CHAPTER 1

INTRODUCTION

1.1 General Overview

The Railway Reservation System facilitates the passengers to enquire about the trains available and decide accordingly based on the source and destination, Booking and enquire about the status of the booked ticket etc. The main aim of this system is to design and develop file maintaining the records of different trains, train status and passengers details.

The Railway Reservation System needs the passengers to enter the ticket_id, passengers name, train number and destination for booking the ticket and to verify whether the ticket is booked or not the passenger needs to enter the ticket_id and train number to get the verification of ticket.

1.2 Problem definition

The Railway Reservation System includes various information regarding the passenger's. The Railway Reservation System needs details of the passenger's such as the ticket_id, passenger's name, train number and destination name. So that they can be kept track of their tickets.

This project helps to handle this problem in a efficient manner. In this the passenger's details are entered i.e. their ticket_id, name, train number and destination name and the entered details are then either confirmed or rejected based on the availability of the ticket_id sepecified or not. This project allow you to insert the passenger details and keep track of the passenger's and their details of the confirmed tickets.

This project also helps to check the ticket status, the passenger needs to enter the ticket_id and the train number. It also helps the authority to check the passenger details and create a confirmation list. This project displays the passenger details and confirmation list.

1.3 Objectives

The main objective is to create a file so that the details of the passengers can be recorded and maintained and used to verify for the confirmation of the ticket. This project is all about how to ease the problem of reservation and confirmation and reduce the human interventions and usage of papers and hand files in recording the details of passengers.

The Railway Reservation System facilitates the passengers to enquire about the trains available on the basis of source and destination, booking and confirmation of the tickets etc. The main aim was to create a graphical user interface so that the user can effectively interact with system and to implement the file handling using the concepts of B-Tree so that the passenger's details can be maintained and can be retrieved for the purpose of confirmation.

The record contains the ticket_id, passenger's name , train number and destination name of the confirmed passenger's ticket whereas the other record contains the ticket_id of the confirmed passenger's ticket.

Passengers can book their tickets for the train in which seats are available. For this passengers has to provide the desired train number and the ticket_id for which the ticket is booked. Before booking a ticket for the passenger, the validity of train number and booking date is checked. Once the train number and ticket_id are validated. If yes, the ticket is boked with confirm status and corresponding ticket_id is generated which is stored along with other details of the passengers. The ticket once booked cannot be cancelled at any time.

CHAPTER 2

HARDWARE AND SOFTWARE REQUIREMENTS

2.1 Hardware requirements

1. Processor: Pentium IV or Core i3, i5, i7, i9.
2. RAM: 2GB or higher.
3. HDD: 40GB or higher.
4. Display: 1024 * 768 Resolution Color Monitor.
5. Keyboard: 104 keys.

2.2 Software requirements

1. Operating System: Windows XP, 7, 8, 10.
2. IDE: Microsoft Visual Studio 2010 or higher version.

- The hardware requirements specified are the hardware components/capacity of the system in which the application is developed and deployed.
- The above software requirements are the necessary softwares required to develop the application and the run the application.
- Microsoft Visual Studio is used to develop the application on windows platform.
- The project is developed in C++ language with concepts of File handling and B-Trees.

CHAPTER 3

SYSTEM DESIGN

3.1 Context-flow Diagram

The context diagram is used to establish the context and boundaries of the system to be modelled: which things are inside and outside of the system being modelled, and what is the relationship of the system with these external entities.

A context diagram, sometimes called a level 0 data-flow diagram, is drawn in order to define and clarify the boundaries of the software system. It identifies the flows of information between the system and external entities. The entire software system is shown as a single process.

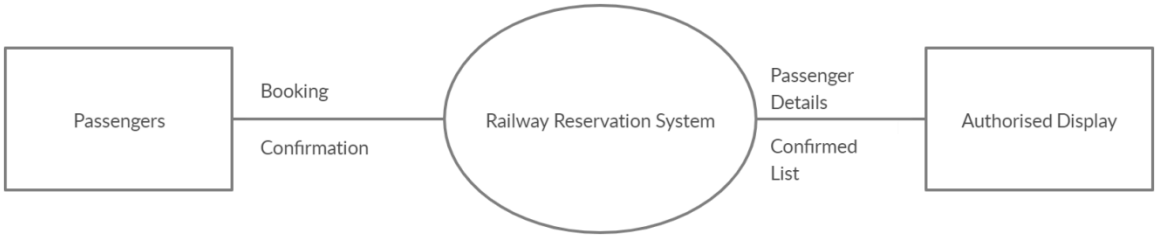

Fig 3.1 Context flow Diagram

3.2 SYSTEM DESIGN

System design is a solution, a “HOW TO” approach to the creation of a new system. It translates system requirements into ways by which they can be made operational. It is a translational from a user-oriented document to a document-oriented programmer. For that, it provides the understanding and procedural details necessary for the implementation. Systems design is the process of defining the architecture, modules, interfaces, and data for a system to satisfy specified requirements.

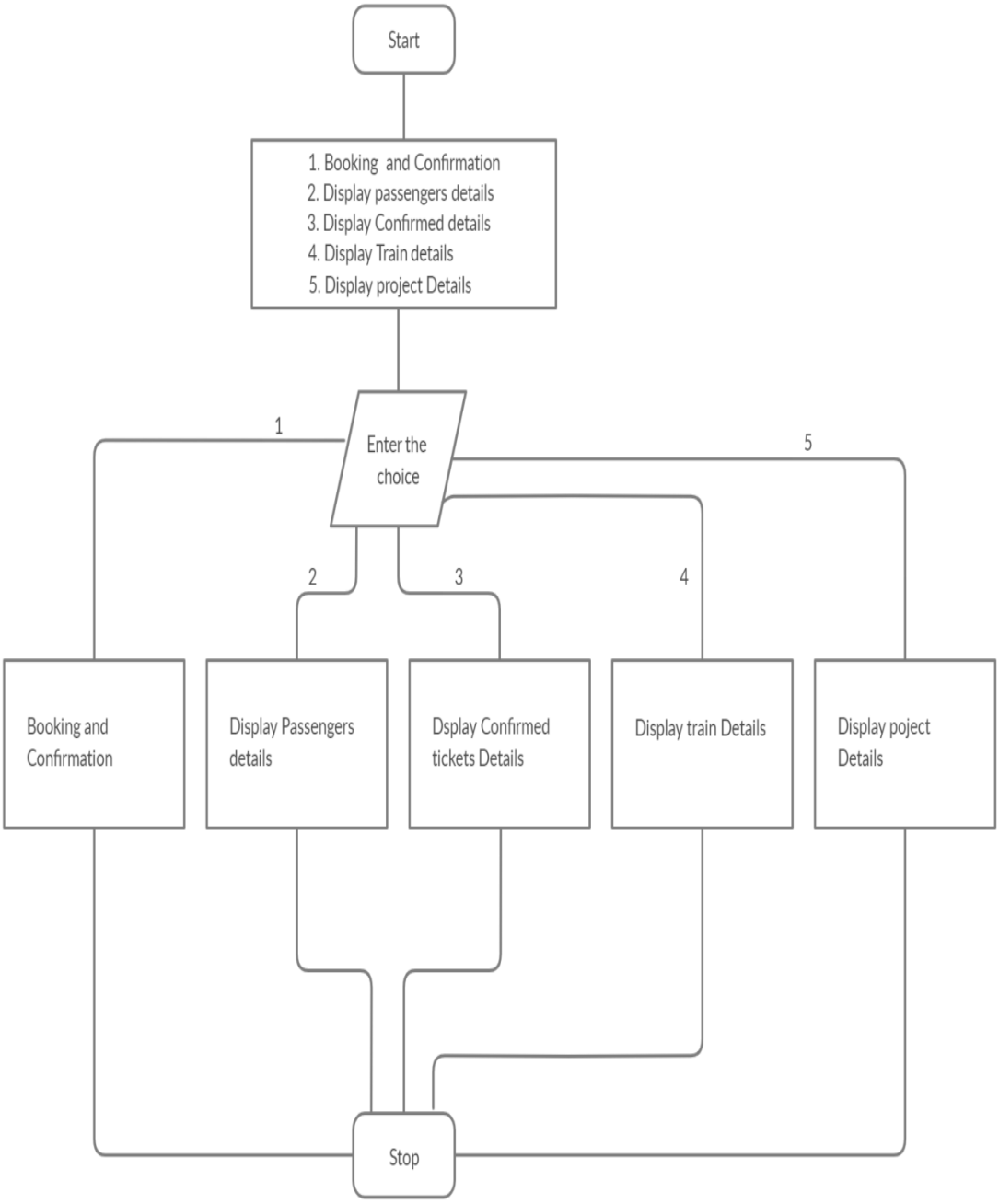

Fig 3.2 System Design for Railway Reservation System

3.3 Data-flow Diagram

A data-flow diagram (DFD) is a way of representing a flow of a data of a process or a system. The DFD also provides information about the outputs and inputs of each entity and the process itself. A data-flow diagram has no control flow, there are no decision rules and no loops. Specific operations based on the data can be represented by

a flowchart. Data flow diagrams are used to graphically represent the flow of data in an information system. DFD describes the processes that are involved in a system to transfer data from the input to the file storage and reports generation.

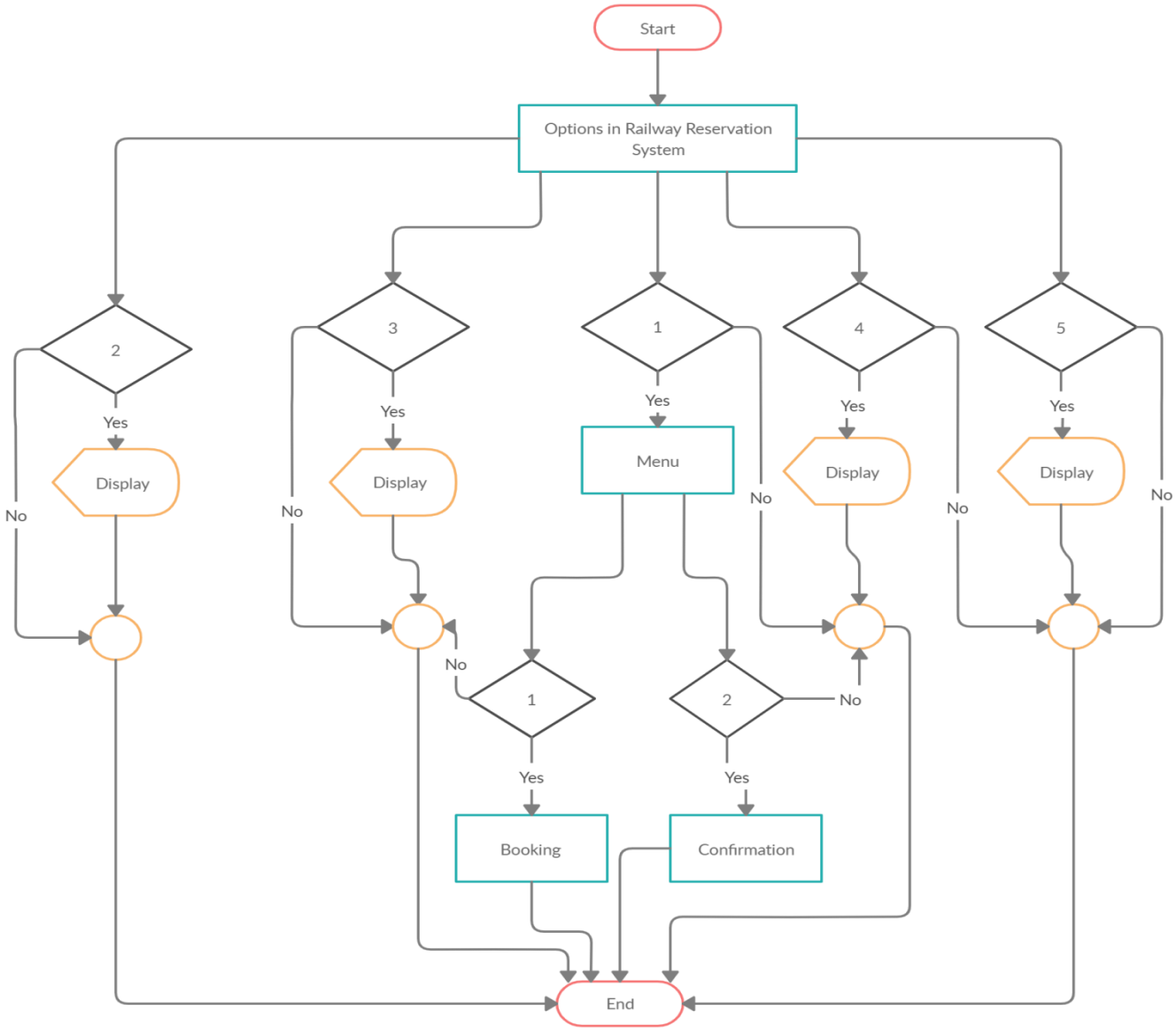

Fig 3.2 Data Flow Diagram for Railway Reservation System

3.4 Data-flow diagram for insertion

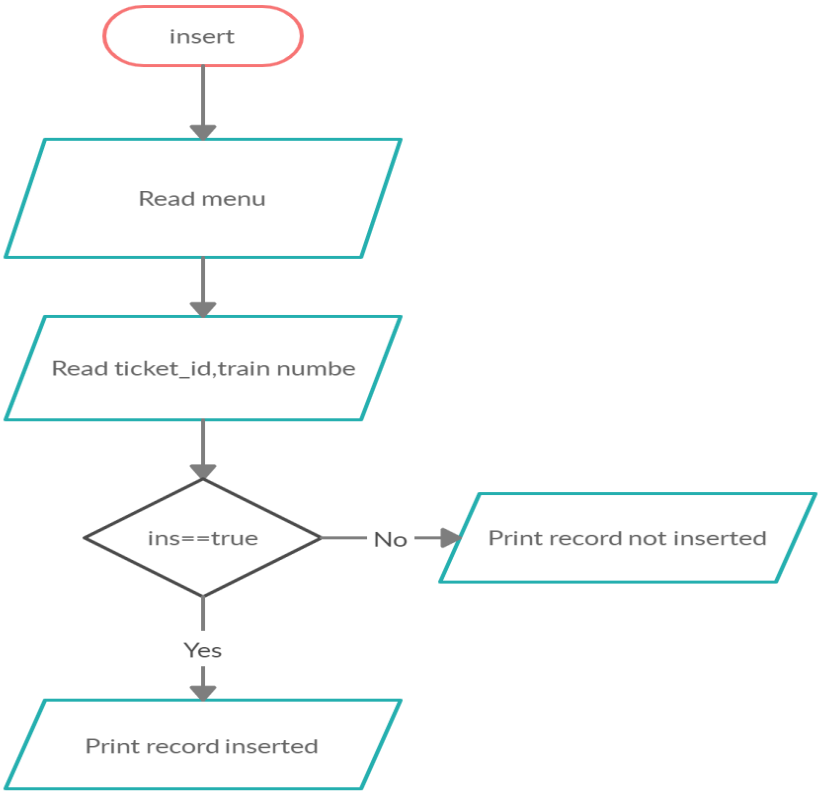

Fig 3.4 Data-flow diagram for insertion

In the Railway Reservation system, the passenger can insert the details like Ticket_ID, Passenger name, Train Number and Destination Name. Then the data is validated and checked for errors and repetition, if there are no errors then the data is saved to the file. The data that is saved in the file can be accessed by the authority later.

3.5 Data-flow diagram for search

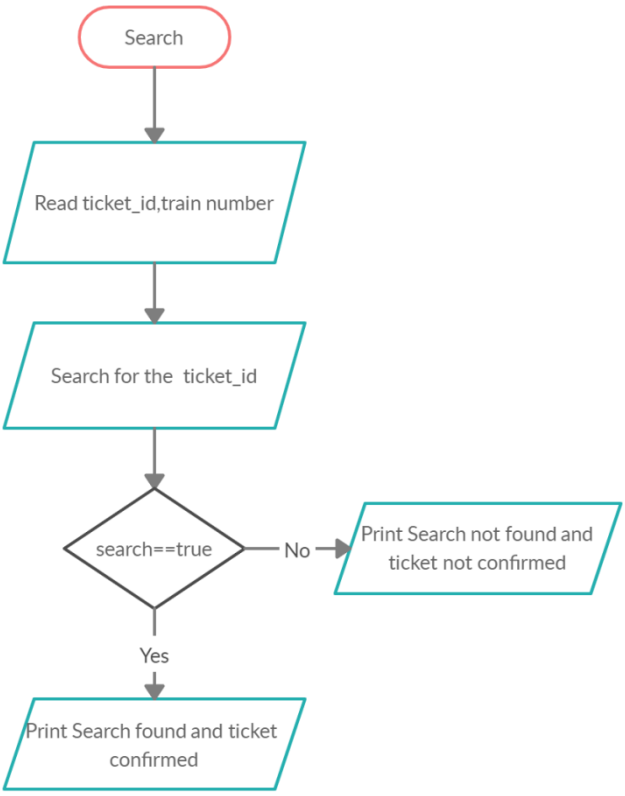

Fig 3.4 Data-flow diagram for search

In the Reservation Railway system, the passenger can search or check the status of the details of ticket by using ticket_id, if the ticket_id is present then the system displays that the ticket is confirmed and displays a message suitably and if the ticket_id is not present then the system will display suitable error message that is ‘ticket is not confirmed ‘ has to be displayed.

3.6 Data-flow diagram for display

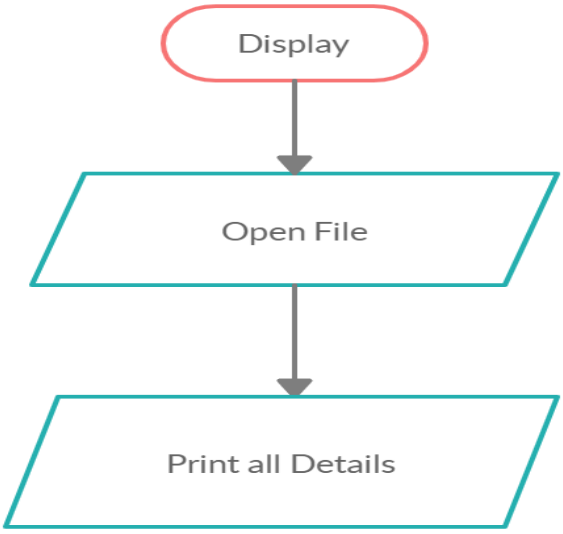

Fig 3.5 Data-flow diagram for display

In the Railway Reservation system, the authority can view all the details of the passengers and the confirmed ticket lists which are previously inserted through various options provided to view so. Both the options have the same data – flow as show in the diagram above. But the train details and the project details do not follow the above data-flow , they are the static pages.

CHAPTER 4

IMPLEMENTATION

4.1 B-TREE

B-Tree − B-tree is a self-balancing tree data structure that maintains sorted data and allows searches, sequential access, insertions, and deletions in logarithmic time.

4.1.1 B-TREE PROPERTIES

1) All leaves are at same level.

2) A B-Tree is defined by the term *minimum degree* 't'. The value of t depends upon disk block size.

3) Every node except root must contain at least t-1 keys. Root may contain minimum 1 key.

4) All nodes (including root) may contain at most 2t – 1 keys.

5) Number of children of a node is equal to the number of keys in it plus 1.

6) All keys of a node are sorted in increasing order. The child between two keys k1 and k2 contains all keys in the range from k1 and k2.

7) B-Tree grows and shrinks from the root which is unlike Binary Search Tree. Binary Search Trees grow downward and also shrink from downward.

8) Like other balanced Binary Search Trees, time complexity to search, insert, traverse and delete is O(Log n).

4.2 Algorithm for insertion

1) Initialize x as root.

2) While x is not leaf, do following

..**a)** Find the child of x that is going to be traversed next. Let the child be y.

..**b)** If y is not full, change x to point to y.

..**c)** If y is full, split it and change x to point to one of the two parts of y. If k is smaller than mid key in y, then set x as the first part of y. Else second part of y. When we split y, we move a key from y to its parent x.

3) The loop in step 2 stops when x is leaf. x must have space for 1 extra key as we have been splitting all nodes in advance. So simply insert k to x.

4.3 Algorithm for Search

The following algorithm is applied:

- Let the key (the value) to be searched by "k".
- Start searching from the root and recursively traverse down.
- If k is lesser than the root value, search left subtree, if k is greater than the root value, search the right subtree.
- If the node has the found k, simply return the node.
- If the k is not found in the node, traverse down to the child with a greater key.
- If k is not found in the tree, we return NULL.

4.4 Algorithm for traversing

Traversal is also similar to Inorder traversal of Binary Tree. We start from the leftmost child, recursively print the leftmost child, then repeat the same process for remaining children and keys. In the end, recursively print the rightmost child.

CHAPTER 5

SOFTWARE TESTING

Software Testing is an empirical investigation conducted to provide stakeholders with information about the quality of the product or service under test, with respect to the context in which respect to the context in which it is intended to operate. Software Testing also provides an objective, independent view of the software to allow the business to appreciate and understand the risks at implementation of the software. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding software bugs. It can also be stated as the process of validating and verifying that a software program/application/product meets the business and technical requirements that guide its design and development, so that it works as expected and can be implemented with the same characteristics.

5.1 Unit Testing

The software units in a system are modules and routines that are assembled and integrated to perform a specific function. Unit testing focuses first on modules, independently of one another, to locate errors. This enables, to detect errors in coding and logic that are contained within each module. The various controls are tested to ensure that each performs its action as required.

Sometimes software developers attempt to save time by doing minimal unit testing. This is a myth because skipping on unit testing leads to higher Defect fixing costs during System Testing, Integration Testing and even Beta Testing after the application is completed. Proper unit testing done during the development stage saves both time and money in the end.

5.2 Unit Testing Table

S.NO	INPUT	OUTPUT	REMARKS
1	As the insert operation is selected the system calls the insert() function.	As the insert operation is selected the system calls the insert() function.	Pass
2	Insertion: if special characters are given for ticket_id.	Data not allowed. And Error will be generated.	Pass
3	As the search operation is selected the system calls the search() function.	As the search operation is selected the system calls the search() function.	Pass
4	Search a record with ticket_id that does not exist.	Data does not exist. And Error message will be generated.	Pass
5	Display a record that exits	Record deleted successfully.	Pass
6	As the display operation is selected the system calls the display() function.	As the display operation is selected the system calls the display() function.	Pass
7	Checking for all menu items are working properly or not .	All menu items are working properly	Pass

Table 5.1 Unit testing

CHAPTER 6

RESULTS

The project is compiled and executed on Microsoft Visual C++. We have put in few screen shots in here to show the working of our Application.

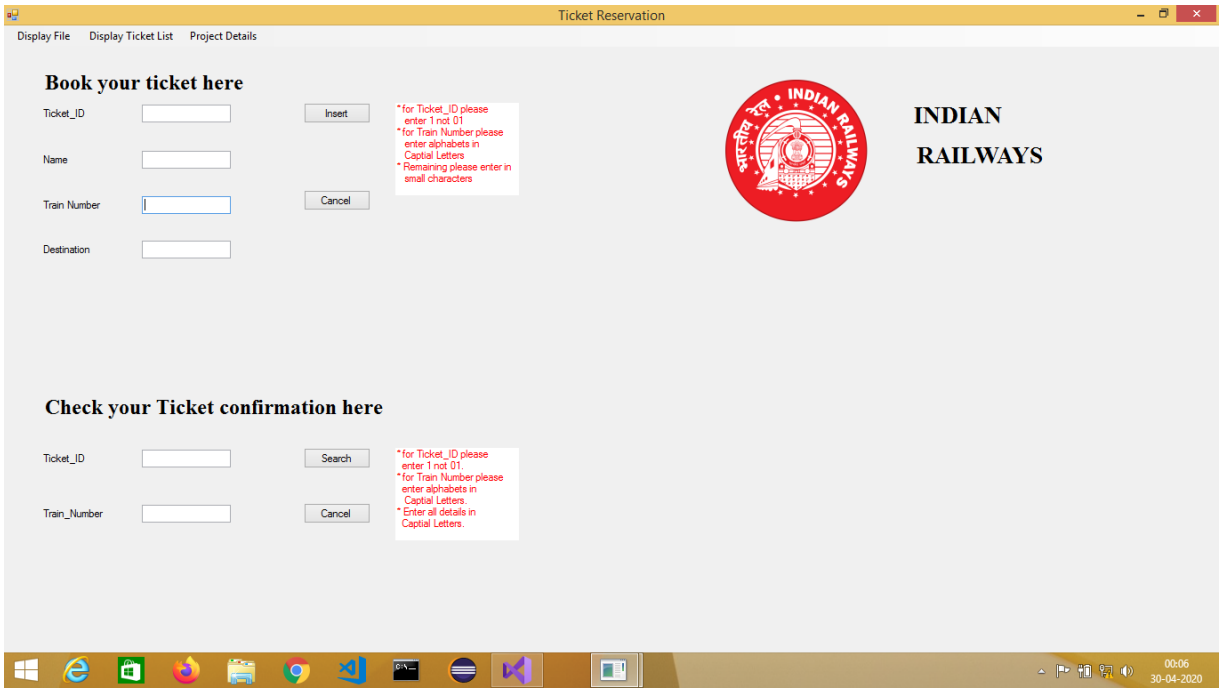

Fig 6.1 Railway Reservation System Main Menu

In the Railway Reservation system main menu page, the passengers and the authorities are provided with various options such as insert a new ticket, search, display all the contents of the passenger details and confirmation file present and then allow to display train status and project details and then finally exit the application. When the passenger clicks on the Display File menu item the passenger will be redirected into a new page, similarly when the passenger clicks on the Display Ticket List the passenger will be redirected to separate page. When the passengers click on the Project Details and the Train Details menu item they will be redirected to complete set of new static pages When the user clicks exit application button, the user will leave the application.

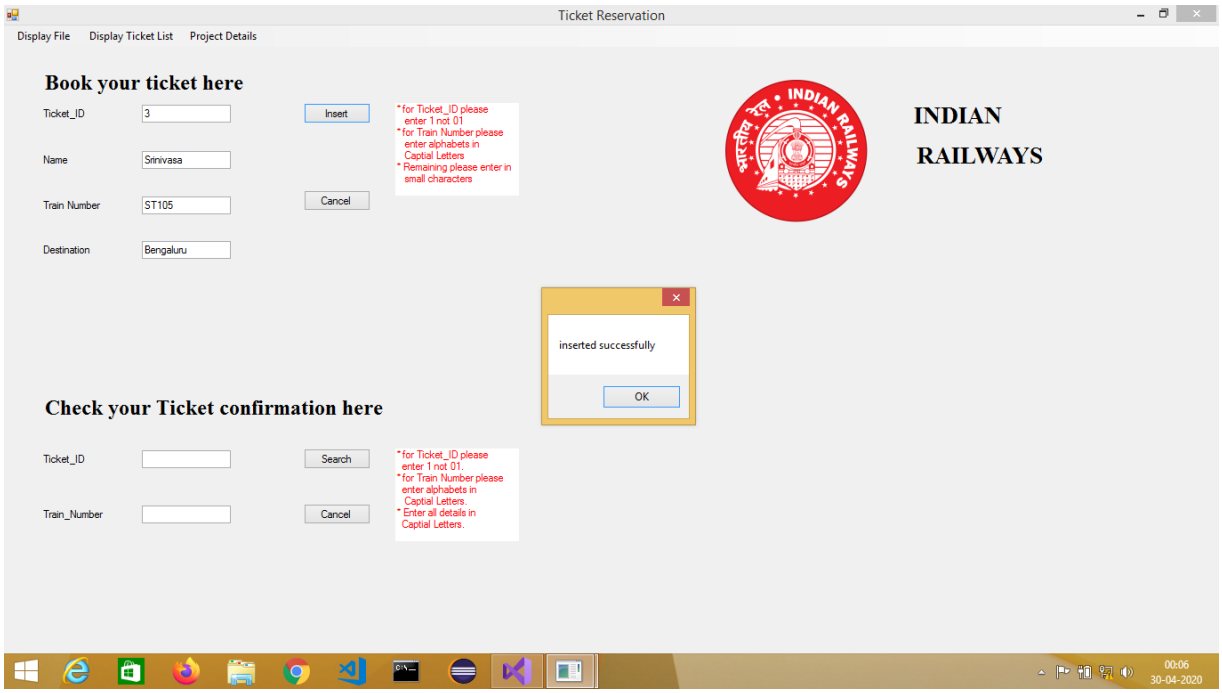

Fig 6.2 Inserting new passenger details

In BOOK YOUT TICKET column page the passenger needs to insert the new contact by adding person's details like Ticket_id, name, Train number and Destination. If the details are valid ticket is inserted and booked.

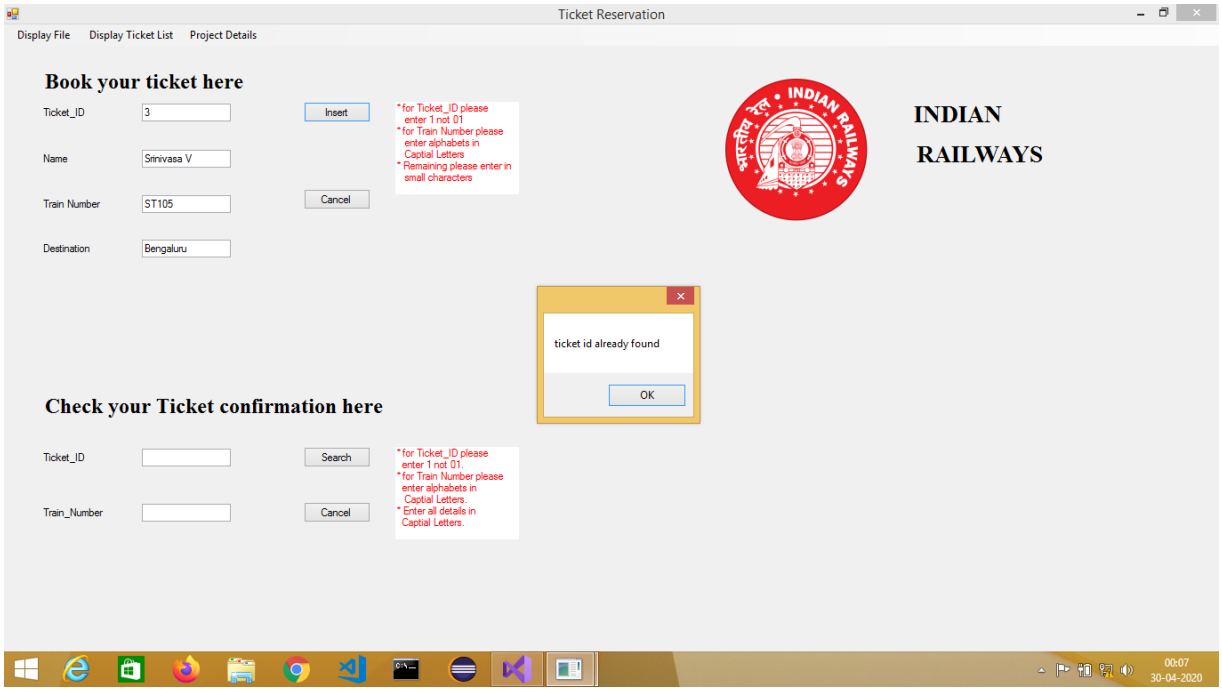

Fig 6.3 Repeatitive Ticket_ID inserted warning

If the passenger enters the repetitive ticket_id the railway reservation system will issue a warning about the same and asks the passengers to re-enter the details for ticket booking.

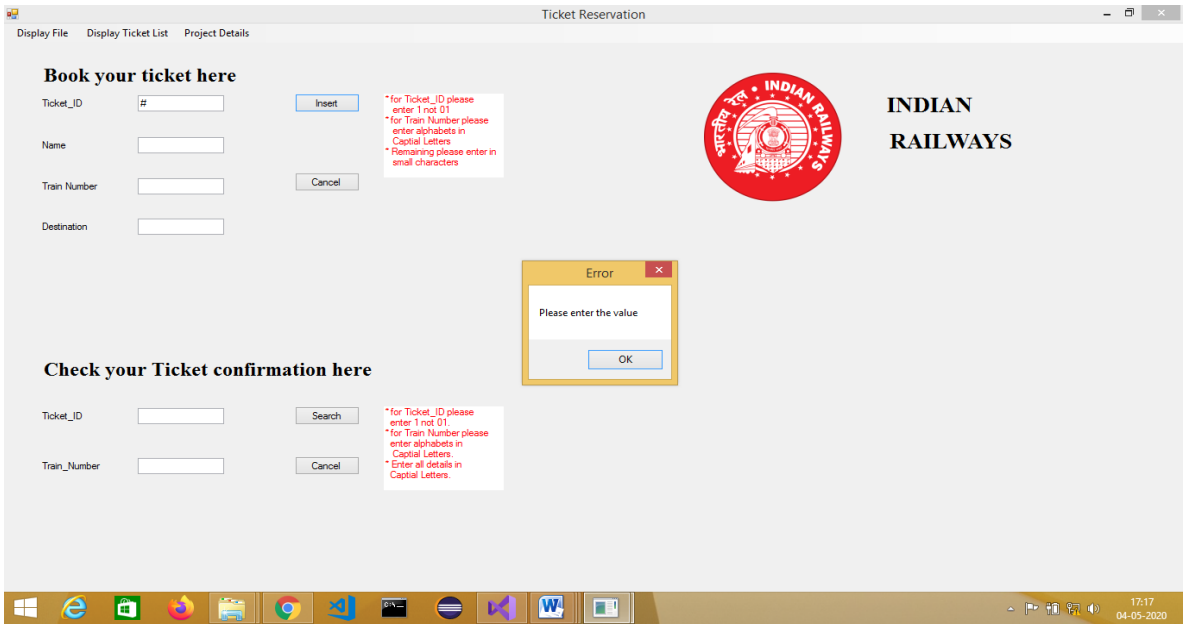

Fig 6.4 Display of error message

In the Railway Reservation System if the ticket_id is anything else then the numerical values and if any fields are left empty then the error message will be displayed.

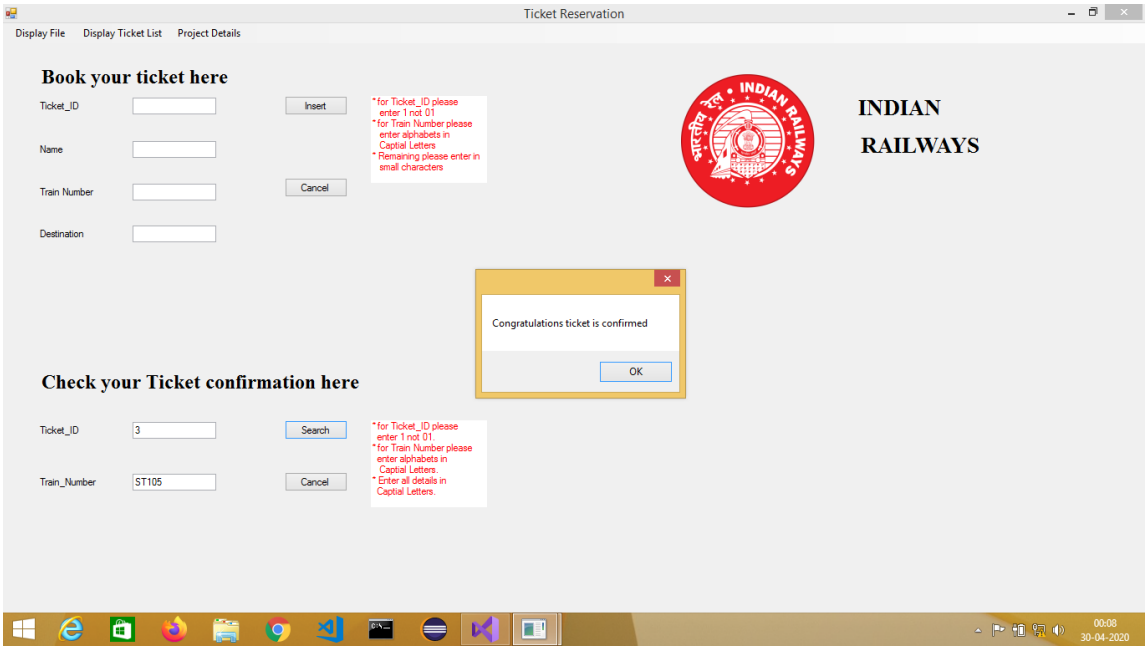

Fig 6.5 Ticket Confirmation Message

When the passengers enter correct id and the ticket is confirmed then this message will be formed.

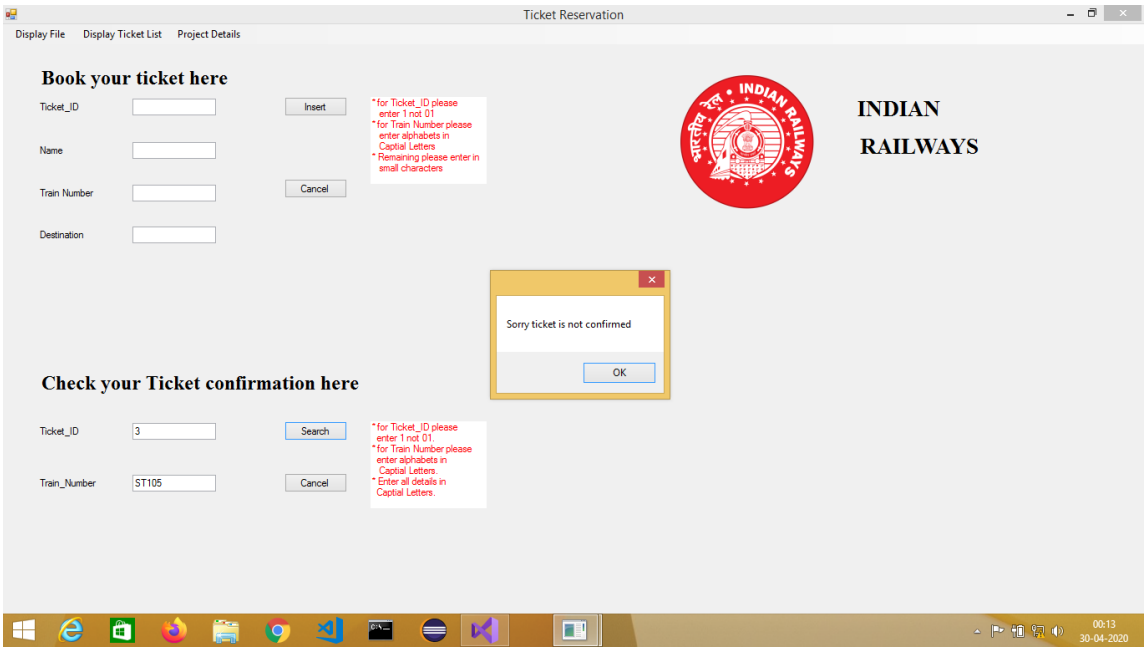

Fig 6.6 Ticket Non-Confirmation Message

If the ticket_id is repeated the ticket does not get confirmed and the above message will be displayed.

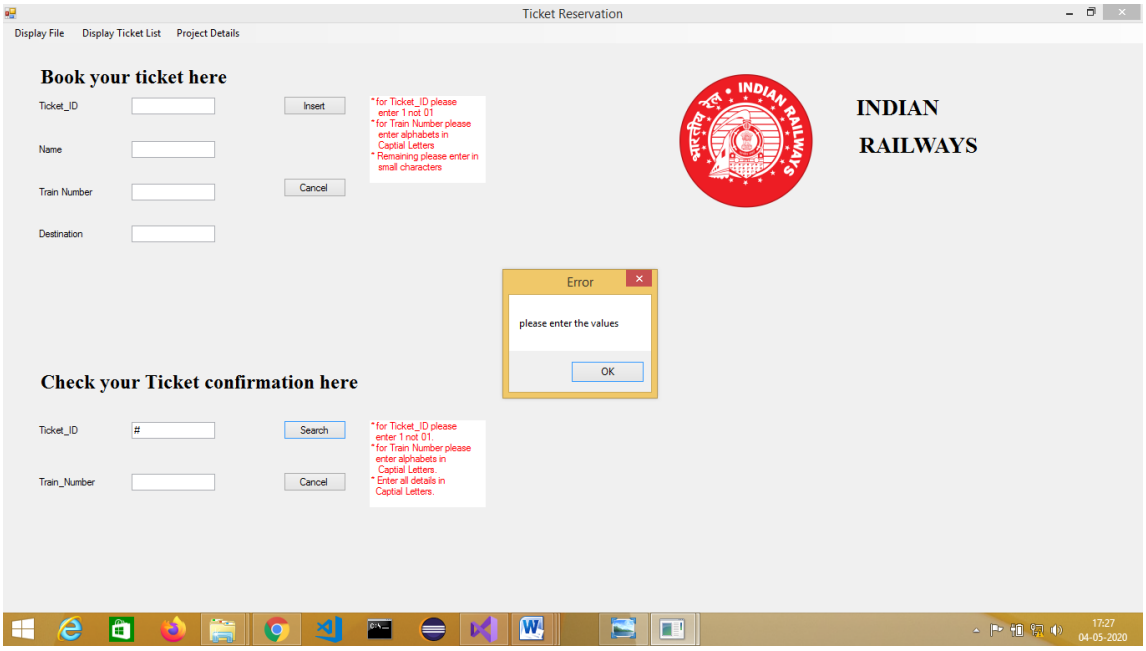

Fig 6.7 Ticket Confirmation Error Message

In the Railway Reservation System if the ticket_id is anything else then the numerical values and if any fields are left empty then the error message will be displayed.

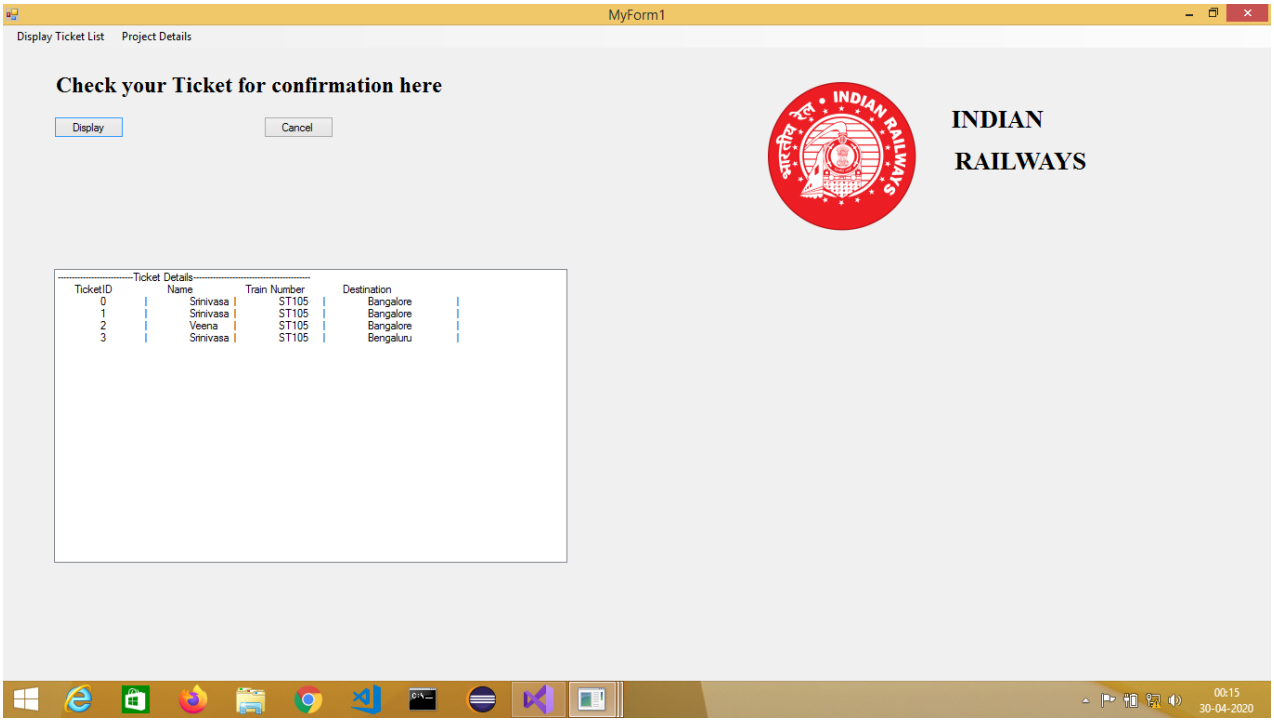

Fig 6.8 Passengers Details Display Form

This form gives the option to display the details of the passengers.

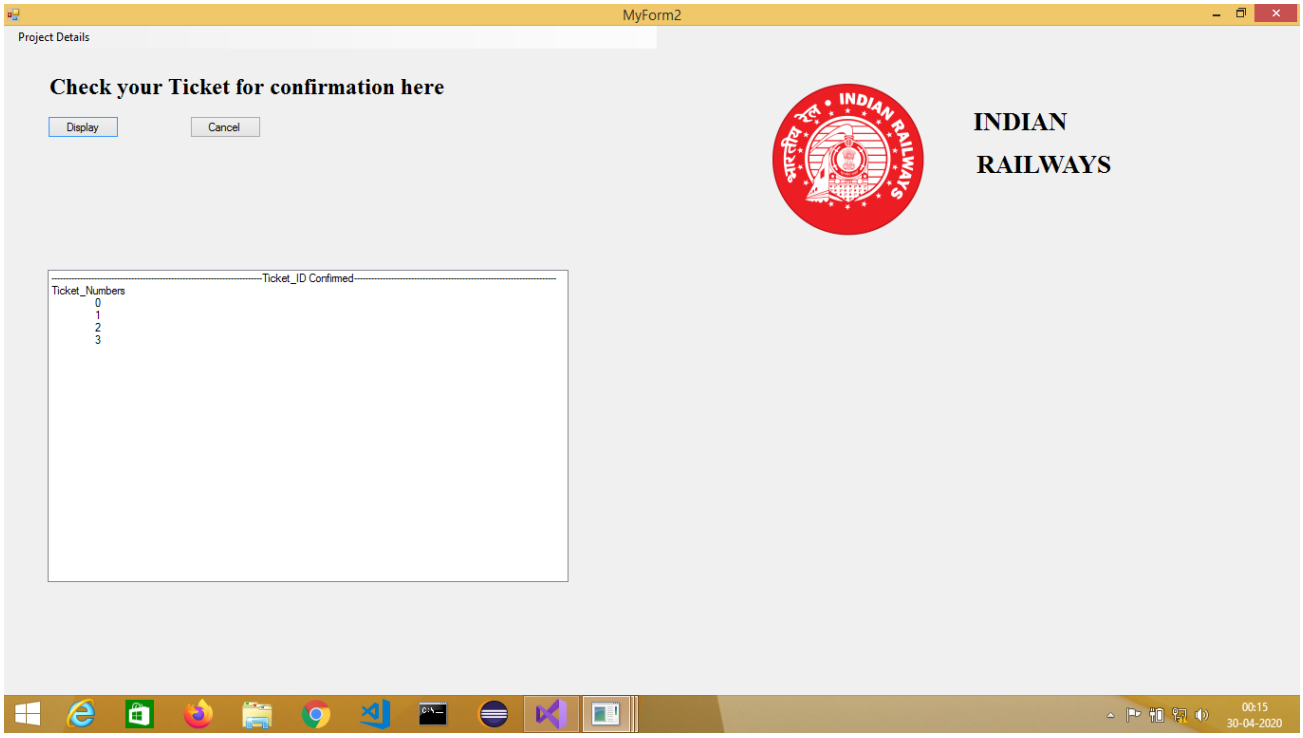

Fig 6.9 Confirmed Ticket Details Display Form.

This form is used to display the confirmed ticket details of the passengers.

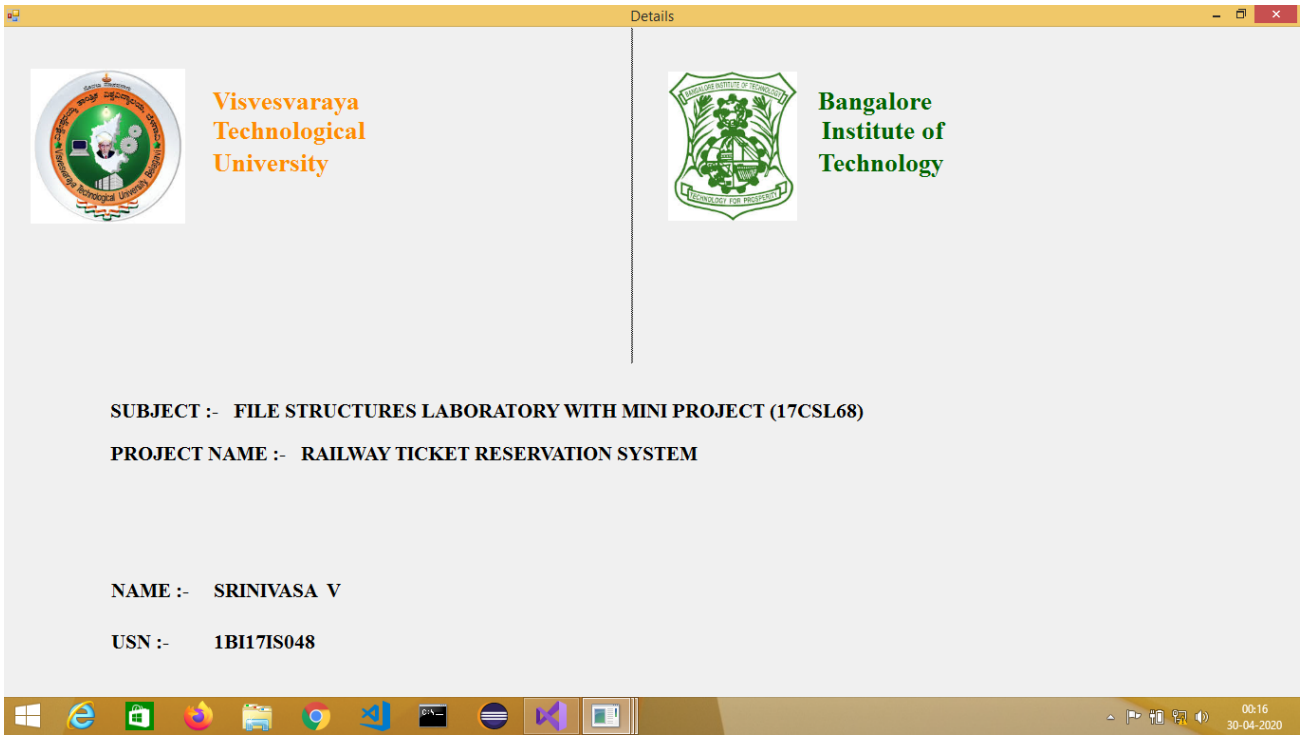

Fig 6.10 Displays the Project Details.

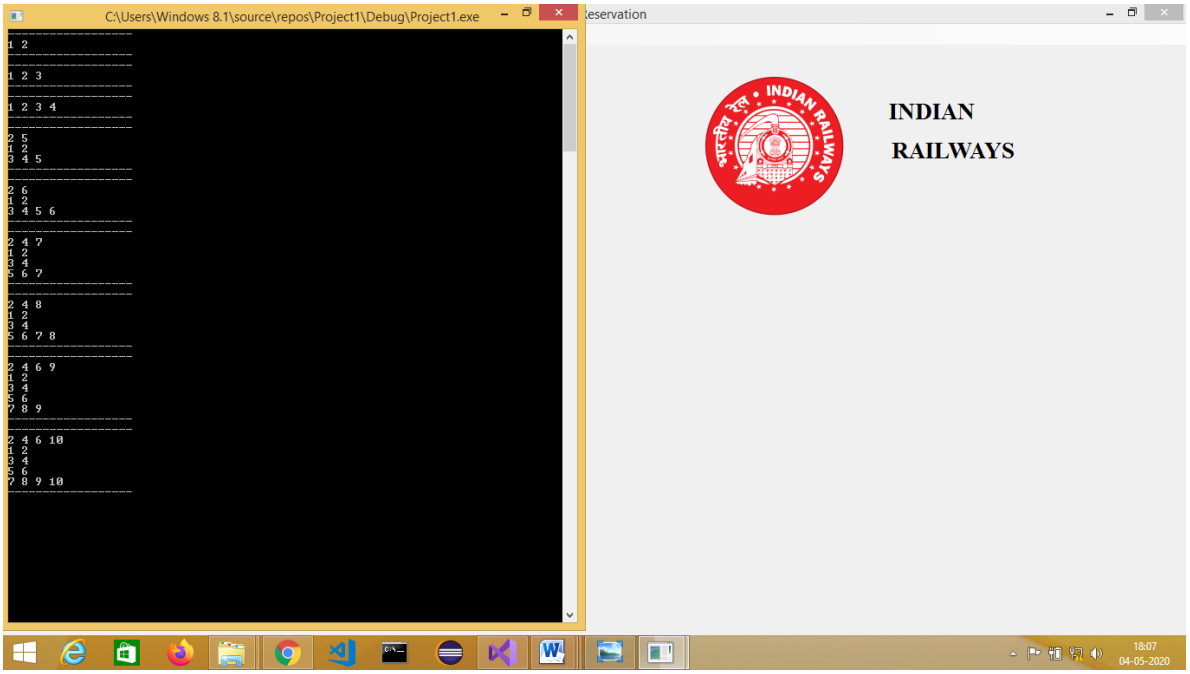

Fig 6.11 Display of the B-tree Structure (insertion operation).

This image displays the B-tree structure formed by the ticket_id when inserted from the graphical user interface. The tree structure is seen in console window.

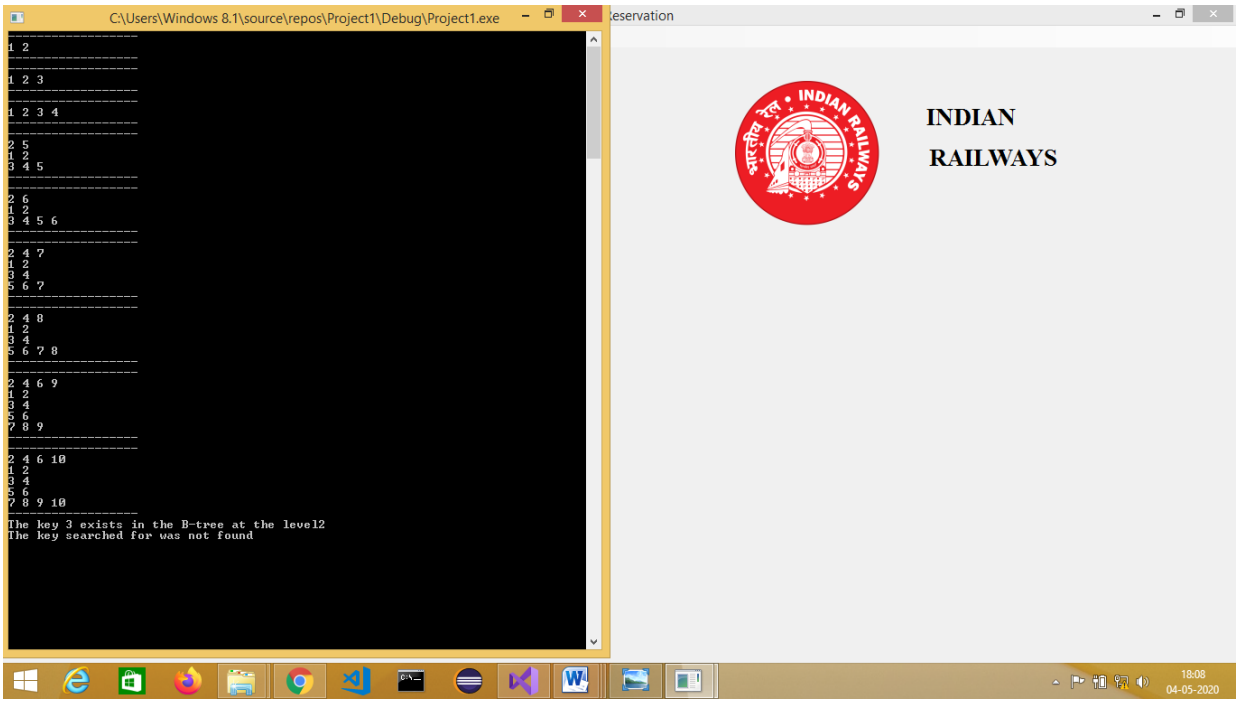

Fig 6.12 Display of the B-tree Structure (search operation).

This image displays the B-tree structure formed by the ticket_id when inserted from the graphical user interface. The tree structure is seen in console window and when the search is called the result is obtained in the console window.

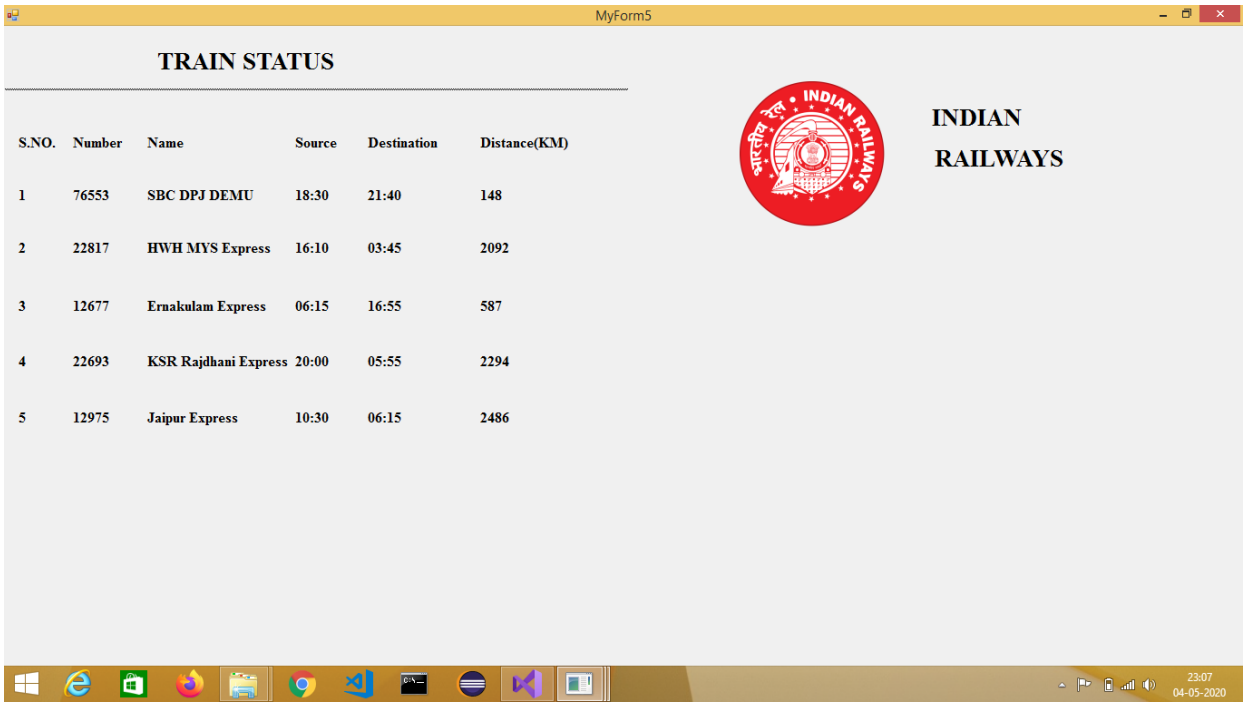

Fig 6.13 Display of the Train status.

Basically the trains are restricted to 5 and their statuses as well are displayed respectively.

CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENTS

This software is efficient in maintaining Passenger details and can easily perform operations on records and also works to handle the information of the contacts available in the application. This software also reduces the work load of the user by keeping track of all contacts electronically. It also reduces a lot of pen and paper work. In future, this system can be launched as a web site for easy online banking system management for any user.

The project has a very vast scope in future. The project can be implemented on internet in future. Projected can be updated in near future as and when requirement for the same arises, as it is very flexible in terms of expansion. With the proposed software of banking ready and fully functional user is now able to manage and hence run the entire work in a much better, accurate and error free manner.

REFERENCES

[1] Blog: Stack overflow

https://www.stackoverflow.com

[2] Michael J. Folk, Bill Zoellick, Greg Riccardi: File Structures-An Object-Oriented Approach

with C++,3rd Edition, Pearson Education, 1998.

[3] Google Search

https:// www.google.com

[4] Visual C++ Tutorial

https://youtu.be/YR6fxe1wa8g

[5] Geeks for Geeks

https://www.geeksforgeeks.com

APPENDIX A

BACK END CODE

```
#pragma once
#include<iostream>
#include<stdio.h>
#include<fstream>
#include<stdlib.h>
#include<string>
using namespace std;

class node
{
public:
	int a[4];
	node* next[4];
	node* parent;
	int size;
	node();
};
node::node()
{
	for (int i = 0; i < 4; i++)
		next[i] = NULL;
	parent = NULL;
	size = 0;
}
class btree
{

public:
	node* root;
	int dg=0, fg1=0;
	node* findLeaf(int key, int& level);
	void updateKey(node* p, node* c, int newkey);
	void search(int key);
	void insert(int key);
	void insertIntoNode(node* n, int key, node* addresss);
	void promote(node* n, int key, node* addresss);
	node* split(node* n);
	void traverse(node* ptr);
	btree();
};
```

```
void btree::traverse(node* ptr)
{
	if (ptr == NULL)
		return;
	for (int i = 0; i < ptr->size; i++)
		cout << ptr->a[i] << " ";
	cout << endl;
	for (int i = 0; i < ptr->size; i++)
		traverse(ptr->next[i]);
}
btree::btree()
{
	root = NULL;
}
node* btree::findLeaf(int key, int& level)
{
	node* ptr = root;
	node* prevptr = NULL;
	level = 0;
	int i;
	while (ptr)
	{
		i = 0;
		level++;
		while (i<ptr->size - 1 && key>ptr->a[i])
			i++;
		prevptr = ptr;
		ptr = ptr->next[i];
	}
	return prevptr;
}
node* btree::split(node* n)
{
	int midpoint = (n->size + 1) / 2;
	int newsize = n->size - midpoint;
	node* newptr = new node;
	node* child;
	newptr->parent = n->parent;
	int i;
	for (i = 0; i < midpoint; i++)
	{
		newptr->a[i] = n->a[i];
		newptr->next[i] = n->next[i];
		n->a[i] = n->a[i + midpoint];
		n->next[i] = n->next[i + midpoint];
	}
	n->size = midpoint;
	newptr->size = newsize;
	for (i = 0; i < n->size; i++)
```

```
	{
		child = n->next[i];
		if (child != NULL)
			child->parent = n;
	}
	for (i = 0; i < newptr->size; i++)
	{
		child = newptr->next[i];
		if (child != NULL)
			child->parent = newptr;
	}
	return newptr;
}
void btree::updateKey(node* parent, node* child, int newkey)
{
	if (parent == NULL)
		return;
	if (parent->size == 0)
		return;
	int oldkey = child->a[child->size - 2];
	for (int i = 0; i < parent->size; i++)
		if (parent->a[i] == oldkey)
		{
			parent->a[i] = newkey;
			parent->next[i] = child;
		}
}
void btree::insertIntoNode(node* n, int key, node* address)
{
	int i;
	if (n == NULL)
		return;
	for (i = 0; i < n->size; i++)
		if (n->a[i] == key)
			return;
	i = n->size - 1;
	while (i >= 0 && n->a[i] > key)
	{
		n->a[i + 1] = n->a[i];
		n->next[i + 1] = n->next[i];
		i--;
	}
	i++;
	n->a[i] = key;
	n->next[i] = address;
	n->size++;
	if (i == n->size - 1)
		updateKey(n->parent, n, key);
}
```

```
void btree::promote(node* n, int key, node* address)
{
	if (n == NULL)
		return;
	if (n->size < 4)
	{
		insertIntoNode(n, key, address);
		return;
	}
	if (n == root)
	{
		root = new node;
		n->parent = root;
	}
	node* newptr = split(n);
	node* t;
	if (key < n->a[0])
		t = newptr;
	else
		t = n;
	insertIntoNode(t, key, address);
	promote(n->parent, n->a[n->size - 1], n);
	promote(newptr->parent, newptr->a[newptr->size - 1], newptr);
}
void btree::insert(int key)
{
	if (root == NULL)
	{
		root = new node;
		root->a[root->size] = key;
		root->size++;
		return;
	}
	int level;
	node* leaf = findLeaf(key, level);
	int i;
	for (i = 0; i < leaf->size; i++) {
		if (leaf->a[i] == key)
		{
			fg1 = 1;
			cout << "The key to be inserted already exists" << endl;
			return;
		}
	}
	promote(leaf, key, NULL);
	cout << "------------------\n";
	traverse(root);
	cout << "------------------\n";
```

```
}

void btree::search(int key)
{
	dg = 0;
	if (root == NULL)
	{
		cout << "The tree does not exist" << endl;
		return;
	}
	int level;
	node* leaf = findLeaf(key, level);
	int flag = 0;
	for (int i = 0; i < leaf->size; i++) {
		if (leaf->a[i] == key)
		{

			flag = 1;
			dg = 1;
			cout << "The key " << key << " exists in the B-tree at the level" << level <<
	endl;
		}
	}
	if (!flag) {

		cout << "The key searched for was not found" << endl;
	}
}
/*
int main()
{
	btree b;
	int choice = 1, key;
	while (choice <= 2)
	{
		cout << "1.Insert a key\n";
		cout << "2.Search a key\n";
		cout << "3.Exit\n";
		cout << "Enter your choice: ";
		cin >> choice;
		switch (choice)
		{
		case 1: cout << "Enter the key to be inserted in a B-tree\n";
			cin >> key;
			b.insert(key);
			break;
		case 2: cout << "Enter the key to be searched\n";
			cin >> key;
			b.search(key);
```

```
                        break;
                }
        }
        return 0;
}
*/
```