

## List of Contents

Sr. No.	Contents	Page No.
1.	Introduction	<b>2.</b>
2.	Walk-through manual for user and admin	<b>3.</b>
3.	List of concepts	<b>26.</b>
4.	Elaboration of concepts	<b>27.</b>
5.	Future visions	<b>34.</b>
6.	References	<b>35.</b>

# INTRODUCTION

We have created a Bus reservation system , using salient features of C++ and concepts of Object Oriented Programming. It handles users as well as admin very efficiently and has been given different privileges to both of them . Users can book a bus ticket , cancel booking ,view their previous travelling history whereas admin can view booking status of all buses, can modify details of bus if required and can reserve some seats if he/she wishes to do so.

The overview of concepts used is as follows :

We have used **Virtual** functions as well as type field for differentiating the types of objects whether they are of user type or admin type during runtime.

We have **Inherited** user class and admin class from the Person class which is our **Base** class.

We have used **Friend** function to load the data from the database . We have made this function friend because every time if we want to upload or download data from the database ,we don't require objects every time.

With the help of **Encapsulation**, we are restricting the unwanted updates and access of the private information of our beloved users.

We have also implemented **Dijkstra** which is single source shortest path algorithm to find shortest distance between given source city and desired destination city. Hence the travelling expenditure of the user is minimized , since they shall take the shortest route to their desired destination .

Rest is explained further....

## **Walk-through manual for users and admin :**

Here we shall be majorly elaborating on the three main components from the home page of our program and they are shown in figure 1.



Figure 1 - Home page

## User Signup

We can see in figure 1, we have placed the user signup option at index 2 on the home page of our program and we shall be elaborating the same below. So, after pressing the key 2 and entering the program will redirect to user signup page .

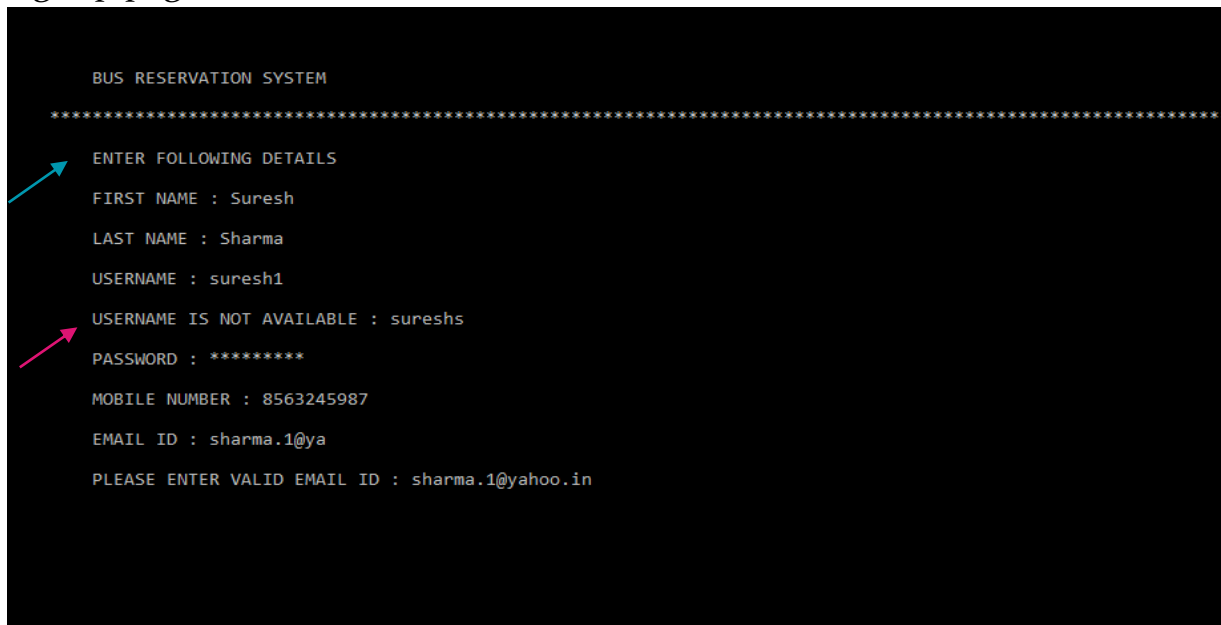


Figure 2-Signup- page

Here user must fill up all the entries just like first name, last name, username, password, mobile number, email id in the right format, in order to signup successfully.

The blue arrow in the console indicates that the username that user is trying to enter is already been allocated to some other user and is not available at this time and he must enter a different username.

The Pink arrow indicates that the email id that has been entered by the user is in wrong format and the program will repeatedly ask the user to enter his/her email id till correctly given so.

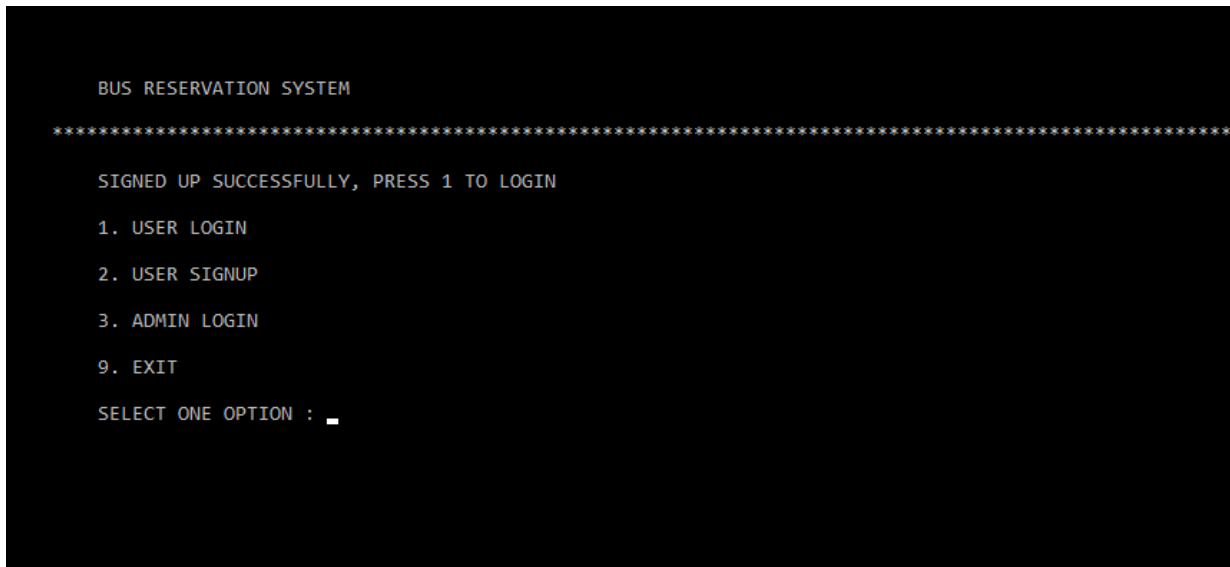


Figure 3 – Home page indicating successful signup

Now after successful signup by the user he/she can now login with his/her login credentials i.e. username and password and enjoy the services that have been provided by us.

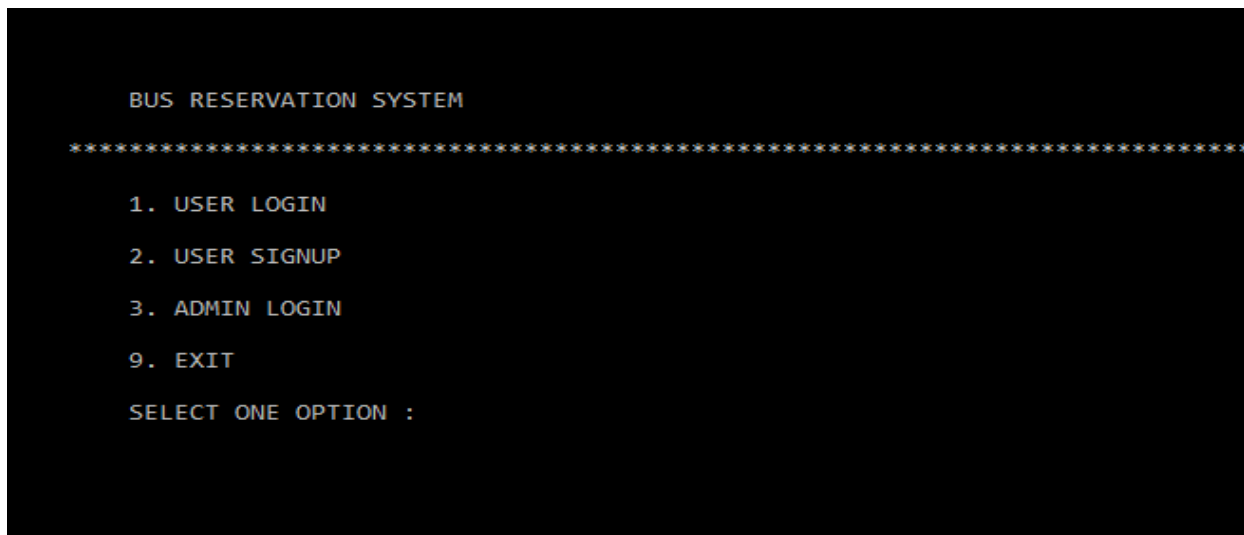


Figure 4 – Home Page

Now user can press 1 to login or 9 to exit.

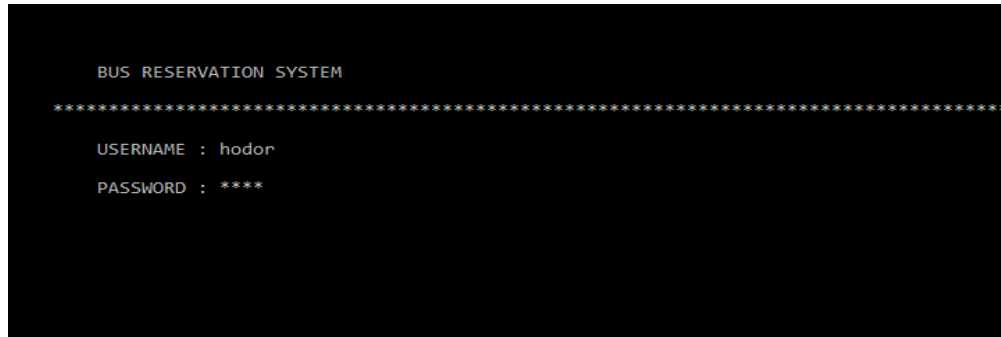


Figure 5 – User login page

Now User have to enter the right credentials that he/she has entered during signup

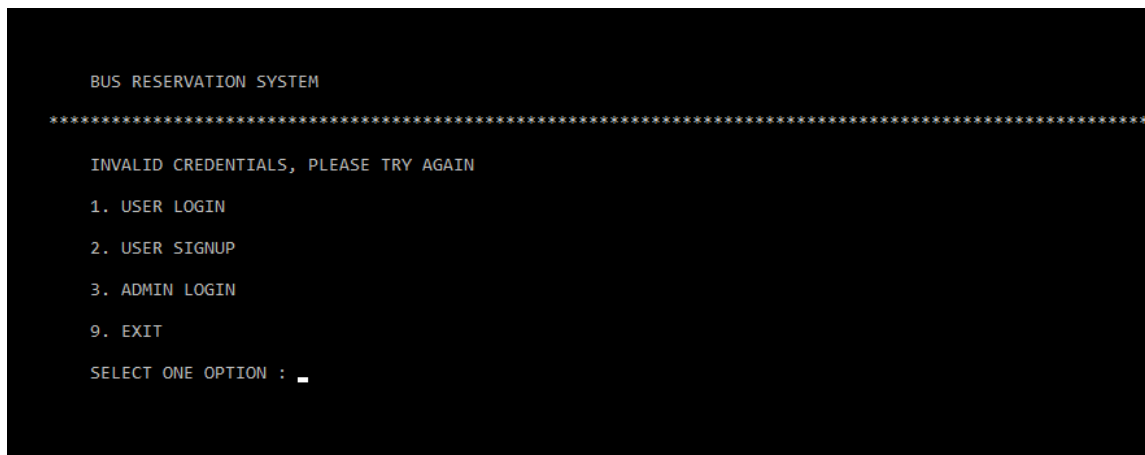


Figure 6-Error page ,user has entered wrong credentials

Here since user has entered wrong credentials he/she has been redirected to home page again.

## User login



Figure 7 – Login page(when user has entered correct credentials)

Now since user has entered correct credentials he/she will be redirected to his/her dashboard.



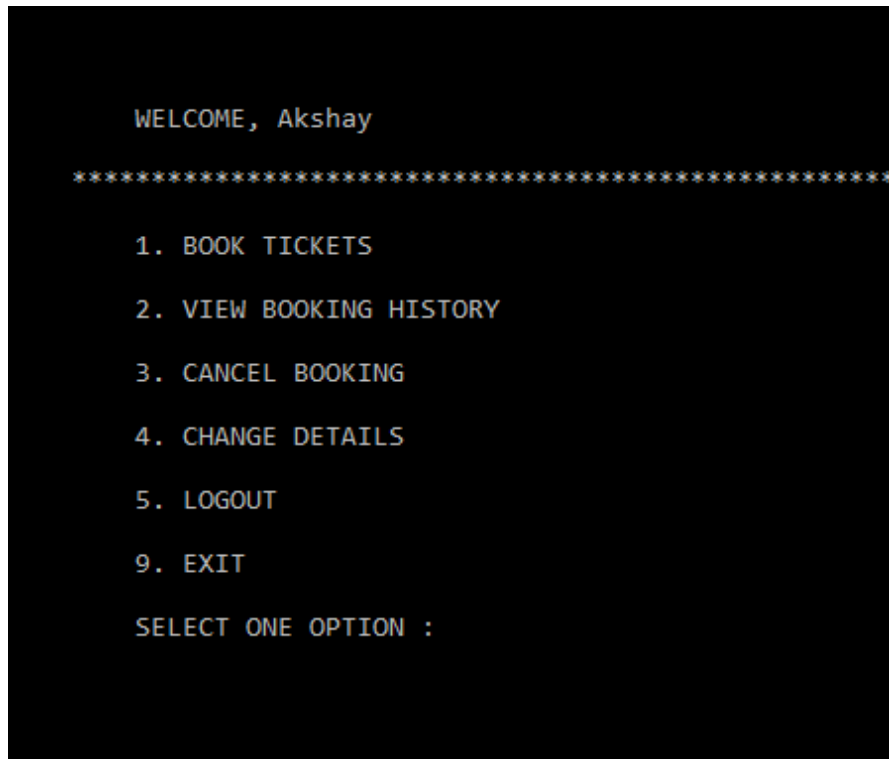


Figure 8: User Homepage

The above screenshot indicates successful sign in of user. As indicated, by entering corresponding indices user can access the above mentioned features. Let's take the example when user is pressing 1 and chooses to Book Ticket.

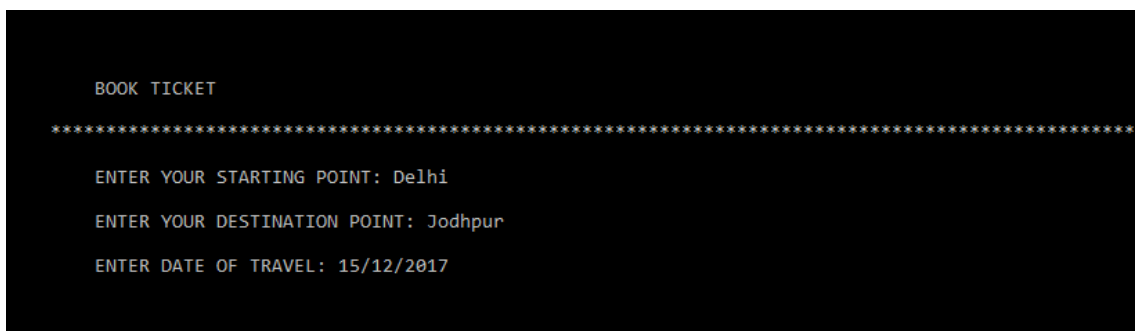


Figure 9 : Booking Ticket Page

The above screenshot shows Ticket Booking page. Here the user will enter Source, Destination and Valid Date to find a bus. Let's consider that user wants to book a ticket from Delhi to Jodhpur for the date 15/12/2017.

```

YOU SEARCHED FOR Delhi TO Jodhpur ON 15/12/2017
*****

TOTAL MINIMUM DISTANCE : 612 KM
TOTAL FARE : Rs. 918 /-
PATH : Delhi -> Jaipur -> Jodhpur
*****

Delhi -> Jaipur
ARRIVAL TIME : 11:12 PM          [N]  [N]  [N]  [A]  [A]  [A]  [A]  [A]  [A]  [A]
DEPARTURE TIME : 11:12 AM        [A]  [A]  [A]  [A]  [A]  [A]  [A]  [A]  [A]  [A]
TOTAL SEATS : 40                 [A]  [A]  [A]  [A]  [A]  [A]  [A]  [A]  [A]  [A]
AVAILABLE SEATS : 37            [A]  [A]  [A]  [A]  [A]  [A]  [A]  [A]  [A]  [A]
*****

Jaipur -> Jodhpur
ARRIVAL TIME : 2:12 PM          [N]  [A]  [A]  [A]  [A]  [A]  [A]  [A]  [A]  [A]
DEPARTURE TIME : 3:45 AM        [A]  [A]  [A]  [A]  [A]  [A]  [A]  [A]  [A]  [A]
TOTAL SEATS : 40                 [A]  [A]  [A]  [A]  [A]  [A]  [A]  [A]  [A]  [A]
AVAILABLE SEATS : 39            [A]  [A]  [A]  [A]  [A]  [A]  [A]  [A]  [A]  [A]
*****

1. CONFIRM BOOKING
2. CHANGE JOURNEY DETAILS
3. HOMEPAGE
SELECT ONE OPTION :

```

Figure 10: Seat Availability chart of all buses in the path (connecting or direct), Fare details

The above screenshot shows detail of the buses searched by the user for a particular date. It shows the shortest path if direct bus is not available. It also shows total fare. It shows details like arrival time, departure time and available seats and seat layout for every bus in the path. Here it has shown two connecting buses, first from Delhi to Jaipur and second from Jaipur to Jodhpur as no direct bus is available. Now the user has 3 options as shown above in Figure 5. First to confirm booking , second to change journey details and third to redirect to user homepage. Let's consider the example where user chooses to confirm his booking.

```
YOUR TICKET DETAILS:
*****

USER NAME : hodor
TICKET NO. : hDj15127
ORIGIN : Delhi
DESTINATION : Jodhpur
DEPARTURE TIME : 11:12 AM
ARRIVAL TIME : 2:12 PM
THANK YOU FOR BOOKING WITH US,PRESS ANY KEY TO CONTINUE.....
```

Figure 11 : Ticket details

The above screenshot shows the ticket details generated by program. It shows username, unique Ticket Number, Origin, Destination, Departure Time and Arrival Time. By Pressing any key it will redirect to home page.

```
WELCOME, Akshay
*****

1. BOOK TICKETS
2. VIEW BOOKING HISTORY
3. CANCEL BOOKING
4. CHANGE DETAILS
5. LOGOUT
9. EXIT

SELECT ONE OPTION : _
```

Figure 12 : User Homepage

After pressing any key the program redirects user to user homepage. User can further access any features as shown in the figure. Now, Let's consider

the case where user wants to cancel the last booked ticket so he can press either 2 or 3 both allow to do so.

```
BOOKING HISTORY
*****

1   DOJ : 15/12/2018
    Ahmedabad -> Mumbai
*****

2   DOJ : 12/12/2018
    Delhi -> Goa
*****

3   DOJ : 15/12/2017
    Delhi -> Jodhpur
*****

1. CANCEL ANY BOOKING
2. TO EXIT TO MAIN MENU

SELECT ONE OPTION : _
```

Figure 13: User Booking History

The above screenshot shows the booking history of the user. Here user can choose any one of the two options i.e to cancel any booking or to exit to main menu. Let's consider the case where user wants to cancel any ticket, so he presses 1.

```
BOOKING HISTORY
*****

1    DOJ : 15/12/2018
      Ahmedabad -> Mumbai
*****

2    DOJ : 12/12/2018
      Delhi -> Goa
*****

3    DOJ : 15/12/2017
      Delhi -> Jodhpur
*****

1. CANCEL ANY BOOKING
2. TO EXIT TO MAIN MENU
SELECT ONE OPTION : 1
ENTER TICKET INDEX TO CANCEL TICKET : 3
```

Figure 14: User Booking History

The above screenshot indicates that user has decided to cancel his Ticket from Delhi to Jodhpur indexed 3 and so he shall press 3.

```
BOOKING HISTORY
*****

1   DOJ : 15/12/2018
    Ahmedabad -> Mumbai
*****

2   DOJ : 12/12/2018
    Delhi -> Goa
*****

1. CANCEL ANY BOOKING
2. TO EXIT TO MAIN MENU

SELECT ONE OPTION : _
```

Figure 15: User Booking History

If we compare this screenshot with figure 14, It is seen clearly that booking number 3 is erased from the booking history. Now user presses 2 to access more features from the user homepage.

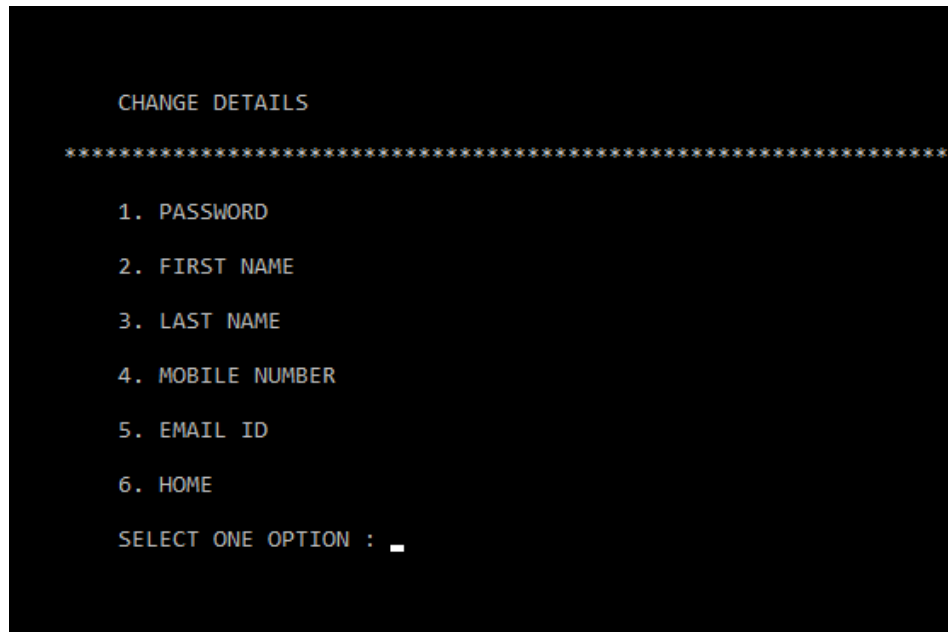
```
WELCOME, Akshay
*****

1. BOOK TICKETS
2. VIEW BOOKING HISTORY
3. CANCEL BOOKING
4. CHANGE DETAILS
5. LOGOUT
9. EXIT

SELECT ONE OPTION :
```

Figure 16: User Homepage

Now, consider that user wants to change some of his details, so he presses 4 to see what all details can be changed.



```
CHANGE DETAILS
*****
1. PASSWORD
2. FIRST NAME
3. LAST NAME
4. MOBILE NUMBER
5. EMAIL ID
6. HOME
SELECT ONE OPTION : _
```

Figure 17- Change details page

Now the user can choose any one of the options to change his/her details and on the next page he/she will be able to change his/her details easily.

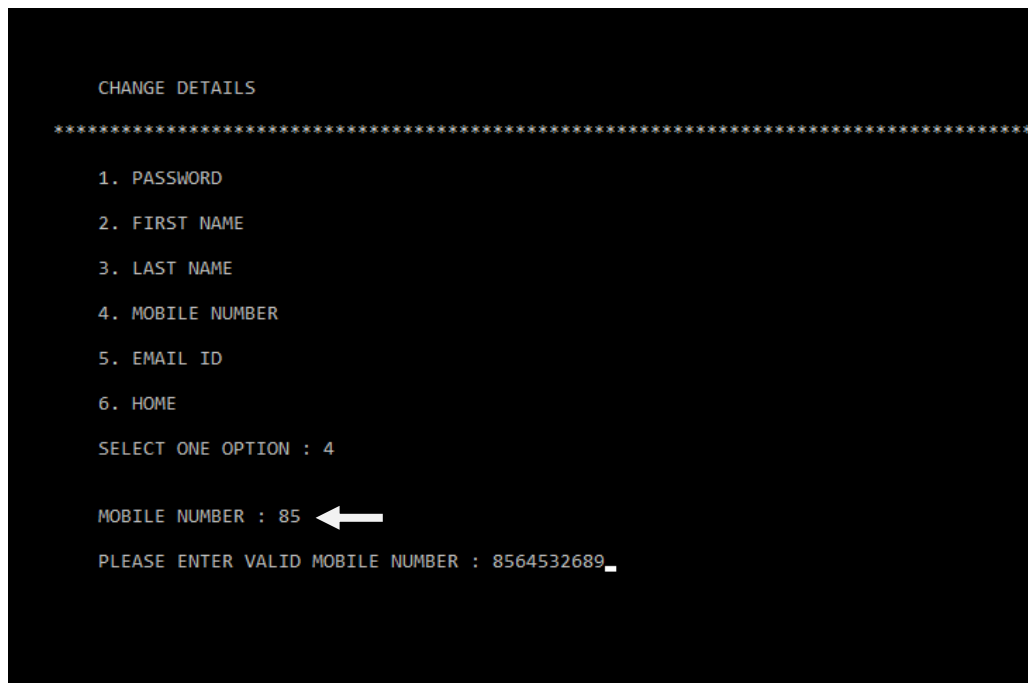


Figure 18- change details page

Now let's suppose user chooses option 4 that is he/she wants to change his/her mobile number. If user enters mobile number in wrong format than he/she will be asked to enter his/her mobile in the right format.

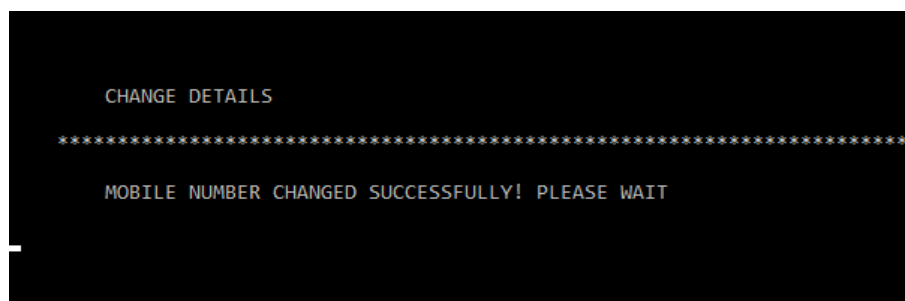


Figure 19 – Message showing that mobile numbers is changed

Now if the user enters the mobile number in the right format his/her mobile number will be changed and will be shown this message.



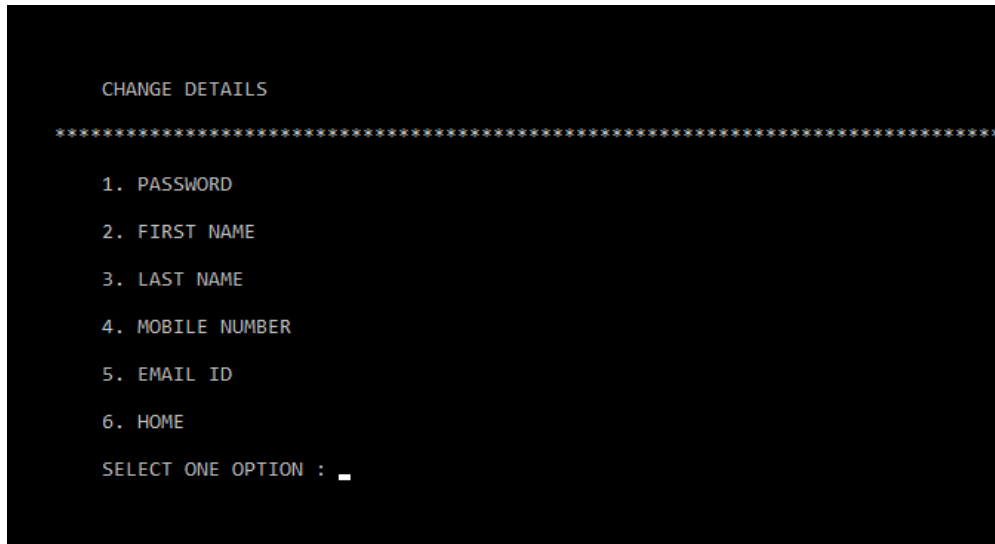


Figure 20 – Change details page

After showing the message of successful change of details the user will be redirected to his/her 'Change Details page' where he/she can change any other details if wishes to do.

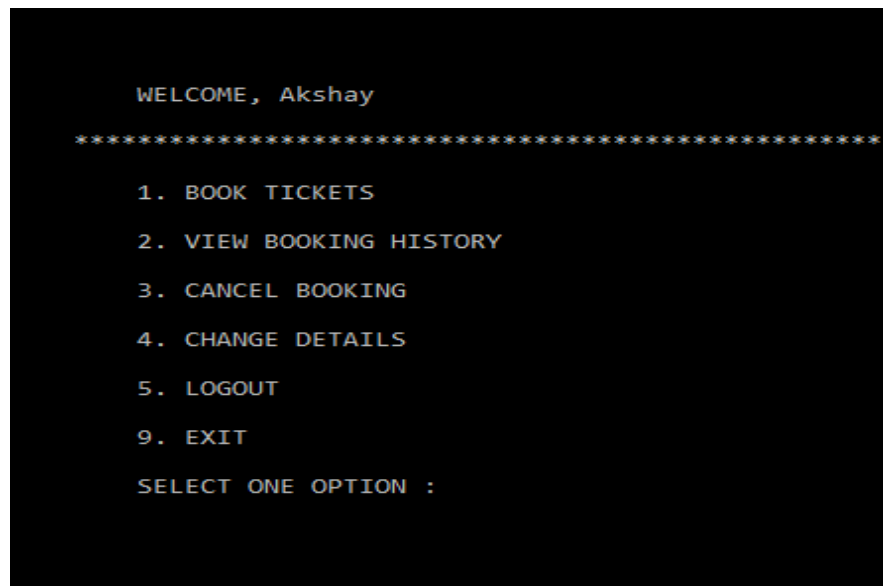


Figure 21 – Users Home page

Now let's say user enters the option 6 on change details then user will be redirected to his/her home page where user can choose any other option.

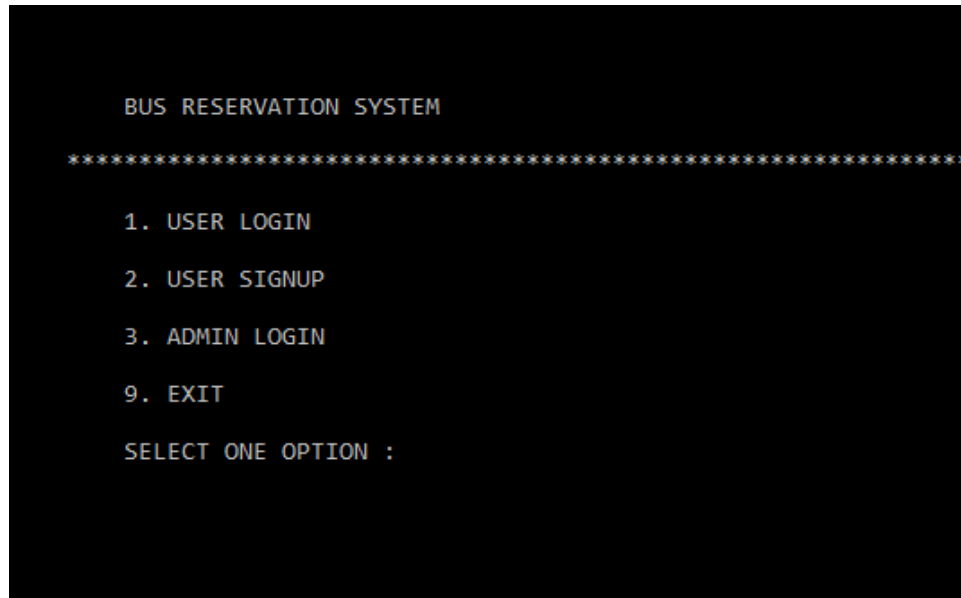


Figure 22: Home page

Now if user enter option 5 on his/her dashboard that is logout option than he/she would be successfully logged out from the account.

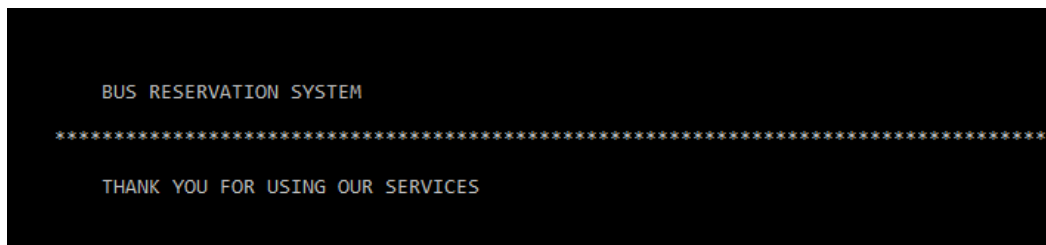


Figure 23 – Thanks giving

Now if the person enters option 9 on the home page than he/she will be successfully exited from our system.

## Admin login

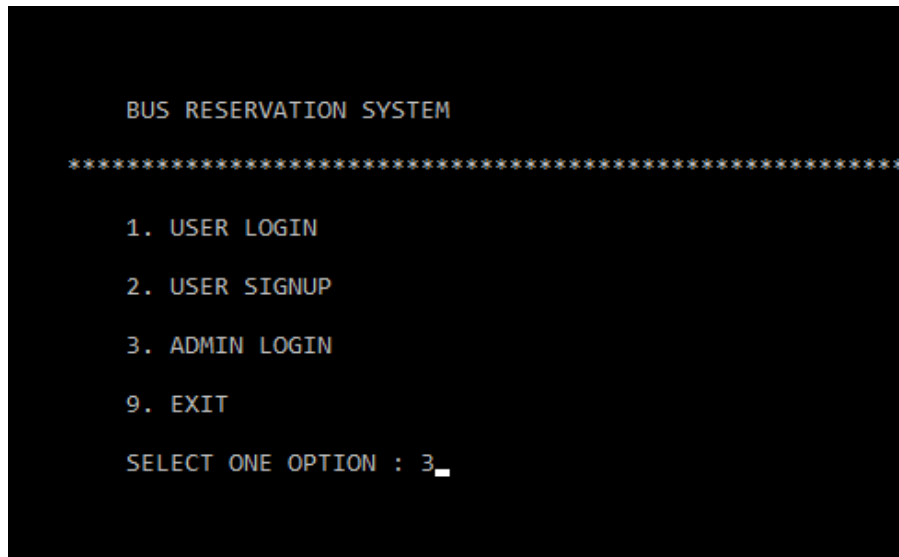


Figure 24 – Home page

Now admin can also login to enjoy his/her privileges .To login as an admin you have to choose option 3.

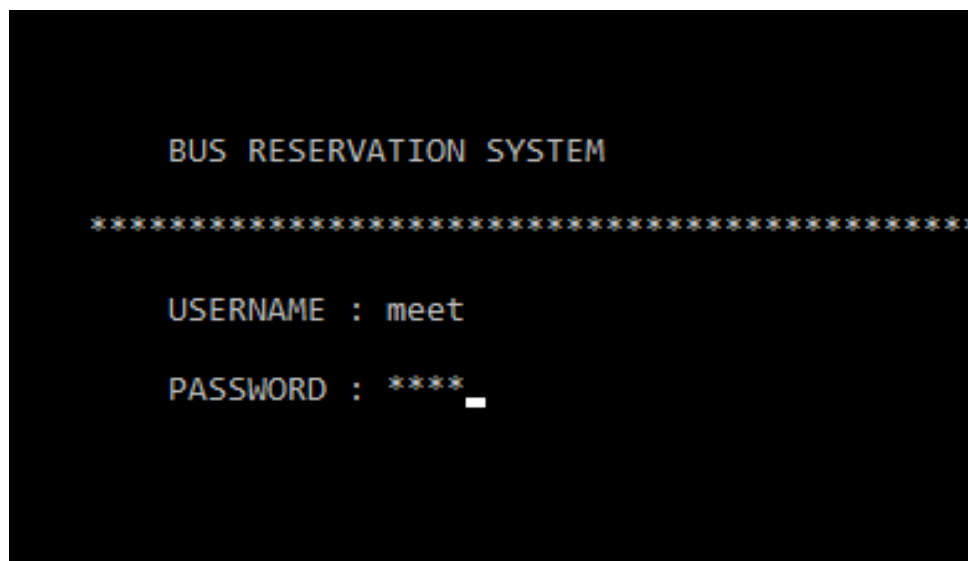


Figure 25-admin login page

Now , admin can enter his/her credentials to login

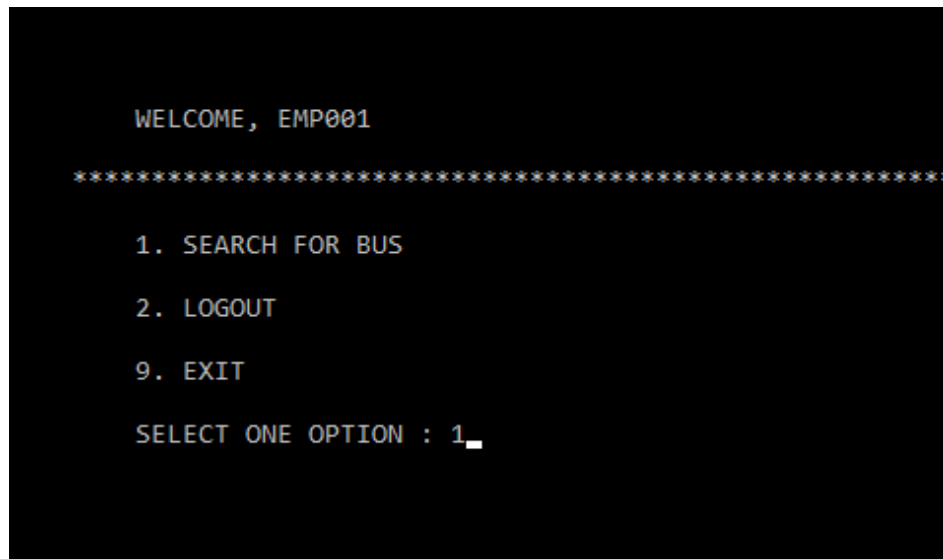


Figure 26 – Admin home page

After successful login of admin , he/she can choose any option let's say admin chooses option 1 of searching buses

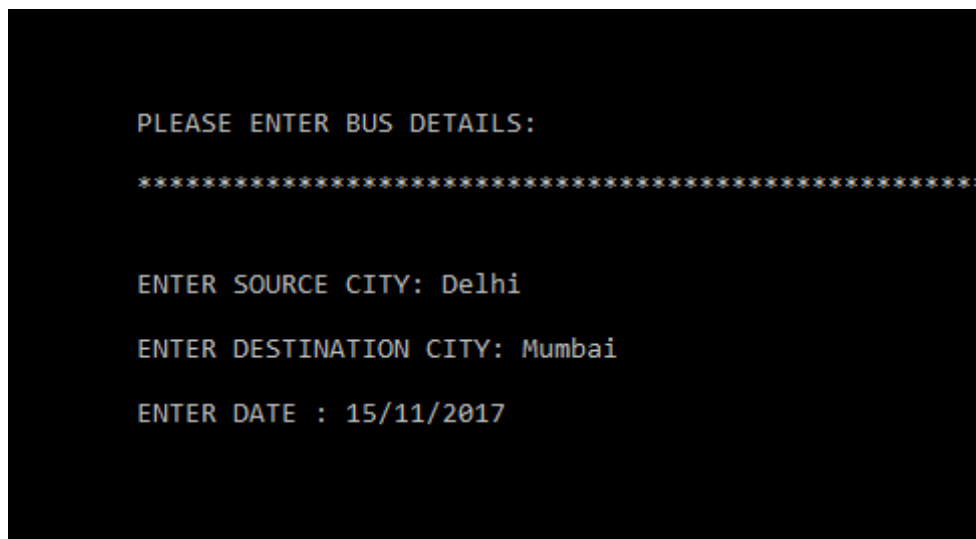


Figure 27 – Entering bus details

Now admin has to enter the source city , destination city and date in order to search for buses .

```

YOU SEARCHED FOR BUSES FROM Delhi TO Mumbai ON DATE 15/11/2017
*****

1: Delhi -> Jaipur
AVAILABLE SEATS :40

2: Jaipur -> Ahmedabad
AVAILABLE SEATS :40

3: Ahmedabad -> Mumbai
AVAILABLE SEATS :40

PRESS ANY BUS NO. TO CHANGE DETAILS:
PRESS 9 TO GO BACK TO HOME PAGE:
CHOOSE ANY OF ONE OPTION : 2

```

Figure 28 – Result of query by admin

After getting the query from the admin we have shown the list of the buses which will be used in the route having shortest distance . Now , admin can choose any of the buses and change the details of the same or go back to home page.

```

STATUS OF BUS FROM Jaipur TO Ahmedabad ON 15/11/2017 :

Jaipur -> Ahmedabad

*****

ARRIVAL TIME : 11:12          [N]  [_]  [_]  [_]  [_]  [_]  [_]  [_]  [_]  [_]
DEPARTURE TIME : 12:24 AM     [_]  [N]  [N]  [_]  [_]  [_]  [_]  [_]  [_]  [N]
TOTAL SEATS : 40              [N]  [_]  [_]  [_]  [_]  [_]  [_]  [_]  [_]  [_]
AVAILABLE SEATS : 35          [_]  [_]  [_]  [_]  [_]  [_]  [_]  [_]  [_]  [_]

*****

PRESS 1 TO CHANGE TIME OF THIS BUS:
PRESS 2 TO RESERVE SOME SEATS :
PRESS 3 TO GO TO HOME PAGE :
CHOOSE ANY ONE OF THE OPTIONS : 2_

```

Figure 29 - Details of a particular bus

In the above figure , admin can choose any of the option . Let's say admin chooses to 'Reserve Some Seats' option .

```
STATUS OF BUS FROM Jaipur TO Ahmedabad ON 15/11/2017 :

Jaipur -> Ahmedabad

*****

ENTER NUMBER OF SEATS TO BE RESERVE:2

ENTER FIRST SEAT NO.: 34

ENTER NEXT SEAT NO.:35

CONGRATS SEATS HAS BEEN RESERVED SUCCESSFULLY _
```

Figure 30 –Admin to reserve some seats

Admin here can enter the number of seats and seat number that he/she wants to reserve and those particular seats of that bus will be reserved for the admin.

```
STATUS OF BUS FROM Jaipur TO Ahmedabad ON 15/11/2017 :

Jaipur -> Ahmedabad

*****

ARRIVAL TIME : 11:12          [N]  [_]  [_]  [_]  [_]  [_]  [_]  [_]  [_]  [_]
DEPARTURE TIME : 11:23 PM     [_]  [N]  [N]  [_]  [_]  [_]  [_]  [_]  [_]  [N]
TOTAL SEATS : 40              [N]  [_]  [_]  [_]  [_]  [_]  [_]  [_]  [_]  [_]
AVAILABLE SEATS : 33          [_]  [_]  [_]  [N]  [N]  [_]  [_]  [_]  [_]  [_]

*****

PRESS 1 TO CHANGE TIME OF THIS BUS:
PRESS 2 TO RESERVE SOME SEATS :
PRESS 3 TO GO TO HOME PAGE :
CHOOSE ANY ONE OF THE OPTIONS : 1
```

Figure 31 - Admin chooses option 1

Suppose , admin now chooses to change the time of buses .

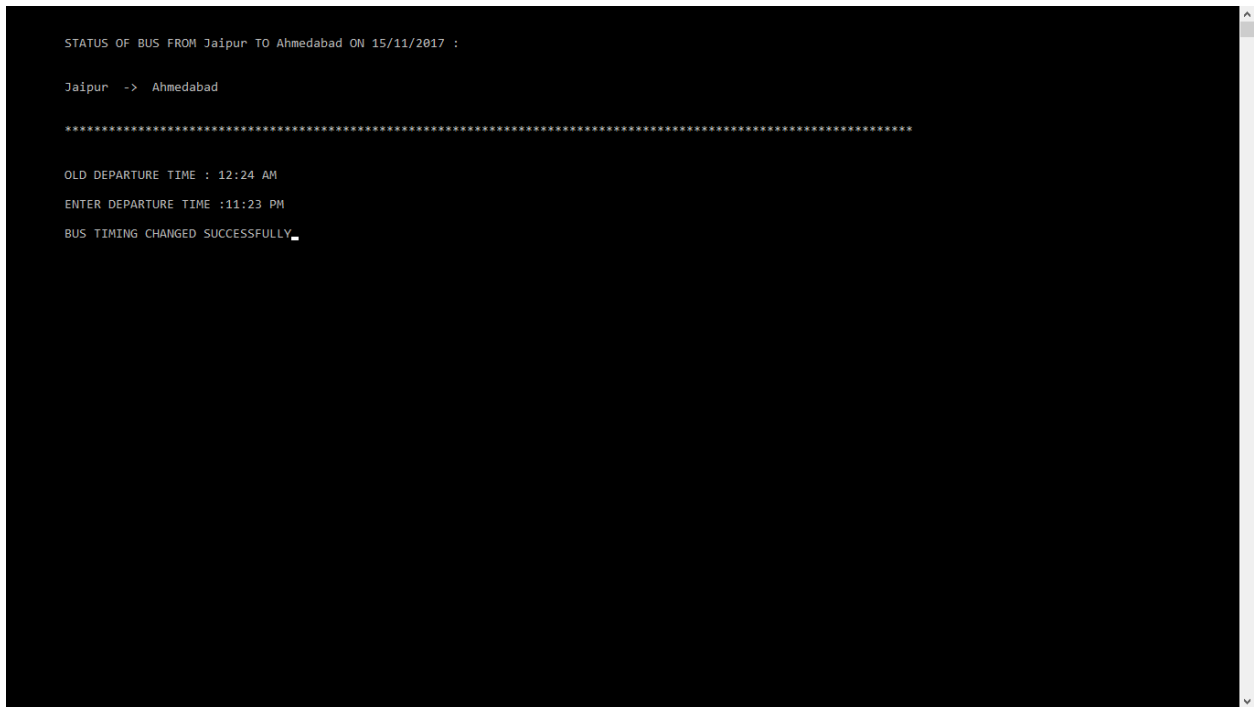


Figure 32 - Admin changing time

After choosing the time changing option of buses , admin will be redirected to figure 32 and admin can change the departure time of the bus. Now the departure time of the bus will be changed successfully as entered by the admin. After that admin can logout from his/her dashboard and if he/she wants to logout that have to enter option 2 and if wants to exit from our system enter option 9 .

# **How we created this project**

- I. List of Concepts used.
- II. Elaboration of how ,where and why are we using the concepts in correlation with each other.



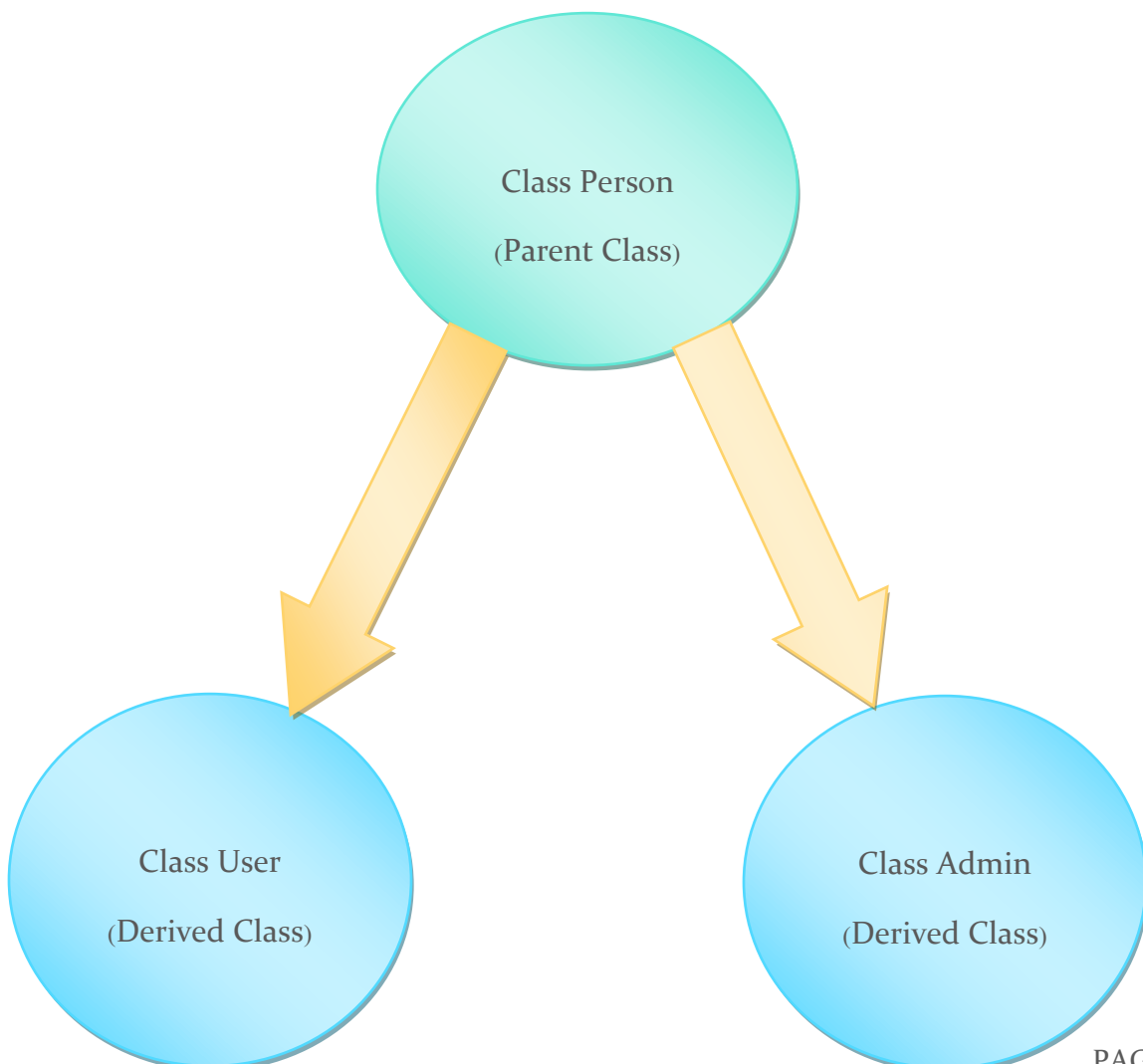
# I.List of concepts

1. Inheritance
2. Run-time Polymorphism (Virtual functions)
3. Type field
4. Friend functions
5. Operator overloading
6. Class, Enumeration
7. Constructor
8. Copy Constructor
9. Static variable and Static Functions
10. Various containers of STL like Map , Vector
11. Exception handling using Try - Catch
12. String class
13. Namespace
14. Constant functions
15. Dijkstra's Algorithm
16. CSV database

## II. Elaboration of how ,where and why are we using the concepts in correlation with each other.

### 1. Inheritance

**Inheritance** is when a class is based on another class, using the same implementation. Inheritance is a mechanism in which one class acquires all the properties and behaviors of the parent class. Inheritance is a technique of code reuse. Inheritance map in our program is as shown below.



As shown in the figure here user class and admin class are derived class and person class is the base class. Below three figures shows the variables in all the classes. Here it can be seen that both user and admin have username, password, and type while only user class has first name, last name, email and mobile number and only admin class has employee ID.

```
1121 class person{
1122 protected:
1123     string username,password;
1124 public:
1125     enum t{U,A};
1126     t type;
```

```
1136 // user class inherited from person cla
1137 class user:public person{
1138     string fname,lname,email,mobile;
1139 public:
```

```
1154
1155 // admin class inherited from
1156 class admin:public person{
1157     string employeeID;
1158 public:
```

## 2. Virtual Functions

**Virtual Function** is a member function which is declared within base class and is re-defined (Overridden) by derived class. When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class's version of the function. Thus , virtual function ensures that the correct derived class function is called when a virtual function is called using pointer of base class, which might point to an object of derived class.

```
1121 class person{
1122 protected:
1123     string username,password;
1124 public:
1125     enum t{U,A}; // Type u
1126     t type;
1127     person(string u="",string p="",t ua=U);
1128     //virtual void change_details(); // Virt
1129     virtual bool login(int &index){} //
1130     virtual void homepage(int index){}
1131     virtual void logout(){} // This
1132     //bool user_auth(); // thi
```

```

1136 // user class inherited from person class
1137 class user:public person{
1138     string fname,lname,email,mobile;    //
1139 public:
1140     user(string u="",string p="",t ua=U);
1141     //void recover_password();    // Admin
1142     void change_details(int &index);
1143     void logout();
1144     bool login(int &index);
1145     void homepage(int index);
1146     void friend print();

```

```

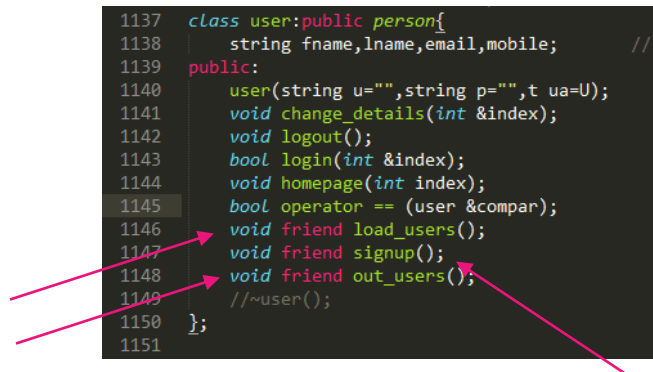
1154 // admin class inherited from person
1155 class admin:public person{
1156     string employeeID;    // th
1157 public:
1158     admin(string u="",string p="",t u
1159     bool login(int &index);
1160     void homepage(int index);
1161     void logout();
1162     void friend load_admins();
1163     void friend print();

```

Here as shown three functions which are required for both user and admin class are made virtual in person class. Here, the functions login ( ) , logout ( ) and homepage ( ) are virtual.

### 3. Friend Functions

Friend Function is the one which is given access to private variables of the class. It's scope is global rather than that of the particular class. It need not be invoked using an object of the class. Use of friend function in our project is as explained below.



```

1137 class user:public person{
1138     string fname,lname,email,mobile;    // t
1139 public:
1140     user(string u="",string p="",t ua=U);
1141     void change_details(int &index);
1142     void logout();
1143     bool login(int &index);
1144     void homepage(int index);
1145     bool operator == (user &compar);
1146     void friend load_users();
1147     void friend signup();
1148     void friend out_users();
1149     //~user();
1150 };
1151

```

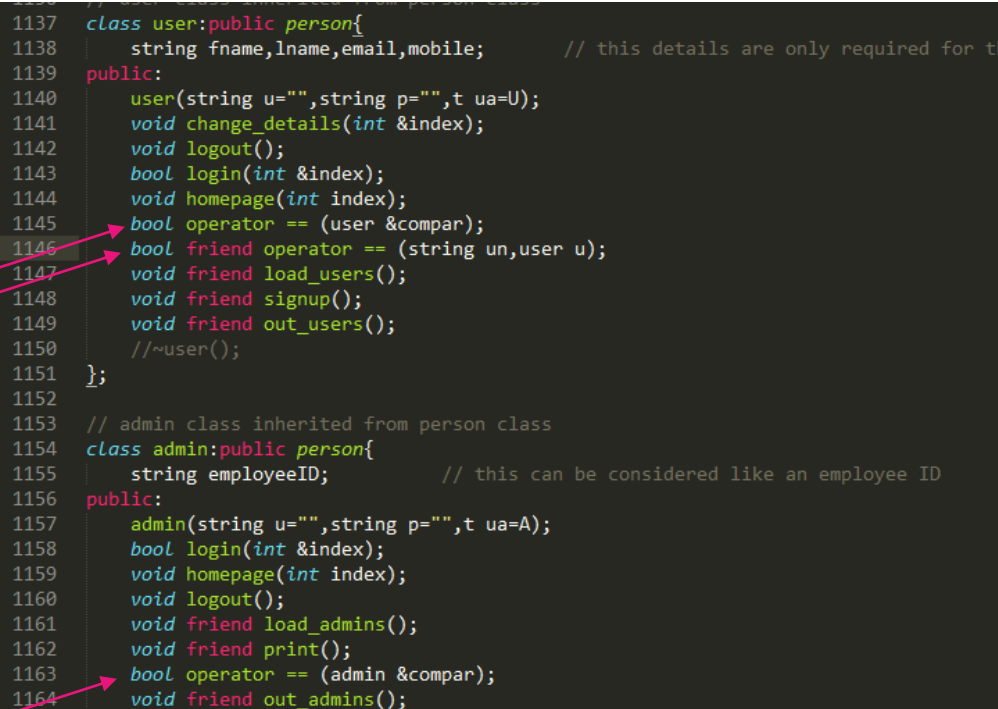
Three pink arrows point to the friend functions: `load_users()`, `signup()`, and `out_users()`.

Here friend functions used are for loading data from csv database, signup of a user, send data back to csv file at the end the program. First consider load\_users() function. It loads data from csv file to a vector thus it pushes back object of user class till the end of the data. If it doesn't have access to private variables than every input has to be saved in a set of extra variables, then a constructor will be called which will make an object of user class which will be added to the vector, this becomes tedious also it will require extra space. If this is declared as member function without friend keyword then it has to be invoked using an object of user class and there is no sense to do this as there is no particular object which calls this function rather this function is called in the beginning to load complete data, so here using friend becomes necessary. This

implies to out\_users() functions to which send data back to csv file. In this case if this function is not given access to private members many separate member functions are to be defined to send data to every private variable. Now consider user signup() function, in this case also if this function is not given access to private variables than extra variables are to be used which causes extra memory.

## 4. Operator Overloading

In programming , operator overloading is a specific case of polymorphism , where different operators have different implementations depending on their arguments. In our program operator overloading is used to return true if username and password match else it returns false.



```
1137 class user:public person{
1138     string fname,lname,email,mobile;        // this details are only required for t
1139 public:
1140     user(string u="",string p="",t ua=U);
1141     void change_details(int &index);
1142     void logout();
1143     bool login(int &index);
1144     void homepage(int index);
1145     bool operator == (user &compar);
1146     bool friend operator == (string un,user u);
1147     void friend load_users();
1148     void friend signup();
1149     void friend out_users();
1150     //~user();
1151 };
1152
1153 // admin class inherited from person class
1154 class admin:public person{
1155     string employeeID;        // this can be considered like an employee ID
1156 public:
1157     admin(string u="",string p="",t ua=A);
1158     bool login(int &index);
1159     void homepage(int index);
1160     void logout();
1161     void friend load_admins();
1162     void friend print();
1163     bool operator == (admin &compar);
1164     void friend out_admins();
```

## 5. Exception Handling

An exception is a problem that arises during the execution of a program. Exceptions provide a way to transfer control from one part of a program to another. A program without exception handling is not considered as user-friendly. In our program exception handling is used at a number of the places , as listed below :

- Check for a valid username, it throws exception when a user enters invalid username, a valid username consists of only alpha-numeric characters.
- Check for valid password.
- Check for valid name, it throws exception if name contains spaces.
- Check for valid mobile number, a mobile number is valid if it contains 10 digits and must contain all digits
- Check for valid email ID, a valid email ID is in the form of user@domain.com, it throws error if entered Email ID is not in the valid format
- While Logging in check if username and password entered are correct
- While Signing up check if username is available or not as username must be unique
- Also . while searching for a bus check whether the entered date is valid or not

This is only to list a few, exception handling is literally very powerful tool to handle exceptions as mentioned above. It makes things easy to code.

## 6. Dijkstra's Algorithm

Dijkstra's algorithm is a single source shortest path algorithm to give the shortest weighted distance from given source to all other nodes.

We have implemented Dijkstra's algorithm in our program to give the shortest path to the user from his current location to the desired location of travel to minimize his/her travel expenditure .We also show the person the connecting path in between cities , if we don't have a direct bus from the given source and destination entered by the user then we will show the user the minimum distance path through which he/she can travel and the number of buses he/she has to change in between his/her whole journey.

```

//*****DIJKSTRA FUNCTION TO FIND MINIMUM *****
int dijkstra(int src,int dest,vector<vector<int> > &dist){
    vector<bool>sptset(13,0);
    // sptset[src]=1;
    vector<int>dfs(13,INT_MAX);
    dfs[src]=0;
    for(int i=0;i<12;i++){
        int u=mindist(dfs,sptset);
        sptset[u]=1;
        for(int j=0;j<13;j++){
            if(dfs[j]>dfs[u]+dist[u][j] && dist[u][j]!=0 && dfs[u]!=INT_MAX && sptset[j]==0){
                dfs[j]=dfs[u]+dist[u][j];
                parent[j]=u;
            }
        }
    }
    return dfs[dest];
}

```

Code snippet of implementation of Dijkstra's Algorithm

In the above code snippet, we are taking as parameters the source and destination which will be given by the user and an adjacency matrix in the form of a 2-D vector. Now to store the shortest distances to all other nodes we have declared a vector **dfs** (denoting distance from source). Further after running the algorithm we are returning from the function the computed shortest distance from source to destination.

## 7. CSV Data

The full form of csv is "comma separated values". It is a good way to store data as csv. When program is turned off all the data stored in a vector is deleted which cannot be gained back, thus it doesn't make sense if a user has to signup every time he/she wants to book a ticket, also there won't be any data of his/her past bookings and it would be required to enter bus details every time the program is turned on. Thus, it is simply impossible to run this type of program without storing data before the program is turned off. This is done very efficiently by storing data in csv file before exiting. Our program is so designed that it loads data from csv file whenever it is opened and stores it in a vector and while exiting it stores this data of vector back to csv file and overwrites previous data so if a user makes changes in any of his/her details it is not lost, also all the booking history is stored in different files as required. Below is a screenshot of the code that is called loading and unloading of data of users from the file "users.csv".

```

1254 void persons::load_users(){
1255     ifstream u;
1256     u.open("users.csv");
1257     user *temp;
1258     temp = new user;
1259     int i=0;
1260     while(u.good()){
1261         getline(u,temp->fname,',');
1262         getline(u,temp->lname,',');
1263         getline(u,temp->username,',');
1264         getline(u,temp->password,',');
1265         //getline(u,temp->squestion,',');
1266         //getline(u,temp->sanswer,',');
1267         getline(u,temp->mobile,',');
1268         getline(u,temp->email,'\n');
1269         temp->type = person::U;
1270         if(i)
1271             users.push_back(*temp);
1272         else
1273             i++;
1274     }
1275     u.close();
1276     delete temp;
1277     users.erase(users.begin()+users.size()-1);
1278 }
1279
1280

```

Function to load data

```

1307
1308 void persons::out_users(){
1309     ofstream u;
1310     u.open("users.csv",ios::trunc);
1311     u<<"First Name,Last Name,Username>Password,Mobile Number,Email ID\n";
1312     for(int i=0;i<users.size();i++){
1313         u<<users[i].fname<<","<<users[i].lname<<","<<users[i].username<<","<<users[i].password<<","<<users[i].mobile<<","<<users[i].email<<"\n";
1314     }
1315 }
1316

```

Function to unload data



# Challenges we faced in this Project

1. Firstly we had challenge to save data. Every time the program is reloaded the previous data is deleted thus user need to signup each time the program is reloaded and buses data need to updated every time which is not possible. To solve this we learned to save data as a csv file and import and export data at the time of starting and ending the program respectively.
2. Next challenge was as the line of code increased it became very difficult to manage everything. To overcome this we started making small functions and called them whenever required.
3. First we didn't know how to show connecting buses, to overcome this we made a graph with cities as vertices and if there is a direct bus between them than there would be a edge between them with weight equal to the distance between them. We learned Dijkstra's algorithm to find the minimum distance and path between two cities.
4. We have never made such a large code and coded only small once by directly coding without writing the structure but it became difficult as the lines keep on increasing so later we first wrote full structure of the program and finally coded it, which made it very simple.
5. After doing this project we also got a experience how to work in a team.

# Future Visions

We have developed this project because of lack of good online bus reservation services.

The services provided nowadays are not upto the mark and we will try hard to improvise the smooth functioning of the whole process of booking the ticket till the user reaches it's destination.

The database used in this project is currently working on a single machine but if we upload the database on a server then we can increase the scale of number of users and also the flexibility of the system.

We can also add real time tracking of buses so that users can check status of their bus in which they are travelling to give a tentative location of the bus and the approximate time of arrival and departure.

This is a small scale project but in future we can expand this also to train reservation system , flight reservation system .

# References

1. The C++ programming language, Bjarne Stroustrup
2. The Complete Reference C++
3. E Balagurusamy C++
4. Online resources like Geeksforgeeks , Tutorials Point , Cpp reference

# Thank You!!

