DETAILED DESIGN

Project Code:	FRS-01
Project Name:	Flight Reservation System

Revision History

Version (x.yy)	Date of Revision	Description of Change	Reason for Change	Affected Sections	Approved By
1.0	16/04/2013	Initial Draft			
1.1	10/05/2013	Revision			
2.10	Aug 2013	Revision			
2.20	Sep 2013	Revision	Mapping with CPC		
2.3	Dec 2013	Revision	Mapping with UCF		

Affected Groups

Development Engineering
Quality Assurance
XYZ Air Travels Ltd.

List of Reference Documents

Name	Version No.
1.RS_FRS	2.20
2.FS_FRS	2.20
3.	
4.	

Prepared by/Date	Reviewed by/Date	Approved by/Date

Table of Contents

1. Introduction

1.1 Background

XYZ Air Travels Ltd provides air travel services to users (travelers) across the globe.

1.2 Purpose

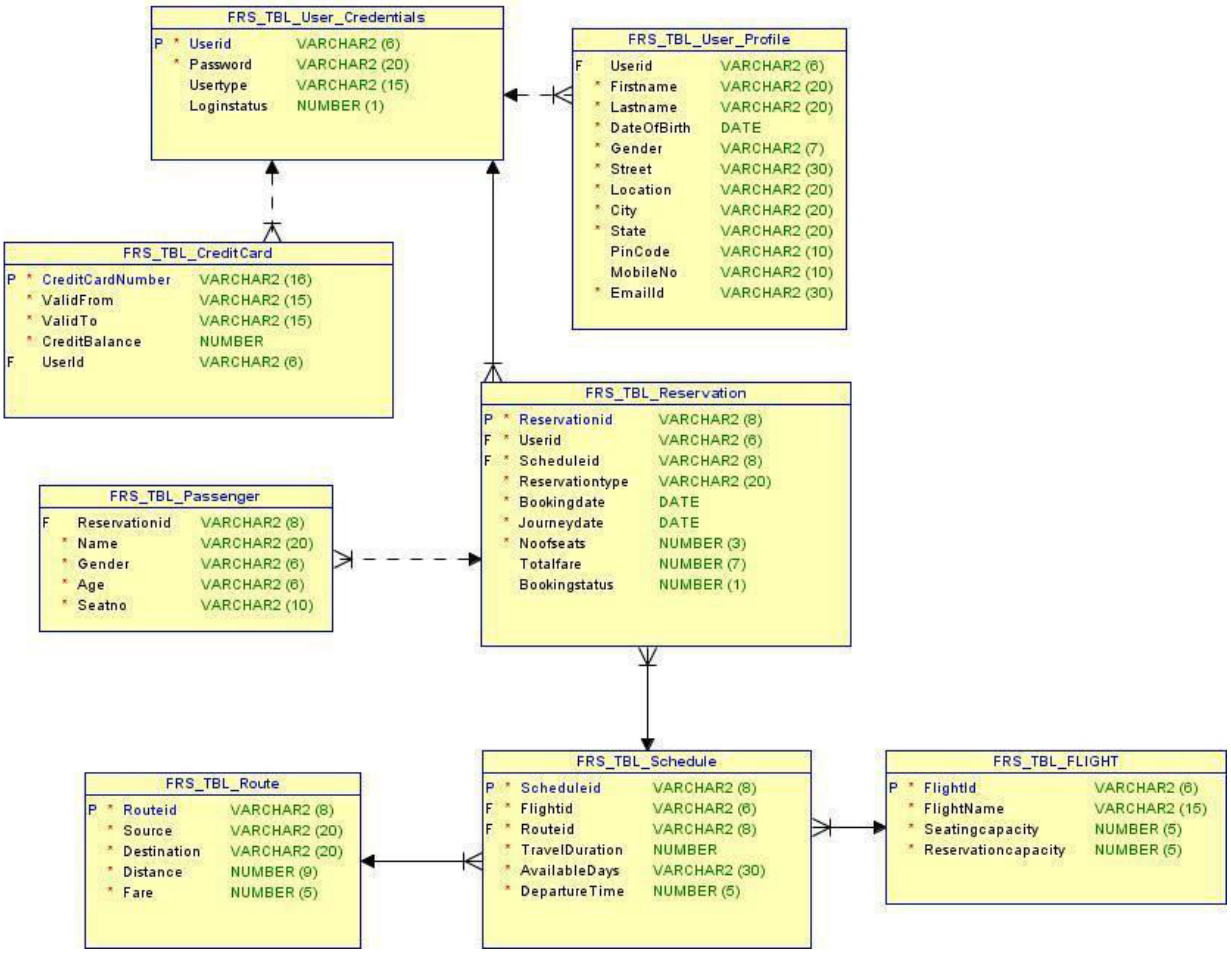
XYZ Air Travels Ltd plans to develop “Flight Reservation System” - a web application where users (travelers) can reserve Domestic/International Flight tickets and manage their reservations.

1.3 Scope

The scope of the Flight Reservation System (FRS) will be to provide the functionality as described in Functional Requirements document. The system will be developed on a Windows XP/Windows 7 machine using Java/J2EE technology and Oracle.

2. Global Data Structures and Shared Data Functions

This section describes the structure of 8 tables to be used for the implementation of requirements as stated in the specification. Refer Appendix for detailed table structure.



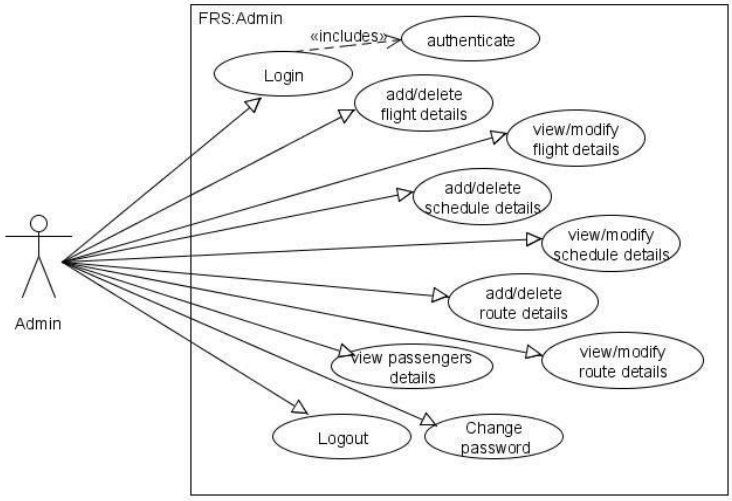
3. High Level Design

This section describes the high level design diagrams Use case diagram with Use Case definition, Sequence Diagram and Class Diagram which provides a visual representation of the requirements , logical flow and their class representations.

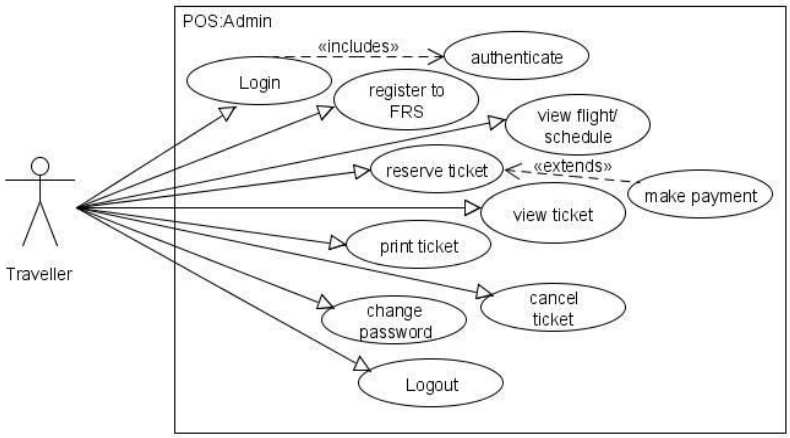
3.1. *Use Case Diagrams*

The requirements of a system can be represented using a use case model in the Use Case Diagram. The use case diagrams for the actors of this case study are given as below.

3.1.1 Use Case Diagram for Administrator



3.1.2 Use Case Diagram for Customer



3.2. *Use Case Definition*

Generally, in a design document, Use case definitions should be written for all the *Requirements* of the system. Below shown Use Case Definition "AD-002 Delete Flight" is the use-case definition for the requirement "AD-002".

3.2.1 Delete Flight Details

Admin performing delete operation on Flight Details.

USE CASE #	AD-002	
Goal	The admin should be able to delete Flight details	
Preconditions	*Flight details should be present in the database and admin should be logged in.*	
Success End Condition	*Flight details should be deleted from the database and concerned reservations should be cancelled. Proper message is*	
Failed End Condition	*No change in Flight details in the database. Proper message is shown.*	
Primary, Secondary Actors	*Primary: Administrator*	
Trigger	*Admin Initiative*	
DESCRIPTION	**Step**	**Action**
	1	*Click on 'View Flight Details' link.*
	2	*Enter FlightID to view a specific Flight detail or View all Flight details. Flight details with delete/modify link are*
	3	*Click on delete link. Confirmation message is shown.*
	4	*Click on OK to delete Flight detail*
	Step	**Branching Action**
	1	*Enter invalid FlightID.*
	2	*Select Cancel on confirmation. Flight details are not deleted.*
Related Information/Use cases	*(AD-001) Add Flight Details*	
Priority	*P1*	
Performance	*2sec (excluding user input)*	
Frequency	*Once in a week*	
Assumptions	*AD-001 is implemented and Flight details exist in DB and Admin could login successfully.*	

3.2.2 Add Flight Schedule Details

Customer performing Add Flight Schedule operation

USE CASE #	*AD-005*	
Goal	The admin should be able to add Flight schedule details	
Preconditions	*Admin should be successfully logged in.*	
Success End Condition	*A new Flight Schedule is added in the database and ScheduleID should be auto-generated.*	
Failed End Condition	*Failure in adding Flight Schedule information.*	
Primary, Secondary Actors	*Primary-Administrator*	
Trigger	*Admin Initiative*	
DESCRIPTION	**Step**	**Action**
	1	*Click on 'Add Flight Schedule' link*
	2	*Enter valid Flight, Route and Schedule information.*
	3	*Click on Add Schedule link*
	Step	**Branching Action**
	1	*Enter invalid/incomplete Schedule information*
	2	*Click on Add Schedule link*
Related Information/Use cases		
Priority	*P1*	
Performance	*1sec (excluding user interaction)*	
Frequency	*1/ Quarterly*	
Assumptions	*Flight and Route details exist in the DB*	

3.3. *Class Diagram*

The class diagram is a very basic concept in object-oriented world. Class diagrams demonstrate a model, describing what attributes and behavior it has rather than describing the methods for accomplishing operations. Class diagrams are very useful in representing relationships between classes and interfaces.

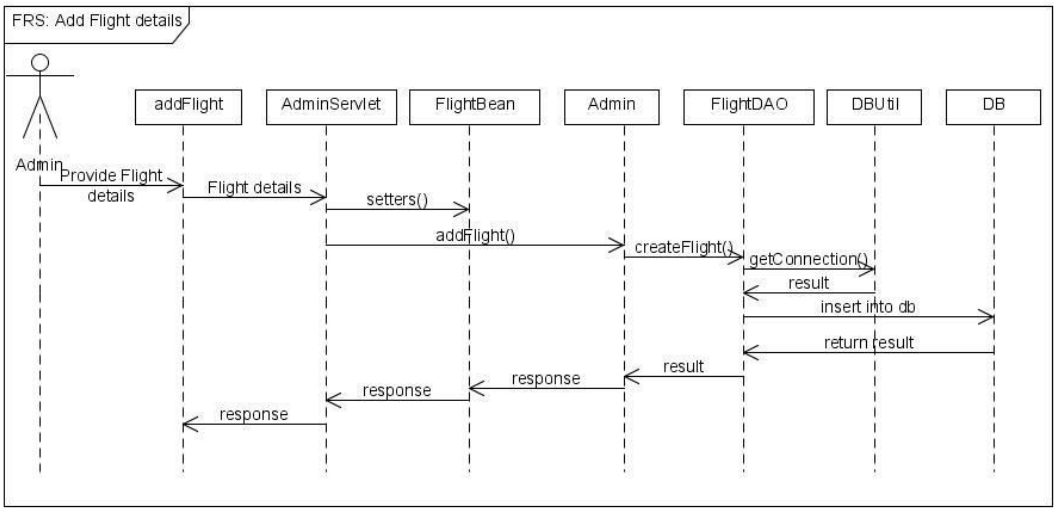
<to be designed by the participant……>

3.4. *Sequence Diagram*

A graphical representation of a module's function invoking functions of other modules in order to achieve a task (specific user requirement) is called a sequence diagram. A sequence diagram for the authentication process is given below for reference. The below example is for a Web Application using servlets/jsp.

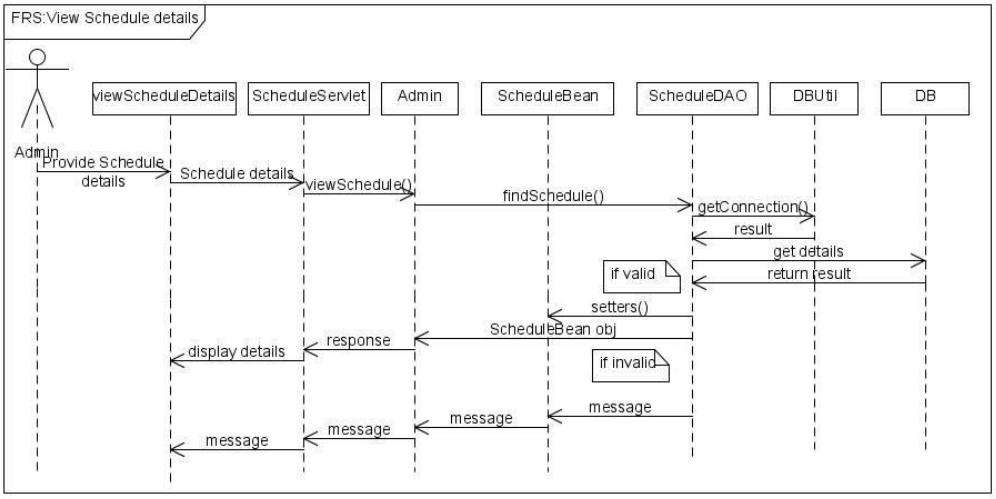
3.4.1 Add Flight details

Admin performing add Flight details



3.4.2 View Schedule Details

Customer performing view Schedule



3.5. *Packages / Interface / Classes*

This section provides a brief outlook on the packaging hierarchy along with the respective classes to be used for the implementation.

The 4 packages mentioned below are for both standalone and Web Application.

Packages	
Package	**Description**
com.lnt.frs.service	This package contains all the Service classes
com.lnt.frs.bean	This package contains all the Bean classes
com.lnt.frs.dao	This package contains all the DAO functionality classes
com.lnt.frs.util	This package contains all the generic functionality classes

This package is used only for a standalone application.

com.lnt.frs.ui	This package contains all the UI related classes [For Core Java]

The package for the controller class should be used as below based on the type of application

com.lnt.frs.listener	**listener -** core java
or	
com.lnt.frs.servlet	**servlet -** Web Applications
or	
com.wirpo.frs.action	**action -** Struts
or	
com.lnt.frs.controller	**controller -** Spring

Package com.lnt.frs.bean

Class Name	Attributes	Data Type
ProfileBean	userID	String
	firstName	String
	lastName	String
	dateOfBirth	Date
	gender	String
	street	String
	location	String
	city	String
	state	String
	pincode	String
	mobileNo	String
	emailID	String
	password	String
CredentialsBean	userID	String
	password	String
	userType	String
	loginStatus	int
FlightBean	flightID	String
	flightName	String
	seatingCapacity	int
	reservationCapacity	int
RouteBean	routeID	String
	source	String
	destination	String
	distance	int
	fare	double
ScheduleBean	scheduleID	String
	flightID	String
	routeID	String
	travelDuration	int

	availableDays	String
	departureTime	String

ReservationBean	reservationID	String
	userID	String
	scheduleID	String
	reservationType	String
	bookingDate	Date
	journeyDate	Date
	noOfSeats	int
	totalFare	double
	bookingStatus	int
PassengerBean	reservationID	String
	name	String
	gender	String
	age	int
	seatNo	int

Package com.lnt.frs.service

Interface Summary	
Interface	**Description**
Administrator	Entity interface for Administrator dealing with the admin process functionalities
	Methods String **addFlight**(FlightBean flightBean) Return value must be either: "SUCCESS", "FAIL", “ERROR” boolean **modifyFlight**(FlightBean flightBean) int **removeFlight**(ArrayList<String> flightID) String **addSchedule**(ScheduleBean scheduleBean) Return value must be either: "SUCCESS", "FAIL", “ERROR” boolean **modifySchedule**(ScheduleBean scheduleBean) int **removeSchedule**(ArrayList<String> scheduleId) String **addRoute**(RouteBean routeBean) Return value must be either: "SUCCESS", "FAIL", “ERROR” boolean **modifyRoute**(RouteBean routeBean) int **removeRoute**(ArrayList<String> routeId)

	FlightBean **viewByFlightId**(String flightId) RouteBean **viewByRouteId**(String routeId) ArrayList<FlightBean> **viewByAllFlights**() ArrayList<RouteBean> **viewByAllRoute**() ArrayList<ScheduleBean> **viewByAllSchedule**() ScheduleBean **viewByScheduleId**(String scheduleId) ArrayList<PassengerBean> **viewPassengersByFlight**(String scheduleId)
Customer	Entity interface of customer for dealing with the customer process functionalities
	Methods ArrayList<ScheduleBean> viewScheduleByRoute(String source, String destination, Date date) String reserveTicket(ReservationBean reservationBean, ArrayList<PassengerBean> passengers) boolean cancelTicket(String reservationId) Map<ReservationBean,PassengerBean> viewTicket(String reservationId) Map<ReservationBean,PassengerBean> printTicket(String reservationId)

Package com.lnt.frs.dao

Find below the suggestive approach for CRUD operations [method naming & signature] for DAO Interface/classes. Create the necessary DAO classes.

Interface Name	Description
xyzDAO	DAO interface/class to deal with operations related to the specific table. **Method Summary** String createXYZ(BeanObject) int deleteXYZ(ArrayList<String>) boolean updateXYZ(BeanObject) BeanObject findByID(String) ArrayList<BeanObject> findAll()

- If required, additional find methods can be created.

Package com.lnt.frs.util

Interface Summary	
Interface	**Description**
Authentication	This interface is responsible for performing the Authentication and Authorization process.
	Methods boolean **authenticate**(CredentialsBean credentialsBean) String **authorize**(String userId) boolean **changeLoginStatus**(CredentialsBean credentialsBean, int loginStatus)
User	Interface for handling different types of users
	Methods String **login**(CredentialsBean credentialsBean) Return value must be either: "A", "C", "FAIL", “INVALID” A->Admin, C->Customer **Wrong username/password should return INVALID.** boolean **logout**(String userId) String **changePassword**(CredentialsBean credentialsBean, String newPassword) Return value must be either: "SUCCESS", "FAIL", “INVALID” String **register**(ProfileBean profileBean) Return value must be either: <userId of length 6>, "FAIL" Note: userId-> first 2 letter of first name followed by 4 digit auto generated number
Payment	Interface for handling payment related information String creditCardNumber, validFrom, validTo, int balance
	Methods boolean findByCardNumber(String userid, String cardnumber) String **process**(Payment payment)
DBUtil	This interface is responsible for establishing Database connections.

Method Summary	
static Connection	getDBConnection(String driverType)

3.6. *UI Templates*

3.6.1 UI Principle

The UI [Presentation Layer] should be designed with the below mentioned principles which helps easy interaction by the user to the application.

3.6.2 UI controls and Usage Principle

The below table provides information of the UI Controls. Based on the User Interface type, the corresponding controls can be used to receive the user input.

UI Type	Controls	Description
Direct Entry	Text Box, Text Area	Any input that cannot be predicted and needs the user to key in. e.g. Name, Address, contact no etc.
Static Selection	Option Button, Check Box, Drop Down	Should be used where the input can be predefined. e.g. gender, month [Jan – Dec] etc. If number of items is more, drop down is preferred.
Dynamic Selection	Drop Down	The items for the drop down should be retrieved from a stored data. e.g. Displaying Branch Id in a drop down from branch table.
Automation	Label Text Field [Read Only]	Data that are calculative or an output of a function. e.g. : Displaying system date, showing total amount etc.
Decision Control	Button	Operations like submit, save, clear should be executed only upon clicking respective buttons.

3.6.3 UI Template

This section contains the design template for the website home page [Fig. 1] that will be displayed at the time of opening this web application and Actor specific home page [Fig. 2].

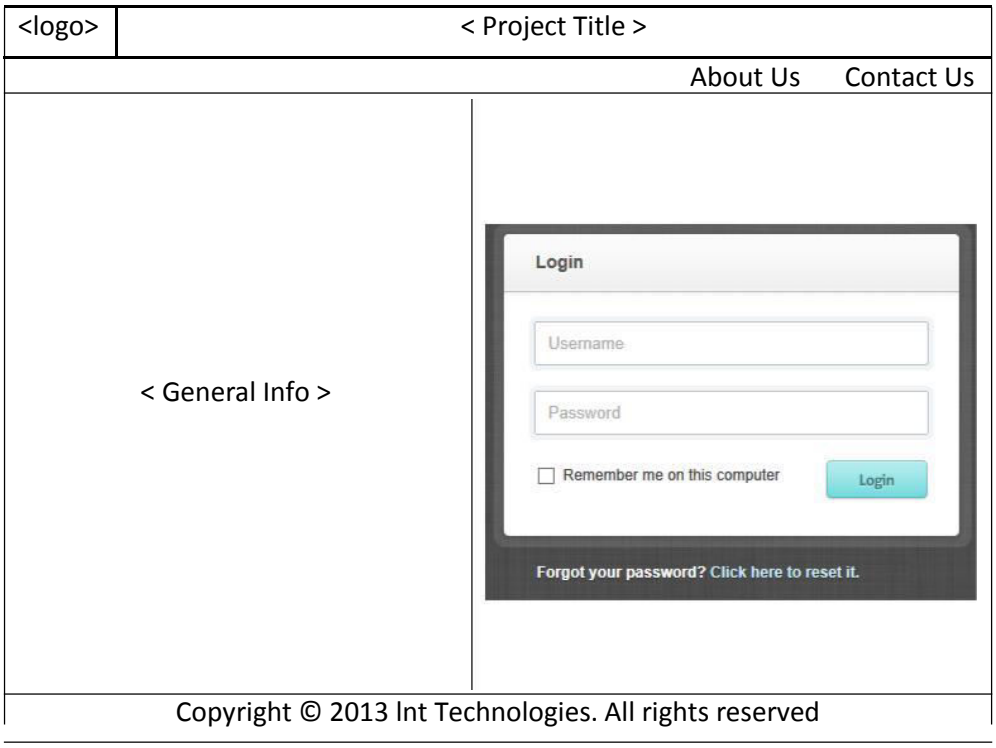


Fig. 1 - Main Page [First Page to open]

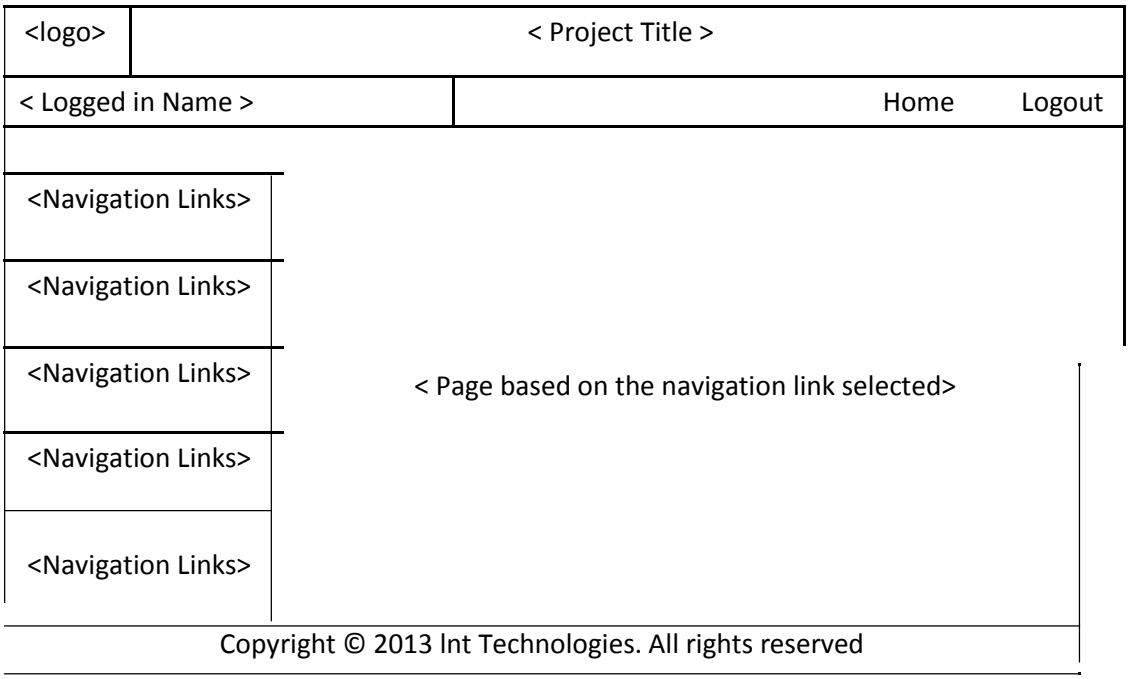


Fig. 2 - Home Page for Actor

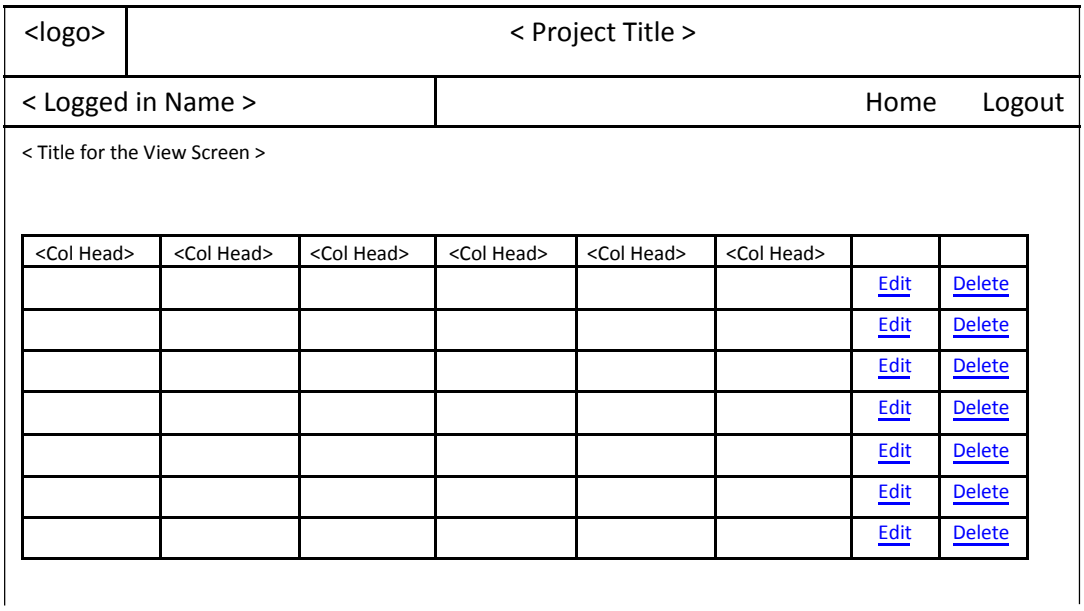


Fig. 3 – View Screen with Edit and Delete Functionality

4. Critical Functions and Focus for Testing

login(), addFlight(), addSchedule(), reservation(), makePayment(), cancelTicket() .

5. Limitations

- Administrator cannot register. Hence, admin's profile is not maintained.
- One way booking is allowed as per current scenario.
- Payment gateway is not implemented.
- Waiting list for passengers is not implemented.
- Customer's credit card details have to be already entered in the database.
- The number of administrators is limited to 2

6. APPENDIX

1. Table: FRS_TBL_User_Profile

This table contains User specific details entered during User Registration.

Field Name	Data Type	Constraints
Userid	VARCHAR2(6)	Foreign Key
Firstname	VARCHAR2(20)	Not Null
Lastname	VARCHAR2(20)	Not Null
DateOfBirth	DATE	Not Null
Gender	VARCHAR2(7)	Not Null
Street	VARCHAR(30)	Not Null
Location	VARCHAR2(20)	Not Null
City	VARCHAR2(20)	Not Null
State	VARCHAR2(20)	Not Null
PinCode	VARCHAR2(10)	
MobileNo	VARCHAR(10)	Exact 10 digit only
EmailId	VARCHAR2(30)	Not Null

2. Table: FRS_TBL_User_Credentials

This table contains Authentication Information for Administrator, Customer

Field Name	Data Type	Constraints
Userid	VARCHAR2(6)	Auto-Generated ,Primary Key*
Password	VARCHAR2(20)	Not Null
Usertype	VARCHAR2(1)	Either ['A','C']
Loginstatus	NUMBER(1)	Either[1 ,0]

* userid should be first 2 letters of First Name followed by 4 digits auto generated number

3. Table: FRS_TBL_Flight

This table contains Flight specific information added by the Admin.

Field Name	Data Type	Constraints
FlightId	VARCHAR(6)	Primary Key*
FlightName	VARCHAR(15)	Not Null
Seatingcapacity	NUMBER(5)	Not Null
Reservationcapacity	NUMBER(5)	Not Null

* FlightId should be first 2 letter Flight Name followed by 4 digit auto generated number

4. Table: FRS_TBL_Route

This table contains Route details as defined below.

Field Name	Data Type	Constraints
Routeid	VARCHAR(8)	Primary Key
Source	VARCHAR(20)	Not Null
Destination	VARCHAE(20)	Not Null
Distance	NUMBER	Not Null
Fare	NUMBER	Not Null

*RouteId should be first 2 letters of source name followed by 2 letters of destination and auto generated 4 digits; Distance would be in miles; Fare is defined as cost per mile.

5. Table: FRS_TBL_Schedule

This table contains Flight Schedule details as provided by the admin.

Field Name	Data Type	Constraints
Scheduleid	VARCHAR(8)	Primary Key
Flightid	VARCHAR(6)	Foreign Key, Not Null
Routeid	VARCHAR(8)	Foreign Key, Not Null
TravelDuration	NUMBER	Not Null
AvailableDays	VARCHAR(30)	Not Null
DepartureTime	NUMBER(5)	Not Null

* ScheduleId should be first 2 letters of source name followed by 2 letters of destination and auto generated 4 digits
* Available days should be comma separated [e.g Mon,Wed, Fri]

6. Table: FRS_TBL_Reservation

This table contains Reservation details provided by the traveler.

Field Name	Data Type	Constraints
Reservationid	VARCHAR(8)	Primary Key
Userid	VARCHAR(6)	Foreign Key, Not Null
Scheduleid	VARCHAR(8)	Foreign Key, Not Null
Reservationtype	VARCHAR(20)	Not Null
Bookingdate	DATE	Not Null
Journeydate	DATE	Not Null
Noofseats	NUMBER	Not Null
Totalfare	NUMBER	Product of Route Fare ,Distance and no. of seats.
Bookingstatus	NUMBER(1)	Either [1, 0]

* ReservationId should be first 2 letters of source name followed by 2 letters of destination and auto generated 4 digits
* Booking Status -> 1 [confirmed], 0[cancelled]

7. Table: FRS_TBL_Passenger

This table contains Passenger details provided by the traveler during ticket reservation.

Field Name	Data Type	Constraints
Reservationid	VARCHAR(8)	Foreign Key
Name	VARCHAR(20)	Not Null
Gender	VARCHAR(6)	Not Null
Age	VARCHAR(6)	Not Null
Seatno	VARCHAR(10)	Not Null

8. Table: FRS_TBL_CreditCard

This table contains CreditCard details of the traveler for ticket reservation.

Field Name	Data Type	Constraints
CreditCardNumber	VARCHAR(16)	Primary Key
ValidFrom	VARCHAR(7)	Not Null
ValidTo	VARCHAR(7)	Not Null
CreditBalance	NUMBER	Not Null
UserId	VARCHAR(6)	

Database Sequences

Sequence Name	Purpose	Starts with
FRS_SEQ_USER_ID	User ID	1000
FRS_SEQ_ROUTE_ID	Route ID	1000
FRS_SEQ_FLIGHT_ID	Flight ID	1000
FRS_SEQ_SCHEDULE_ID	Schedule ID	1000
FRS_SEQ_RESERVATION_ID	Reservation ID	1000