SSN College of Engineering, Kalavakkam

Department of Computer Science and Engineering

III Semester - CSE 'A ',’B’ & ‘C’

UCS 1312 Data Structures Lab Laboratory

**Academic Year**: 2019-2020

**Batch**: 2018-2022

**Class**: CSE B

**MINI PROJECT**

**RAILWAY RESERVATION SYSTEM**

**PROJECT MEMBERS:**

185001119 Raghav R

185001124 Ramakrishnan A M

185001126 Rishi Vardhan R

**DATE**: 15/10/2019

**ABSTRACT:**

The Railway Reservation System project mimics a short scale version of the IRCTC booking website for Indian Railways. The project focuses on maintaining a simpler user-interface with features like Book a ticket, View ticket status, View Reservation chart, View Trains List, View Seat Availability, cancel a ticket and options to view short and efficient routes between places. The project is based on incremental modular programming approach with measures to instill better upgrades in the future.

**METHODOLOGY**

**MAIN**

**railapp.c:** The main program file that implements the application like interface and has all the function calls.

**HEADER FILES**

**railway.h:** Contains all the functions required for the ticket booking process. This is the core file of the project as it implements the main task at hand.

**dijkstra.h:** Implements Dijkstra Algorithm to find the shortest path between two stations and returns the shortest distances

**passengerlist.h:** Maintains passenger details in a doubly linked list.

**stations.h :** Stores details regarding distances and routes between stations.

**graph.h:** Implements concepts for Breadth First Search and Depth First Search.

**traindate.h:** Contains details regarding every train say availability for each date.

**stack.h:** Contains Stack Functions

**queue.h:** Contains Queue Functions

**STRUCTURE DEFINITIONS:**

**node: Structure to store the station details.**

**Location: stations.h**

struct node

{

char dest[100];

int vertex;

int dist;

struct node\* link;

};

**details : Structure to store the details of Passenger.**

**Location: passengerlist.h**

typedef struct

{

char name[100];

char gender;

int age;

char proof[50];

char Class[50];

char dest[50];

char src[50];

float cost;

}details;

struct Node

{

details det;

Position Next;

Position Prev;

}

**snode : Structure to store stack details.**

**Location: stack.h**

typedef struct s

{

int x;

struct s \*next;

}snode;

**qnode : Structure to store queue details.**

**Location: queue.h**

typedef struct q

{

int n;

struct q \*next;

}qnode;

**train : Structure to store the admin login details.**

**Location: TrainDate.h**

typedef struct

{

char tname[100];

char tno[10];

int tick1,tick2,tick3;

char src[100];

char dest[100];

char via[100];

List L;

}train;

**date : Structure to store the train details.**

**Location: TrainDate.h**

typedef struct

{

int dd;

int mm;

int yy;

train t[20];

}date;

**MODULES:**

* **railway.h**
  + **ticketcost(float dist, char Class[])** – returns cost of rail ticket considering both class of seat booked
  + **Book(int G[][5])** – Handles the booking of tickets
  + **Status()** – Handles information regarding the status of tickets
  + **Cancel() –** Handles operations regarding cancellation of tickets
  + **Routes(int G[][5]) –** Displays routes available for station
  + **Chart() –** Displays Reservation Chart
  + **Availability() –** Displays Availability of seats in Train
* **dijkstra.h**
  + **dijkstra(int G[MAX][MAX], int n, int startnode) –** Implements Dijkstra’s Algorithm
  + **dijkstradist(int G[MAX][MAX], int n, int startnode, int dest) –** To find short distances between routes
* **graph.h**
  + **initial(int adj[][5], int n) –** Initial values of Adjacency list
  + **insert(int adj[][5], int src, int dest) –** Insert values into Adjacency list
  + **display(int adj[][5], int n) –** Display Adjacency list
  + **visited(int ele, int ar[], int n) –** Marking Visited Lists
  + **dfs(int start, int adj[][5], int ar[], int dest) –** Implements DFS Algorithm
  + **bfs(int start, int adj[][5], int ar[], int dest) –** Implements BFS Algorithm
* **passengerlist.h**
  + **CreateEmptyList() –** To Create a Empty List
  + **isEmpty(List L) –** To return bool value to denote empty list
  + **Find(List L, char X[]) –** To find given record in the passengers list
  + **addend(List L, details d) –** To add to end of List
  + **delete(List L, char X[]) –** To delete a specific node from the List
  + **disp(List h) –** To display contents of List
* **queue.h**
  + **qEmpty() –** returns a bool value to denote empty queue
  + **queuedisplay() –** To display Queue
  + **enqueue(int num) –** To add element to Queue
  + **dequeue() –** To delete element from queue
* **stack.h**
  + **push(int value) –** To push element into stack
  + **Pop() –** To pop element from stack
  + **Peek() –** To peek at element in stack
  + **isEmpty() –** Returns a bool value to denote a empty stack
  + **stackdisplay() –** Display Stack
* **stations.h**
  + **init(int n) –** To initialize values to graph
  + **addEdge(int src, int dest, char ar[], int dist) –** To add an edge to the graph
  + **printGraph(int n) –** To print the graph
  + **ListtoMatrix(int G[][5]) –** To convert Adjacency List to Adjacency Matrix
  + **stationInitialise(int G[][5]) –** To initialize graph with all stations
* **TrainDate.h**
  + **traininitialise() –** Initialize all trains with their respective details
  + **displaytrains() –** To Display Train Details

**DATA STRUCTURES USED TO BUILD THE PROJECT**

**Doubly Linked List:** To store Passenger details.

**Graph:** To find the different routes form source to destination between stations.

**Linked List:** To store Adjacency List

**Stack:** To implement BFS and DFS

**Queue:** To implement BFS and DFS

**ADT:** To store related functions and structures into one file and implement this structure as and when required along with the interface.

**CONCEPTS USED TO BUILD THE PROJECT**

**Arrays:** Used for storing collection of the same data like list of all the trains.

**Structures:** Used for storing collection of related data like user\_details, train\_details.

**Functions:** For implementing specific operations.

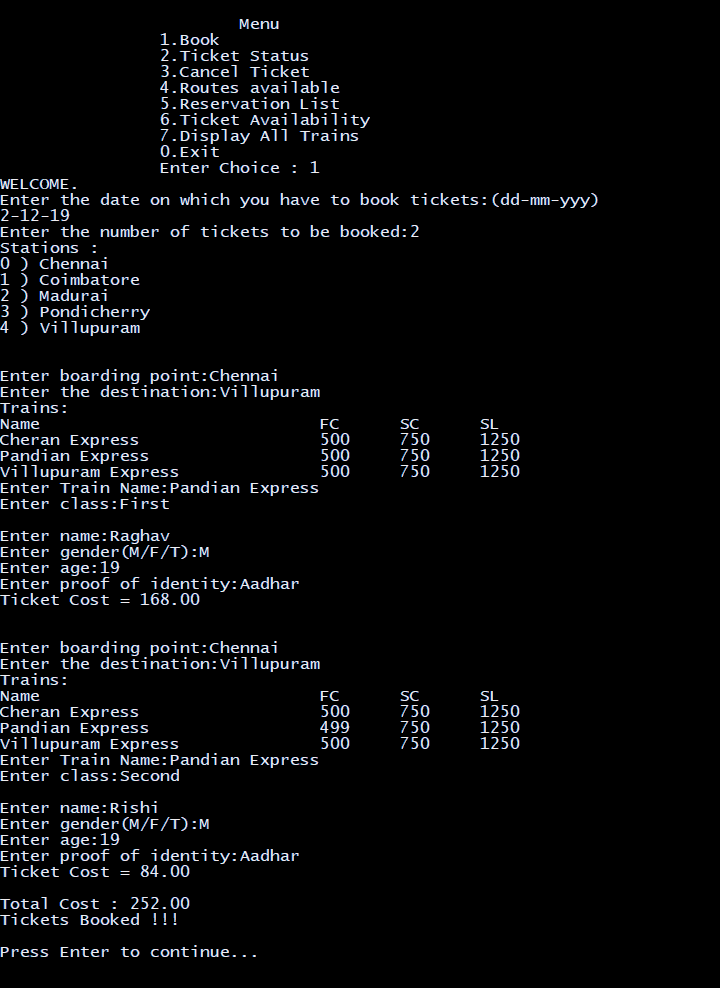
**Modularity:** The entire program is split into several functions and parts, each of which serves a specific purpose and can be repeatedly used.

**Incremental Programming:** Several versions were created prior to the final and were constantly modified to make it the best possible version there is.

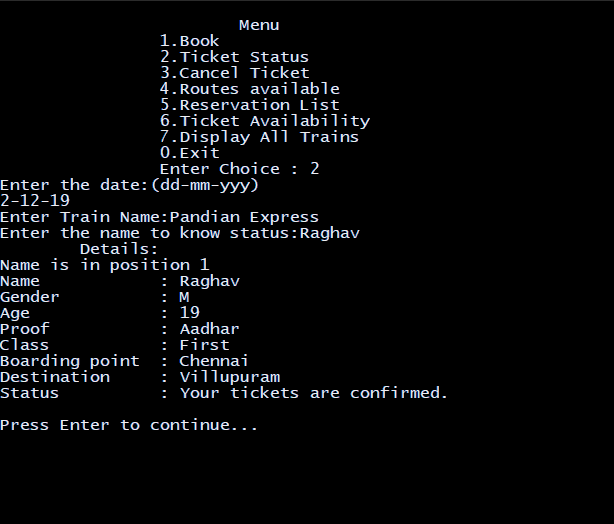
**Integration:** Several parts of the code was designed by different people and was integrated into one program using the interface code as the base.

**Output Screenshots**

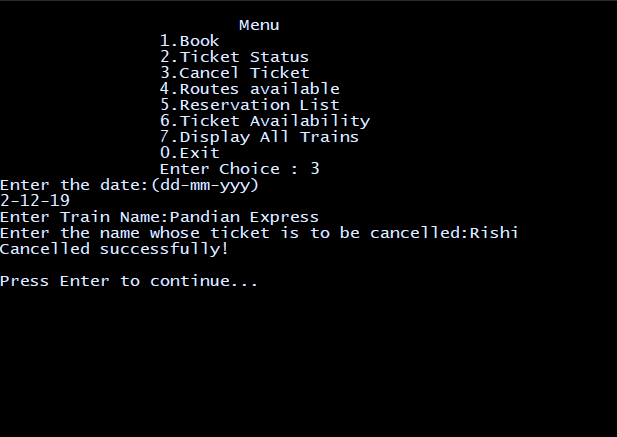
**Booking**



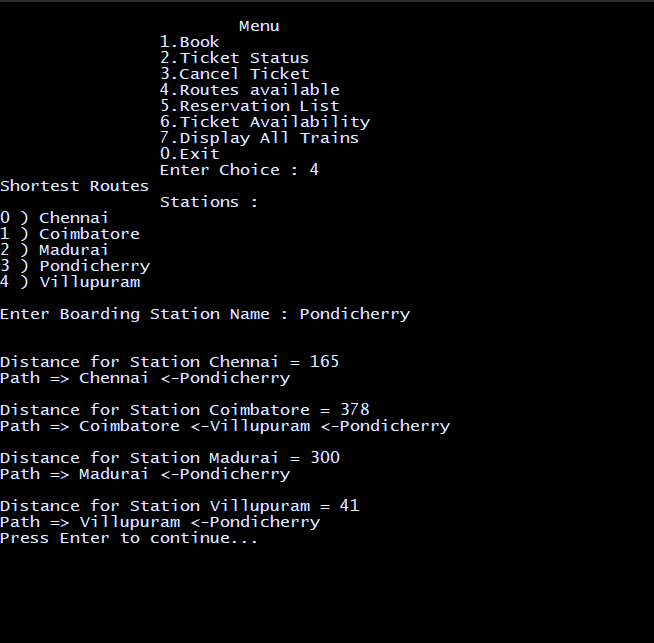
**Ticket Status**



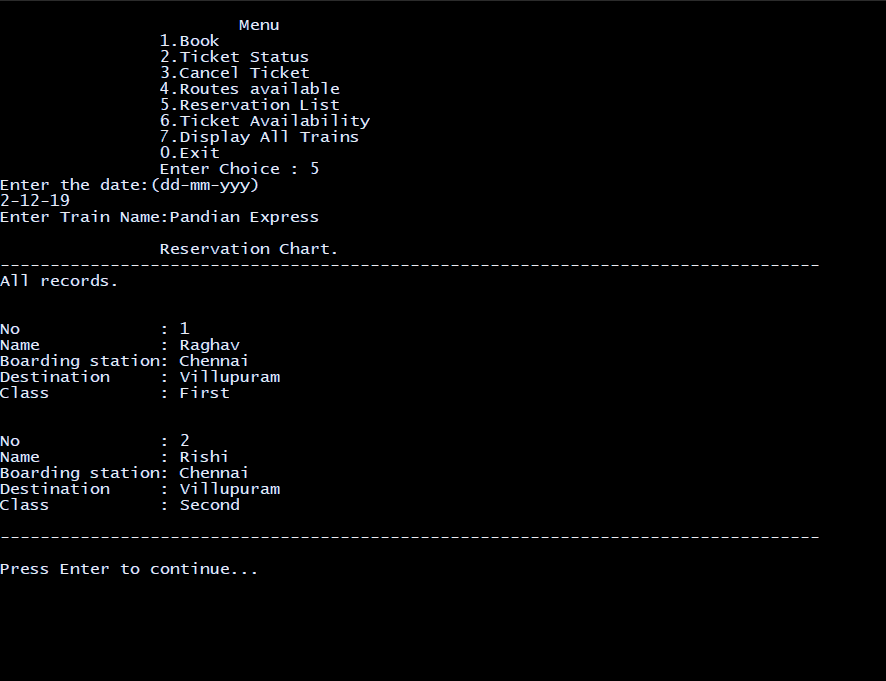
**Cancellation**



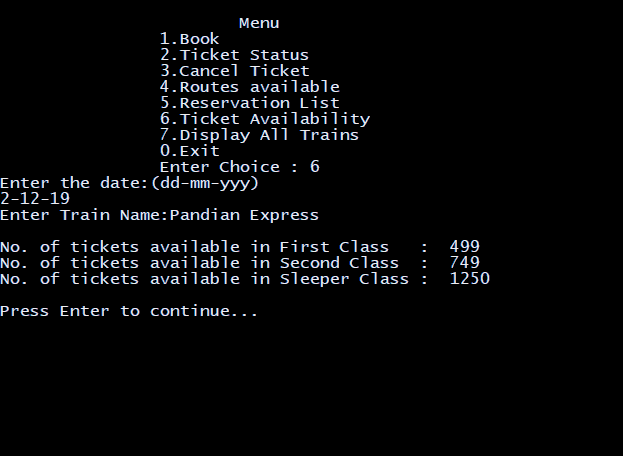
**Routes Available**



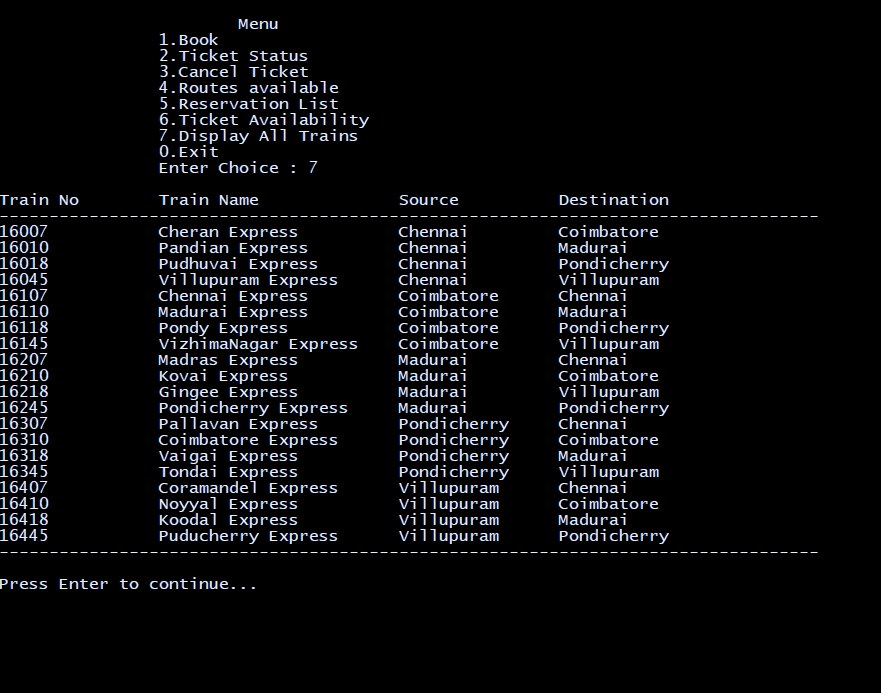
**Reservation list**



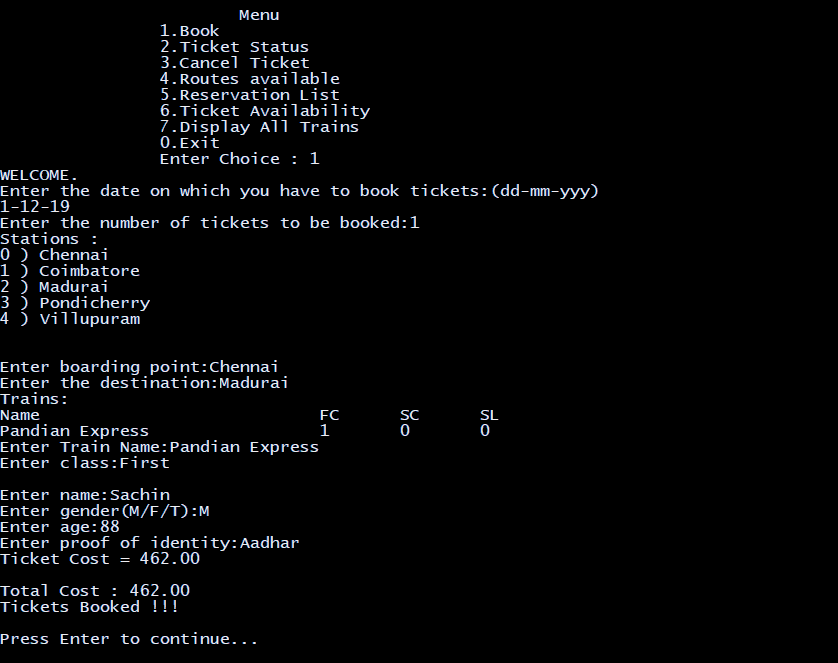
**Ticket Availability**

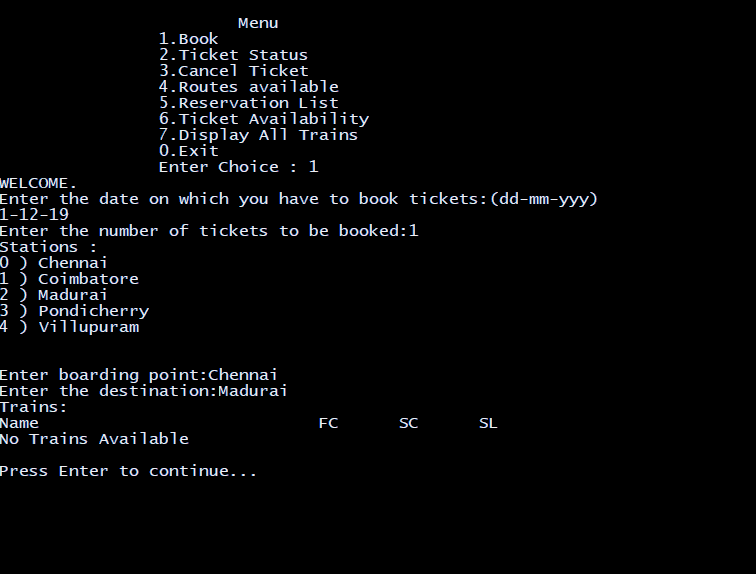


**All trains**



**No Trains Available**





**CONCLUSION**

The Railway Reservation System project is a simulation of a general Railway Ticketing website. It tries to accomplish several features of a real-world application with primary focus given to booking the of the railway. This is accomplished by applying data structures including graphs, linked lists, ADTs.