

TRAVEL AGENCY CRM
FLIGHT MANAGEMENT SYSTEM
20K-0325, 20K-0313, 20K-0470

Introduction:

The project we have developed over the past few months aims to replicate travel agency CRM software capable of flight reservation, administration, and customer profile management, all under one roof. The motivation was to deliver a seamless flight reservation experience to the consumer with minimal technical bottlenecks and maximum customer interactivity and transparency.

Background:

Flight management was selected for its manageability, modular nature, and the sheer amount of data associated with complex flight reservation. The algorithms and logic demanded by this application were perceived as rather straightforward and feasible which led us to our choice. Research was done by surveying various existing software such as TravelPort GDS and Tutterfly

Project Specification:

Our solution allows users to store their login information which is associated with the data entered for each booking they will make. New users will sign up for an account using the Sign Up feature. Subsequently, if users have already made existing reservations through their account, they will be able to view all specific details for each and every booking created in a convenient tabular form, which includes tickets for a booking. User will have the freedom to select from any of the present routes which will each have their own route charges. Optional

feedback module has been provided as well. Every booking will ask for experience rating from customer for the sake of updating flight ranking information. Database management will be done through administrator portal.

Problem Analysis:

- Storing customer information on disk
- Storing data for flights, routes, feedbacks, bookings, and rankings on disk
- Linking data for flights, routes, feedbacks, bookings, and rankings to each other for functionality
- Updating information as necessary in files
- File handling
- Record I/O
- Accessible user interface through console (menus)
- Assigning new and unique ids for every new customer whose information is to be stored
- Unique ID assignment for Flights and Bookings as well
- Dynamic declaration due to changing file size

Solution design:

Features:

- Add feedback portal for customers
- ▪ Add ability to upgrade seating plan
- ▪ Rank airlines in terms of user feedback

- ▪ Prompt user to repeatedly enter seat reservation details using loop control structures
- ▪ Use of conditional statements to provide user the option to upgrade current seat to business class.
- ▪ Sort airlines based on key words written in user feedback.

➤ **Users:**

- ▪ log in using username and password. This will be stored in separate file.
- ▪ Select a destination and origin airport for route
- ▪ Select particular flight, if available, operating on that route
- ▪ View flight seating plan
- ▪ View updated pricing for type of seat booked: economy, business, first class
- ▪ Prompt user to return if fully booked
- ▪ Select number of seats to be booked (not more than 6)
- ▪ Enter details for each seat booked (Name, Individual/Group). Seat could be in economy, business class, or first class.
- ▪ Offer complementary upgrade from economy to business class if seats available.
- ▪ Type of seat: aisle or window
- ▪ Select meal plan for all tickets for each reservation
- ▪ Users can view booked flights and current status of payment
- ▪ Submit written feedback
- ▪ Select from a number of tour packages offering ingoing and return tickets for varying
- discount depending on ticket quantity
- ▪ View each airlines' ranking and stored information such as rating frequency.
- Update password for account

➤ **Administrator:**

- ▪ Logs in using pin number

- ▪ Can view all user accounts and their reservation status (reserved, paid)
- ▪ Add a new flight plan (origin, destination, route)
- Update/Reset flight rankings
- Add new routes
- Add flights for existent routes
- Update ticket prices

Functionality:

- Store all customer and system data in **binary files** for security
- Usage of **structs** to store all data and write it to file
- Create **separate files** for each of Booking, Flight, Route, Customer, Rankings, Tickets, and Feedbacks
- Use of **bubble sort** algorithm to arrange rank records in ascending order
- Use of modular approach to entire program
- All features for both **admin** and **user** have been written as functions
- **File nesting** implemented
 - /Route/Flight/Booking etc.
- Algorithms to **prevent redundancy** i.e., record duplication.
- Ask for input (could be flight or route code), **search** the file once for the data entered. If found, then **flag** as found and proceed with rest of program. If not found, allow user to **retry** and **reenter** the data, otherwise **quit** the **module**
- **Assignment of unique IDs:** Declare **constant** value for first ever customer, flight, and booking in their respective functions. When next entry will be made, last customer id or booking id stored in file will be **incremented** by 1 and then assigned to new user to maintain **uniqueness** of ids and **data integrity** and to avoid data redundancy
- Use of **getchar** function to enable hidden password entry
- Use of **fseek** to update an existing record in **rb+** mode by setting offset equal to size of the struct being used (fseek (fptr, -sizeof(struct), SEEK_CUR))
- Use of **ftell** function to determine file size (first seek to end of file)

- Added separate conditional block for data being written for the first time i.e.:
 - **First** route
 - **First** flight
 - **First** booking
- **Organized** code by writing validation statements as separate functions. For instance, **int** getpin() method that takes admin pin code as input from stdin. Another such function would be **void** checkflightcode()
- Booking data, Flight data, Customer data, and that for feedbacks is all tied together with a **single variable**
- Use of pointer to **FILE** pointer to allow File I/O outside “**main()**” function (**FILE **pointer**)
- Use of **text filing** to store ticket prices for:
 - Economy (least expensive)
 - Business
 - First (most expensive)
- Use of **typedef** syntax to shorten structure variable declaration
- Use of **Sleep()**, **getch()** and **system(“cls”)**; functions to enhance usability and user-friendliness of program (**stdlib.h**)
- Use of case statements to display menu for user and admin portals within **while** loops
- Use of **dynamic structure array** to store booking records for a customer (count variable used to adapt to newer records). User asked to enter serial number associated with a particular booking. Upon correct input, all stored ticket information for each of the tickets booked will be displayed in **tabular form**.
- **Ranking** implemented by totaling all ratings of users for a particular flight code and then applying **sorting**
- Use of **nested loops** and **nested conditional statements** to enable data validation and **iterative** data input
- Add **menu** for both USER and ADMIN Portals

ADMIN PORTAL:

- Add a route
- Add a flight
- Display Routes
- Display Flights
- View ticket price
- Update ticket price
- Manage customer accounts
- Update/Reset Flight Rankings
- Logout

USER PORTAL:

Following is the list of all features present in the program as functions:

- Create a booking
- View account information
- View updated ticket prices
- View route information
- View booking information
- FEEDBACK PORTAL
- View Updated Ticket Prices
- Logout

All Functions Used in Program (19):

```

9  int cno=1110;
10
1  int ulogin(FILE **ptr,FILE **fp,char* fname,char* cfname,char* str,int* cval);
2  void currenttime();
3  int getpin();
4  void displayadminmenu();
5  void addroute(FILE **routeptr);
6  void addflight(FILE **flightptr,FILE **routeptr,FILE **rankptr);
7  int checkroute code(char* code);
8  void viewrouteprice(FILE **routeptr);
9  void viewticketprice(FILE **ticketptr);
10 void changeticketprice(FILE **ticketptr);
11 void displayusermenu();
12 void viewaccountinfo(FILE **ptr,int cnumber);
13 void addbookings(FILE **ptr,FILE **ticketptr,FILE **routeptr,FILE **flightptr,FILE **bookptr,FILE **tickptr,int customernumber,FILE **rankptr);
14 void checkflightcode(char* fcode);
15 void viewbookinginfo(FILE **bookptr,FILE **tickptr,int custno);
16 void displaycustinfo(FILE **ptr);
17 void viewadminflight(FILE **flightptr);
18 void addfeedbacks(FILE **feedptr,FILE **bookptr,int custno);
19 void adminrank(FILE **flightptr, FILE **rankptr);
20 void viewranks(FILE **rankptr);
21
22

```

All Structures Used in Program:

- **Feedback:** Used for storing 149-character long string for a particular booking.
 - typedef struct Feedback{
 - char feedstr[150];
 - int booknum;
 - }feedbacks;
- **Rankings:** Used for storing rank data for each flight code. Rank comprises of total points and total number of different types of ratings for each flight. Consists of the nested structure ratings that contains count of different rating types (1* to 5*).
 - typedef struct rankings{
 - char fcode[30];
 - int pts;
 - ratings review;
 - }rankings;
 -
 - typedef struct Feedback{
 - char feedstr[150];
 - int booknum;
 - }feedbacks;
- **Route:** Used to store a particular route along with its origin, destination, unique route code, and price.
 - typedef struct Route{
 - char origin[20];
 - char destination[20];
 - char routecode[20];

- int routeprice;
- }routes;

- **Flight:** Used to store flight(s) for a particular route code. Also indicates current seat capacity and whether flight is fully booked.
 - typedef struct Flight{
 - char flightcode[7];
 - int tseats;
 - int full;
 - int seats[4][50];
 - char froute[20];
 - }flights;

- **Tickets:** Used to store ticket information for each ticket booked for a certain booking id.
 - typedef struct Ticket{
 - int bookno;
 - int seatno;
 - int rowno;
 - char flightno[7];
 - char fclass;
 - char meal;
 - }tickets;

- **Bookings:** Used to store a unique booking with pricing and associated customer id
 - typedef struct Booking{
 - int cnum;
 - int bookno;
 - int bookcost;
 - char origin[20];
 - char destination[20];












- }bookings;

- **Customer:** Used to store all information for a customer on disk. Consists of unique customer id and name strings
 - typedef struct Customer{
 - int cnum;
 - char uname[30];
 - char upass[30];
 - char fname[40];
 - char lname[40];
 - int cardnum;
 - int costdue;
 - }customers;

Implementation and Testing:

Testing: Involved both blackbox and whitebox testing. Data for routes, flights, and bookings by customers was entered on console and outputs were observed. Subsequently, corrections in the code were made. Error detection and correction process was made simpler through use of comments before the beginning of experimental code scattered throughout the program. Most common errors included comparison and logical operators in conditional statements, with ampersand errors also arising occasionally in scanf and fscanf statements. All loops and rogue conditions were tested using boundary data, extreme data, and erroneous data as well. Below are the files created throughout the execution of the program

Files Generated by Program

 adfa	1/20/2021 9:54 PM	C Source File	83 KB
 adfa	1/20/2021 9:59 PM	Application	177 KB
 BookingInfo.dat	1/20/2021 10:03 PM	DAT File	1 KB
 CustomerInfo.dat	1/20/2021 10:03 PM	DAT File	1 KB
 Feedbacks.dat	1/20/2021 10:00 PM	DAT File	1 KB
 FlightsInfo.dat	1/20/2021 10:03 PM	DAT File	12 KB
 Rankings.dat	1/20/2021 10:03 PM	DAT File	1 KB
 RoutesInfo.dat	12/19/2020 5:03 PM	DAT File	1 KB
 TicketingInformation.dat	1/20/2021 10:02 PM	DAT File	1 KB
 TicketsPrice	12/19/2020 5:07 PM	Text Document	1 KB
 UserInfo	12/12/2020 4:20 PM	Text Document	1 KB

Breakdown and Timeline:

Administrator portal was drafted in the very first phase of development. This was proceeded by integration of admin and user login portals into **main()** function. Afterwards, the user login function **int ulogin();** was created, first as a rough model for the storage of customer accounts using simple variables and txt filing, but was later transformed into robust and secure structure driven data that stored all customer records in the file **CustomerInfo.dat**. Later, administrator module was developed, with Routes and Flights entry functions being the first to be written.

Timeline:

November 14th:

- Function ulogin() fully developed.
- Contributions: Ali Nadir

December 1st:

- Function addroute() fully developed:

- Ali Nadir and Ahmed Abdullah.

December 3rd to 5th:

- Function addflight() fully developed:
- Ali Nadir and Imtiaz Ali.
- Tested until December 5th.

December 6th:

- Function changeticketprice() fully developed:
- Ali Nadir.

December 6th to 7th:

- Function viewticketprice() fully developed:
- Ali Nadir, Ahmed, Imtiaz.
- Tested till December 7th.

December 8th to 10th:

- Functions Display Routes and Display Flights added to admin module.
- Developed by Imtiaz Ali and Ahmed Abdullah
- Administrator Portal was finalized on December 10th.

December 11th to 12th:

- **USER PORTAL** development began for viewcustomerinfo() function.
- Update password feature added on 12th.
- Imtiaz Ali and Ali Nadir

December 13th to 14th:

- Addbookings() function drafted and then finalized on 14th
- Testing implemented on 14th
- Ali Nadir and Ahmed Abdullah

December 14th to 15th:

- Program fully functional at this stage with zero bugs or errors.
- Redundancies in bookings removed.
- Console I/O implemented and console menus updated
- Ali Nadir and Imtiaz Ali

December 17th to 18th:

- Module viewbookings() added to **USER**.
- Ali Nadir

December 19th to 20th:

- **Feedback** feature added to **USER**.
- Further testing and correction implemented
- Ali Nadir, Ahmed Abdullah

December 27th 2020 to Jan 5th 2021:

- **Revision_of addbookings()**
- Rows divided into Aisle and Window types
- All rows divided into Economy, Business, and First class
- Additional loops added to ensure booking integrity

Jan 11th to Jan 18th:

- **Flight Rankings/Ratings_implementation** decided
- Addition of adminrank() function to **ADMIN**.
- Initialized ranking of every new flight entered through **addflight()** module
- Added flight rank functionality to **addbookings()**
- Ali Nadir, Ahmed Abdullah, Imtiaz Ali
- **Project finalized**

Results:

- Added feedback portal
- Added complementary upgrade from economy class to business class
- Fully developed reservation system using C language
- Added administration features
- Made program more robust through vigorous testing

Conclusion:

Our flight reservation system combines complete customer interaction with an organized, diversified, and detailed flight booking solution. The system is capable of storing and manipulating a large number of customer records with maximum possible efficiency while maintaining simplicity and accessibility in terms of console I/O. The Operating System compatible with our project is Windows, though some additional functions will allow it to run on linux machines as well. Through **USER** module, program allows a single customer to make a maximum of 6 bookings (developer limitation), store any number of tickets for each booking created, retrieve ratings for any booking a use may have created, and optionally enter feedback for the former as well. All active routes as well as flights can be added and managed through **ADMIN** portal, as well as ticket pricing and rank management as well. Makes extensive use of binary file I/O along with structures.