and clients are currently having a lot of issues with the arrival of the cab, as well as occasionally with the termination of the journey and the payment of the cost. A programme that has been streamlined as part of this vehicle booking project may be able to address several issues. Therefore, customers do not need to waste time calling to reserve the reserved vehicle.

The idea does provide a conclusive answer. Additionally, it is an ajar issue. In order to discover a good solution with the greatest amount of simplicity and scalability, the problem is being attempted to be solved.

The clients can rapidly search for drivers and book vehicles thanks to this project. Now, it's not always certain that a cab will be available when you try to book one, so you must periodically check to see if one becomes available if none are available when you try to book one. It is always available for immediate booking.

1.2 OBJECTIVES

The project's primary goal is to assist the client and server in making cab reservations and hiring drivers. We must first insert a few drivers using the server function in order to accomplish this. The driver's name and age are gathered as we input. The position, rating, distance travelled, and number of rides taken by the driver are initially set to NULL in the driver's database. The database's driver list can have any number of drivers added to it. The server has the same authority to remove the most recent driver from the list. The server has access to all driver details in the list.

The information includes the driver's name, age, the number of rides he has provided, his rating, his location, and the distance he has travelled. All of the available drivers' information will be displayed using the display function. By entering the distance to be travelled, the customer can make a cab reservation. I'm using the delay feature to give us a realistic impression that a cab is on the way to pick you up by delaying the process for a few seconds. The customer has the option to continue the ride or end it once the driver has arrived. The distance travelled must be manually inputted after the ride. The cost of the trip is computed and shown. Later the consumers can rate the drivers 1 through 5.

1.3 METHODOLOGY

The various methodologies used the project are, the data structure linked list. It would be borderline impossible to have to write the code without the implementation of linked list. Dynamic memory allocation enables, to effectively make use of the memory locations as pointers. Arrays are used to define some of the categories. Efficient use of functions in the program helps reduce the code complexity, invoking the already declared function enables for a faster execution, user defined functions such as structures were also used in the project.

1.4 EXPECTED OUTCOMES

1. This project was primarily built to be an efficient booking software without the trouble of one wanting to call and reconfirm the ride

2. The client and server both offer a variety of classes to pick from. By gathering the driver's basic information, such as name, age, aadhar number, etc, the server can add them into the database. This was primarily done to ensure safety to the clients in unfortunate events.

3. The server has the ability to delete a driver and view the database's list of drivers. After inputting the desired distance, the client can browse through the list of drivers and select one for their ride.

4. After the journey is scheduled, the driver arrives at the client's location. The customer has the option to begin or end the ride once the driver arrives at their destination. A penalty is generated if the ride is cancelled.

1.5 HARDWARE AND SOFTWARE REQUIREMENTS

HARDWARE REQUIREMENTS

1) Personal computer
2) Minimum of 2GB RAM
3) 64-bit operating system
4) Processor - x86 Compatible processor, 1.7 GHz Clock speed

SOFTWARE REQUIREMENTS

1) Operating system - Windows 7 or above
2) Atom editor and GCC compiler

CHAPTER 2

DATA STRUCTURES

2.1 LINKED LIST

A linked list is a group of randomly stored items in memory, collectively referred to as nodes. A node has two fields: the data that is stored there and a pointer that points to the node in memory that comes after it. Pointer to the null is contained in the list's last node.

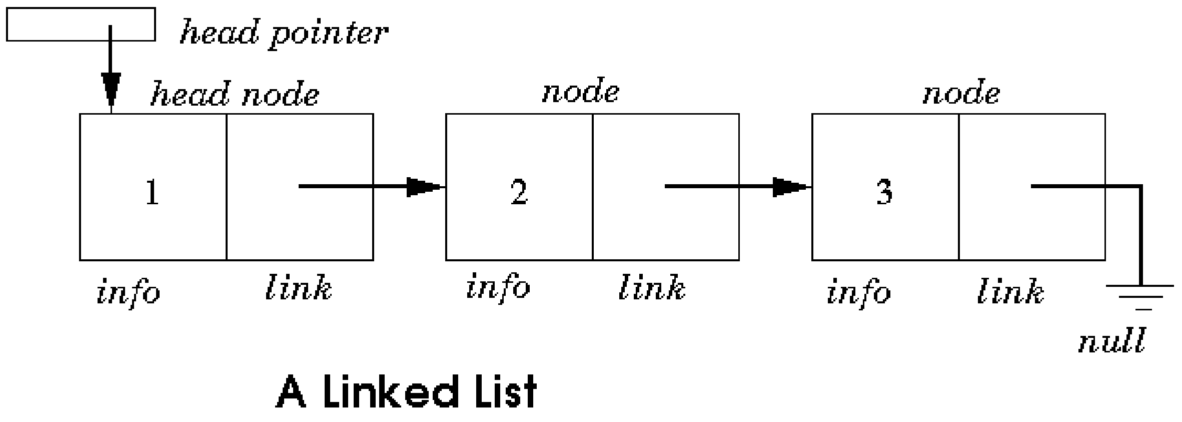
Every Item belongs to the List as a Node. In our function, we keep the Pointer to the first Item in memory so that we can later utilise it in another variable called current pointer to obtain the data for all of our Nodes without losing it. And also, some of the major disadvantages faced while implementing arrays such as, code execution time, etc. can easily be tackled by using linked list.

All dynamic memory allocation functions, including calloc, malloc, realloc, and deallocation function free, must be used in linked lists (). A block of memory from the heap is allocated by each dynamic memory allocation function.

Syntax – A linked list in C can be written using structures and pointers

```
struct linked_list{
    int variable;
    struct linked_list *next;
    };
```

Each node in the list is made using the aforementioned definition. The element is stored in the data field, and the following is a pointer to the address of the following node.



(Fig- 2.1.1)

SOME OF THE SUB CATAGORIES OF LINKED LISTS ARE:

a) Doubly linked list
b) Circular linked list
c) Circular doubly linked list

2.1.a DOUBLY LINKED LIST

A node in a doubly linked list carries a pointer to both the preceding and following node in the sequence, making it a more complex sort of linked list. Consequently, a node in a doubly linked list has three components: node data, a pointer to the node after it in the sequence (the next pointer), and a pointer to the node before it (previous pointer). The graphic depicts a sample node in a doubly linked list.

The syntax to declare a structure of a node for double linked list can be done as follows:

```
struct node
{
    struct node *prev;
    int data;
    struct node *next;
}
```

2.1.b CIRCULAR LINKED LIST

Using a circular, a pointer to the list's first node is contained in the list's last node of a singly linked list. Circular doubly linked lists and singly linked lists can both be circular.

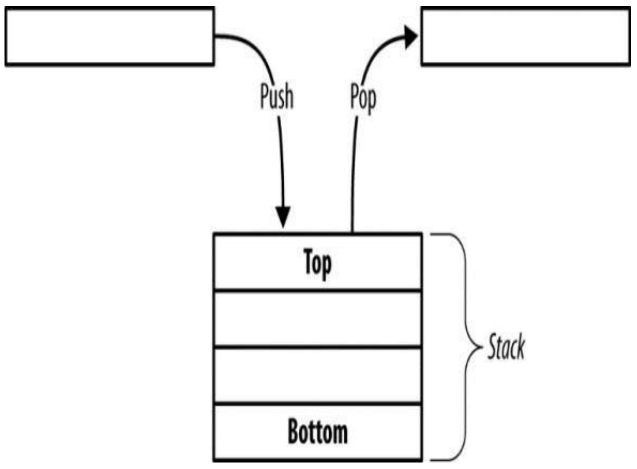
Until we get at the initial node, we move around a circular singly linked list. There is no beginning or conclusion to the circular list of single favourites. The following section of none of the nodes contains a null value.

There are multiple operations that can be performed using this data structure some the examples are, insertion at the beginning or end, deletion at the beginning or end, searching, traversing etc.

2.2 STACKS (general)

It is a linear data structure that carries out the operations in a specific order. Since, we can only access the elements on the top of the stack, maintaining the pointer to the top of the stack, which is the last element to be placed, is necessary to implement the stack.

Stacks data structures follows a strategy called the LIFO(Last In First Out). According to this method, the element that was added last will appear first. As an actual illustration, consider a stack of books stacked on top of one another. We can claim that the book we put last comes out first because the book we put last is on top and we remove the book at the top.



(Fig- 2.2.1)

There are two types of stacks there are;

REGULAR STACK - This kind of stack, which is also a memory component in the memory unit, is limited to handling little data. Because the capacity of the register stack is so little in comparison to the memory, its height is always constrained.

MEMORY STACK - A lot of memory data can be handled by this kind of stack. The memory stack can be any height because it uses a lot of memory space.

Some of the basic primitive operations performed with stacks are

1) Push ()
2) Pop ()
3) Display ()

2.3 GRAPHS

A graph is a type of data structure made up of the following two elements:

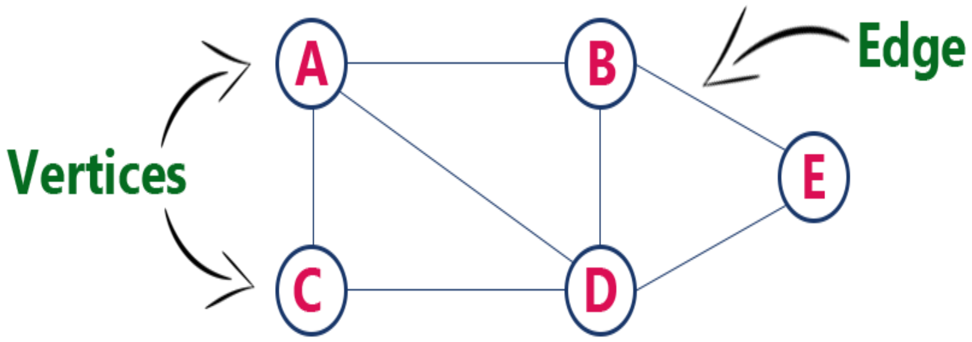
1. A node is a group of finite vertices.

2. An edge is a finite set of ordered pairs of the form (u, v).

Since (u, v) is not the same as (v, u) in the case of a directed graph, the pair is ordered (di-graph). There is an edge from vertex u to vertex v, as indicated by the pair of the form (u, v). The edges might have weight, worth, or cost.

In many real-life applications, graphs are employed as representations: Networks are depicted using graphs. The networks could be telephone, circuit, or city-wide path networks. Graphs are also utilised in social media sites like Instagram, twitter, etc. where the nodes are the person's information.

Some of the examples where graphs are used are, Bellman-Ford's algorithm, Kruskals algorithm, Dijkstras algorithm, and so on.



(Fig- 2.3.1)

2.4 QUEUES

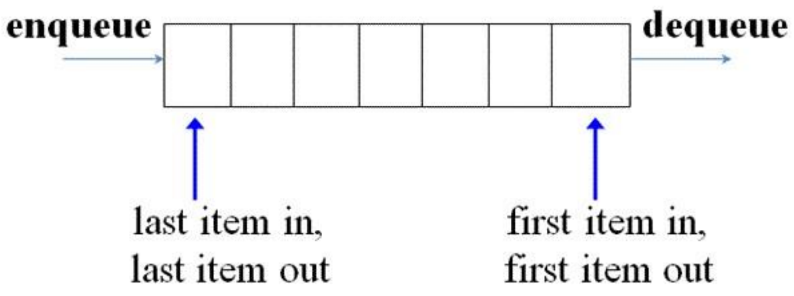
In C, a queue is essentially a linear data structure for managing and storing data components. First In, First Out is the sequence that it follows (FIFO). The first element added to an array in a queue is also the first element withdrawn from the array. Let's use the example of a ticket booth for buses as an example. Here, a C programming queue style is utilised. The first person to arrive will be issued with the tickets, as they are given out on a first-come, first-served basis. There is an open line at both ends. There are two ends: one for inserting data and the other for removing it. You can implement a queue in any programming language, even C.

Some of the examples of queue are,

Input restricted queue, Output restricted queue, Circular queue, Double ended queue and Priority queue

Simple operations performed with queues are:

1) **INSERTION** - The act of adding a new element to the queue is known as insertion, and each time, the rear is incremented before the item is added.
2) **DELETION** - The front element's removal from the queue is referred to as deletion, and following deletion, the front pointer must be increased.
3) **DISPLAY** – is used to display all the contents/elements embedded in the queue.



(Fig- 2.4.1)

3. DESIGN GOALS

- The main objective held in mind while the project was developed was to ensure a very highly user-friendly server for the clients to easily interact with the program.
- This program also helps to reduce the redundant data stored as, it being simple to use, there exists little to no data exchange.

4. PSEUDO CODE

```
1. Start
2. Enter one of the three presented options
    a) Owner
    b) Customer
    c) Exit
3. If the owner is selected, he'll be hit with 4 options
    a) Remove
    b) Add
    c) Display
    d) Exit
```

Remove is to remove a particular ride, add is to add one and display is to display the already appended options of rides.

```
4. When option 2, as in customer is selected,
```

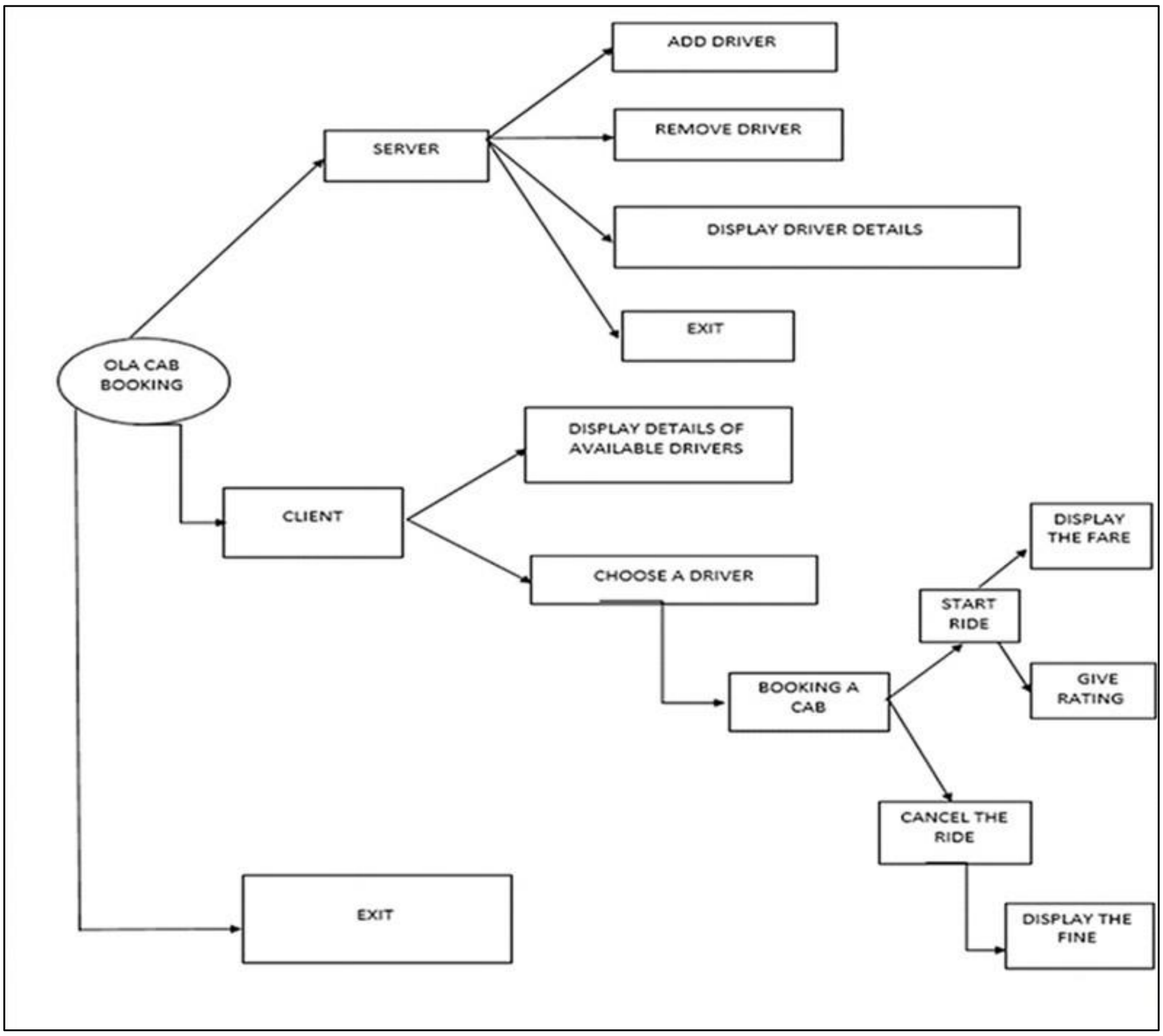
The customer is now into the portal where they can either

```
    a) View
    b) Select
    c) Exit
```

If the former is selected the can view all of the available rides at the moment, and if the latter is selected they can choose and book a ride.

```
5. From then on, they can input the total distance in kilometres, which will then
   automatically calculate a reasonable price
6. The customer can rate their performance. Which gets appended into their
   database/profile.
7. Stop.
```

5. PICTORIAL REPRESENTATION OF THE PROJECT.



6. FUNCTIONALITY

6.1 Main function

```
void main()
{
  int choice=0;
  int statement=0;
  List_t *List=NULL;
  List=(List_t*)malloc(sizeof(List_t*));
  List=List_initilize(List);
  while(1)
  {
    choice=0;
    clrscr();
    printf("\nARE YOU A CLIENT OR SERVER\n1-CLIENT\n2-SERVER\n3-EXIT THE
PROGRAM\n");
    printf("\nENTER YOUR CHOICE:");
    scanf("%d",&choice);
    switch(choice)
    {
      case 1:Client_function(List);
             break;
      case 2:Server_function(List);
             break;
      default:printf("\nINVALID STATEMENT,EXITING THE PROGRAM BYE BYE!");
              statement=1;
              break;
     }
    if(statement==1)
    {
      break;
    }
  }
}
```

6.2. SERVER FUNCTION

```
void Server_function(List_t *list)
{
  int choice1=0;
```

```
 printf("ENTER YOUR CHOICE\n 1-ADD A RIDER\n 2-REMOVE A RIDER \n 3-DISPLAY THE
RIDERS AND THEIR PERFORMANCE \n 4-Exit\n CHOICE:");
 scanf("%d",&choice1);
 switch(choice1)
 {
  case 1:Add_Rider(list);
        break;
  case 2:Remove_Rider(list);
        break;
  case 3:Display_Rider(list);
        break;
  default:printf("\nNO MORE OPERATIONS");
         break;
 }
}
```

6.3. CLIENT FUNCTION

```
void Client_function(List_t *list)
{
 int choice1=0;
 printf("ENTER YOUR CHOICE\n 1-DISPLAY THE RIDERS\n 2-CHOOSE A RIDER\n 3-Exit\n
CHOICE:");
 scanf("%d",&choice1);
 switch(choice1)
 {
  case 1:Display_Rider(list);
        break;
  case 2:Choose_Rider(list);
        break;
  default:printf("\nSORRY TO HEAR THIS! HOPE YOU VISIT US AGAIN");
         return;
 }
}
```

6.4.1 APPEND RIDES – INSERTION OF EACH NODE

```
void Add_Rider(List_t *list)
{
 Rider_t *NewRider=NULL;
 char Name[20];
 int Age=0;
 int phone_number = 0;
 int aadhar = 0;
 NewRider=(Rider_t *)malloc(sizeof(Rider_t));
 printf("\nENTER THE NAME OF THE RIDER:");
 scanf("%s",&Name);
 strcpy(NewRider->Name,Name);
 printf("\nENTER THE AGE OF THE RIDER:");
 scanf("%d",&Age);
 printf("\nENTER THE PHONE NUMBER OF THE RIDER:");
 scanf("%d",&phone_number);
 printf("\nENTER THE AADHAR OF THE RIDER:");
 scanf("%d",&aadhar);
 NewRider->Age=Age;
 NewRider->phone_number=phone_number;
 NewRider->aadhar=aadhar;
 NewRider->Rating=0;
 NewRider->Number_of_rides=0;
 NewRider->Distance_travelled=0;
 NewRider->link=NULL;
 if(list!=NULL)
 {
    list->head=NewRider;
    list->total_number=1;
  }
  else
  {
    Rider_t *temp=list->head;
    while(temp->link!=NULL)
    {
      temp=temp->link;
    }
    if(temp->link==NULL)
    {
    temp->link=NewRider;
    list->total_number+=1;
    }
  }
}
```

6.4.2 Display your rides – Traversing each node

```
void Display_Rider(List_t *list)
{
  if(list==NULL)
  {
    printf("\nTHERE ARE NO RIDERS\n\n\n");
    getch();
  }
  else
  {
    int count=1;
    Rider_t *temp;
    temp=list->head;
    while(temp!=NULL)
    {
      printf("\nNAME OF THE RIDER IS : %s\n",temp->Name);
      printf("AGE OF THE RIDER IS :%d\n",temp->Age);
      printf("PHONE NUMBER OF THE RIDER IS :%d\n",temp->phone_number);
      printf("AADHAR OF THE RIDER IS :%d\n",temp->aadhar);
      printf("RATING OF THE RIDER IS :%d\n",temp->Rating);
      printf("NUMBER OF RIDES BY THE RIDER IS :%d\n",temp->Number_of_rides);
      printf("DISTANCE TRAVELLED BY THE RIDER IS :%d\n",temp->Distance_travelled);
      printf("THE POSITION OF THIS IS %d\n____________\n\n\n",count);
      getch();
      count+=1;
      temp=temp->link;
    }
  }
}
```

6.4.3. Remove Rider

```
void Remove_Rider(List_t *list)
{
  if(list->head==NULL)
  {
    printf("\nTHERE ARE NO RIDERS TO REMOVE");
  }
  else if(list->head->link==NULL)
  {
    printf("\nTHE LAST RIDER");
    list->head=NULL;
    list->total_number=0;
```

```
 }
 else
 {
   Rider_t *temp=list->head;
   Rider_t *prev=NULL;
   while(temp->link!=NULL)
   {
     prev=temp;
     temp=temp->link;
   }
   prev->link=NULL;
   free(temp);
   list->total_number-=1;
 }
}
```

6.4.4. Choose Rider

```
void Choose_Rider(List_t *list)
{
 Display_Rider(list);
 if(list->head==NULL)
 {
   printf("\nTHERE ARE NO RIDERS TO CHOOSE\n\n\n");
 }
 else if(list->head->link==NULL)
 {
   Rider_t *Rider=list->head;
   int i;
   int journey=0;
   int distance=0;
```

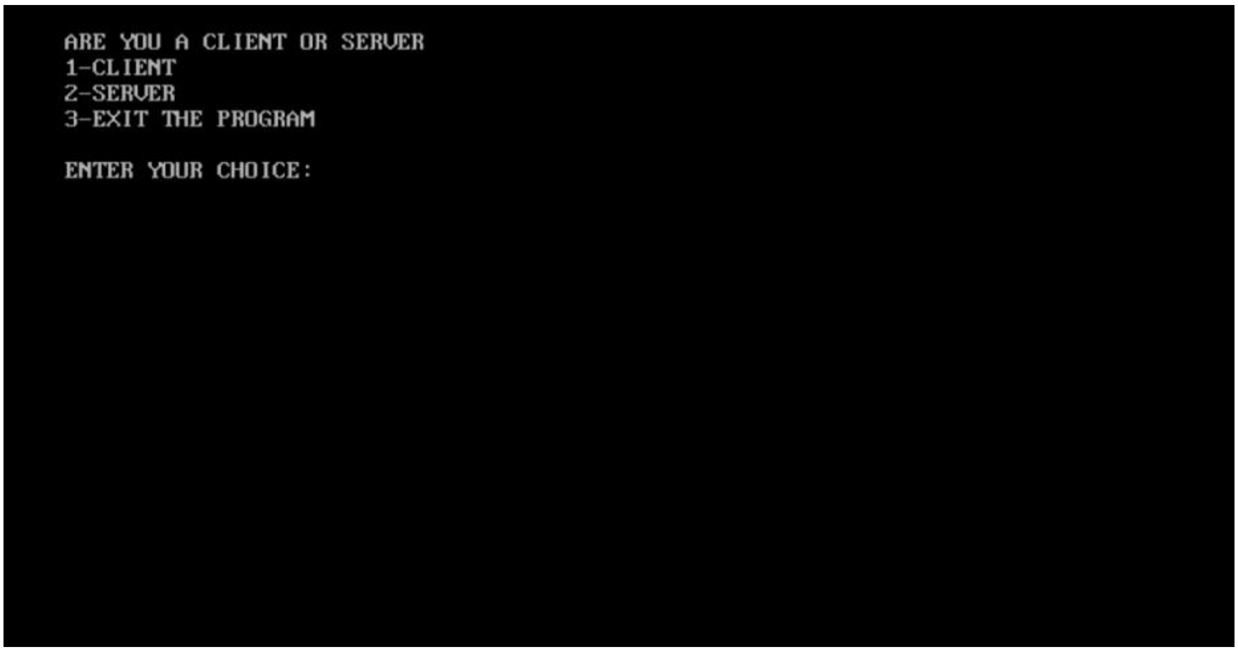
```
  int rating=0;
  printf("\nTHERE IS ONLY ONE RIDER TO CHOOSE SO YOU HAVE TO CHOOSE HIM\n");
  printf("ENTER THE DISTANCE TO BE TRAVLLED :\n");
  scanf("%d",&distance);
  printf("THE RIDER IS ON HIS WAY\n");
  for(i=0;i<10;i++)
  {
    printf("#");
    delay(1000);
  }
  printf("THE RIDER HAS SUCCESSFULLY REACHED YOUR LOCATION\n");
  printf("DO YOU WANT TO START THE JOURNEY?\n1-YES\n0-NO");
  scanf("%d",&journey);
  if(journey==0)
    {
          printf("\nSORRY TO HEAR THIS,HAVE A GOOD DAY\nNOTE YOU WILL BE FINED FOR
CANCELLING THE RIDE ON YOUR NEXT RIDE");
          return;
    }
  else
    {
          int i;
          int expense=13;
          int totalcost=expense*distance;
          printf("\nTHE COST OF THE RIDE FOR EACH KM IS %d",expense);
          printf("\nYOUR JOURNEY HAS STARTED");
          for(i=0;i<distance;i++)
          {
            printf("#");
            delay(1000);
```

```
         }
         printf("\n");
         printf("\nYOUR JOURNEY HAS FINISHED");
         printf("\nTHE COST OF THE RIDE IS %d",totalcost);
         Rider->Distance_travelled+=distance;
         Rider->Number_of_rides+=1;
         printf("\nENTER THE RATING OF THE RIDE:");
         scanf("%d",&rating);
         Rider->Rating=((Rider->Rating*((Rider->Number_of_rides)-1)+rating)/Rider-
>Number_of_rides);
     }
   }
   else
     {
         int journey=0;
         int distance=0;
         int rating=0,i;
         Rider_t *Rider=list->head;
         int pos=0;
         printf("ENTER THE POSITION OF THE RIDER YOU WANT TO CHOOSE");
         scanf("%d",&pos);
         if(pos>list->total_number)
         {
           printf("SORRY YOU HAVE CHOOSEN AN INVALID POSITION\n");
           return;
         }
         for(i=1;i<pos;i++)
         {
           Rider=Rider->link;
         }
```

```
printf("\n\n\n\nTHE RIDER NAME IS %s\n",Rider->Name);
printf("ENTER THE DISTANCE TO BE TRAVLLED:\n");
scanf("%d",&distance);
printf("THE RIDER IS ON HIS WAY\n");
delay(5000);
printf("THE RIDER HAS SUCCESSFULLY REACHED YOUR LOCATION\n");
printf("DO YOU WANT TO START THE JOURNEY?\n1-YES\n0-NO");
scanf("%d",&journey);
if(journey==0)
 {
   printf("\nSORRY TO HEAR THIS,HAVE A GOOD DAY\nNOTE YOU WILL BE FINED FOR
CANCELLING THE RIDE ON YOUR NEXT RIDE");
   return;
 }
else
 {
   int i;
   int expense=15;
   int totalcost=expense*distance;
   printf("\nTHE COST OF THE RIDE FOR EACH KM IS %d",expense);
   printf("\nYOUR JOURNEY HAS STARTED");
   for(i=0;i<distance;i++)
   {
     printf("#");
     delay(1000);
   }
   printf("\n");
   printf("\nYOUR JOURNEY HAS FINISHED");
   printf("\nTHE COST OF THE RIDE IS %d",totalcost);
   Rider->Distance_travelled+=distance;
```

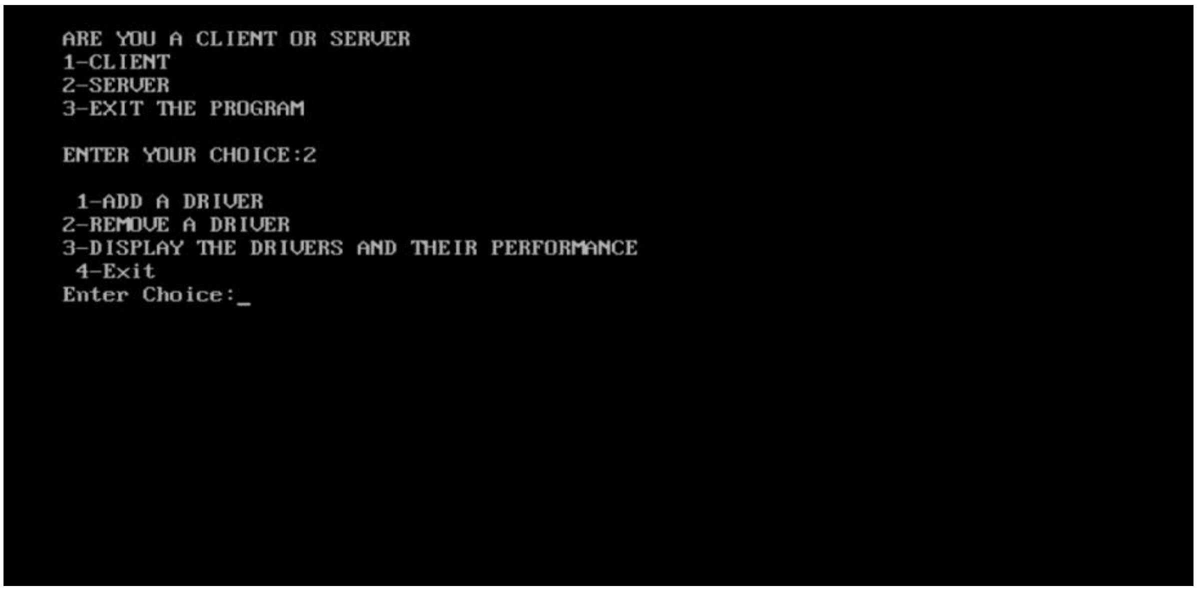
```
            Rider->Number_of_rides+=1;

            printf("\nENTER THE RATING OF THE RIDE");

            scanf("%d",&rating);

            Rider->Rating=((Rider->Rating*((Rider->Number_of_rides)-1)+rating)/Rider-
>Number_of_rides);

          }

     }

}
```

6. OUTPUT



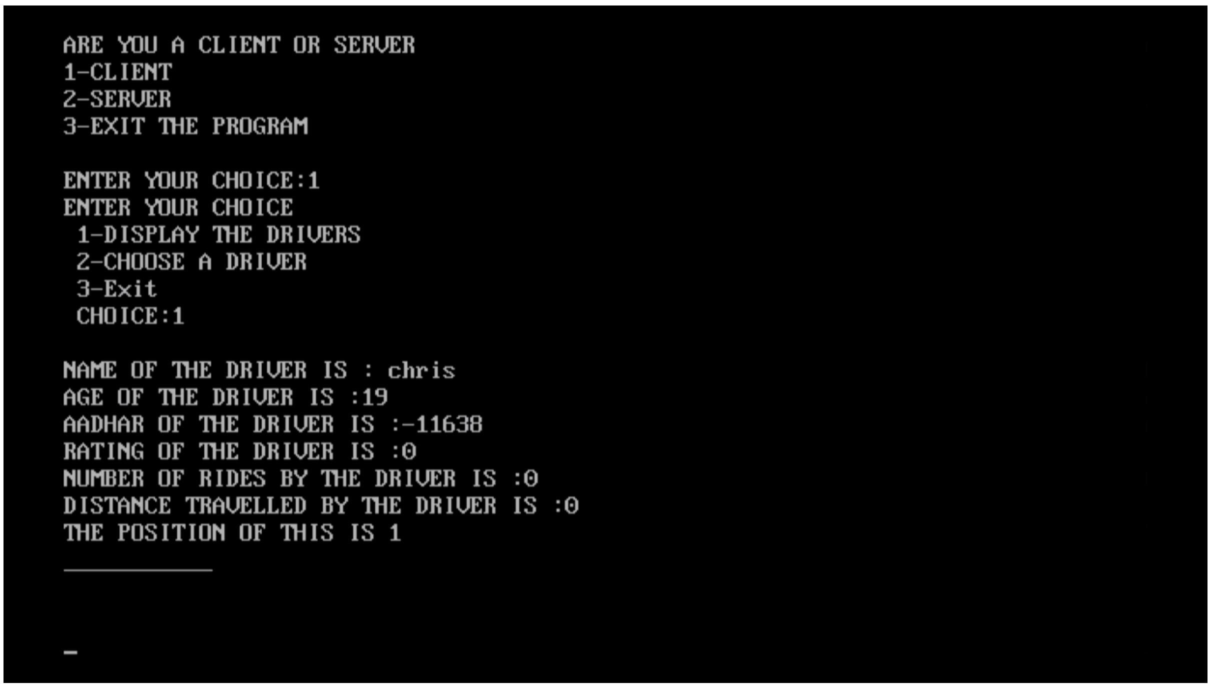
```
ARE YOU A CLIENT OR SERVER
1-CLIENT
2-SERVER
3-EXIT THE PROGRAM

ENTER YOUR CHOICE:
```

(Fig - 7.1)



```
ARE YOU A CLIENT OR SERVER
1-CLIENT
2-SERVER
3-EXIT THE PROGRAM

ENTER YOUR CHOICE:2

 1-ADD A DRIVER
2-REMOVE A DRIVER
3-DISPLAY THE DRIVERS AND THEIR PERFORMANCE
 4-Exit
Enter Choice:_
```

(Fig – 7.2)

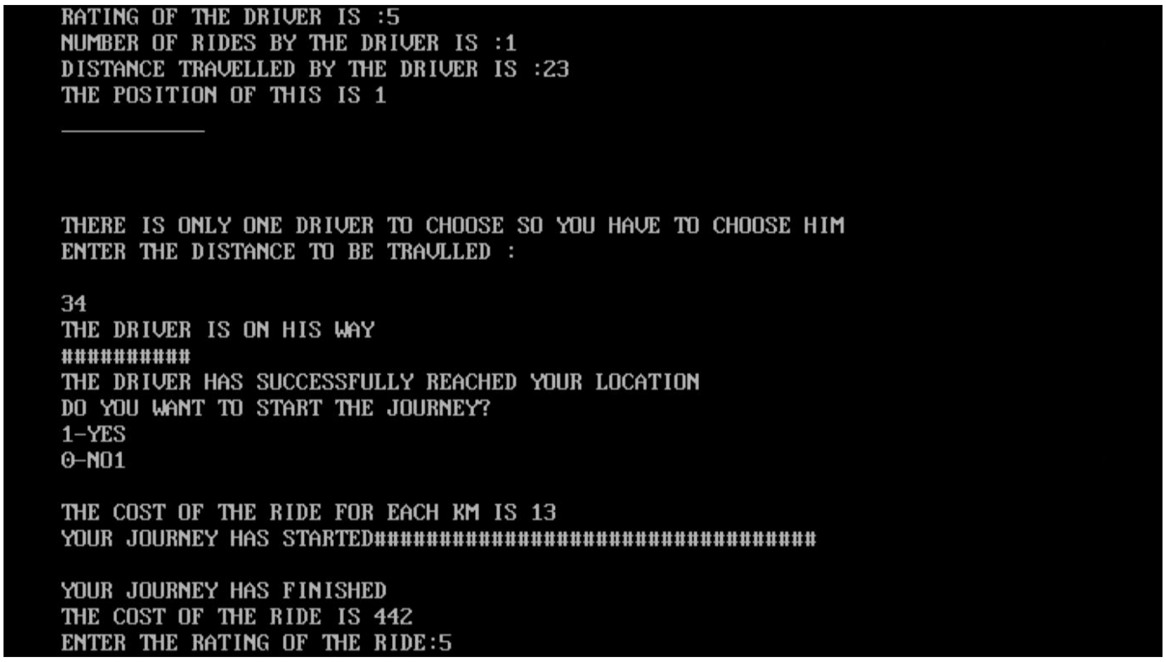


```
ARE YOU A CLIENT OR SERVER
1-CLIENT
2-SERVER
3-EXIT THE PROGRAM

ENTER YOUR CHOICE:1
ENTER YOUR CHOICE
 1-DISPLAY THE DRIVERS
 2-CHOOSE A DRIVER
 3-Exit
 CHOICE:1

NAME OF THE DRIVER IS : chris
AGE OF THE DRIVER IS :19
AADHAR OF THE DRIVER IS :-11638
RATING OF THE DRIVER IS :0
NUMBER OF RIDES BY THE DRIVER IS :0
DISTANCE TRAVELLED BY THE DRIVER IS :0
THE POSITION OF THIS IS 1
___________


_
```

(Fig – 7.3)

Once the server has appended all the necessary details of the rider into the database it gets stored, and can be mined when required for the next step which is the customer.

The customer then inputs the relevant answers about their ride, depicted in the picture above.



```
RATING OF THE DRIVER IS :5
NUMBER OF RIDES BY THE DRIVER IS :1
DISTANCE TRAVELLED BY THE DRIVER IS :23
THE POSITION OF THIS IS 1
___________



THERE IS ONLY ONE DRIVER TO CHOOSE SO YOU HAVE TO CHOOSE HIM
ENTER THE DISTANCE TO BE TRAVLLED :

34
THE DRIVER IS ON HIS WAY
##########
THE DRIVER HAS SUCCESSFULLY REACHED YOUR LOCATION
DO YOU WANT TO START THE JOURNEY?
1-YES
0-NO1

THE COST OF THE RIDE FOR EACH KM IS 13
YOUR JOURNEY HAS STARTED##################################

YOUR JOURNEY HAS FINISHED
THE COST OF THE RIDE IS 442
ENTER THE RATING OF THE RIDE:5
```

(Fig - 7.4)

Once the journey has ended, people can vote and update the feedback which is used to show the rating, and is appended into the riders database.

7. CONCLUSION

This project was predominantly aimed to achieve what other online vehicle booking systems have already done but with added new features and by putting the concept of linked list into work. As we all know that linked list is extremely efficient in traversing, this concept was used. Some of the new features such as, reduction of data redundancy, a very easy and interactive interface, with reinforced security systems. As the phone number and aadhar number of the rider's are displayed to the customer, they can also be used to investigate, in the case of any unfortunate events. Since there is very little to no data exchange between the customer and the rider, hence leads to lower data redundancy. The main goal of this project was to implement two wheelers, as it is easier to cut through the ever-increasing traffic.

When a new rider is created in the program, a completely new node is created. Using the concept of linked list we could also implement a number of its features to perfect this project.

8. REFERENCES

1. https://www.geeksforgeeks.org/

2. https://www.wikipedia.org/

3. https://stackoverflow.com/

4. Kernighan, B. W. 1988. The C programming language (2nd ed.). Pearson.