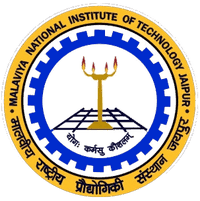
**MALAVIYA NATIONAL INSTITUTE OF TECHNOLOGY, JAIPUR**

****

SOFTWARE REQUIREMENTS SPECIFICATION

DOCUMENT ON

“TOURISM MANAGEMENT SYSTEM”

Submitted by:

TARISHI JAIN (2016UCP1443)

**TABLE OF CONTENTS:**

1. **Introduction**
   1. Purpose
   2. Scope
   3. Definitions, Acronyms and Abbreviations
   4. References
   5. Overview
2. **General description**
   1. Product perspective
   2. Product functions
   3. User characteristics
   4. General constraints
   5. Assumptions and Dependencies
3. **Specific Requirements** 
   1. Functional requirements
   2. Performance requirements
   3. External interface requirements

3.3.1.User Interface

3.3.2.Hardware Interface

3.3.3.Software Interface

3.3.4.Communication Interface

1. **Non Functional Requirements**
2. **Appendices**
3. **Introduction:**

The introduction of the Software Requirements Specification (SRS) provides an overview of the entire SRS with purpose, scope, definitions, acronyms, abbreviations, references and overview of the SRS. The aim of this document is to gather and analyze and give an in-depth insight of the complete Tourism Management software system by defining the problem statement in detail.

* 1. **Purpose:**
* The purpose of this SRS document is to provide a detailed overview of our software product, its parameters and goals.
* This document describes the project's target audience and its user interface, hardware and software requirements.
* It defines how our users and admin see the product and its functionality. It can also be used for future reference of our project to review from where the project was started and what all facilities it does provide.
  1. **Scope:**
* The Tourism Management System manages to give all the information about the tourist places, the attractions over there, accommodation facilities provided to the tourist, stay details and all the travel expenses in the form of packages.
* User can view the transport facilities available to reach to the destination place via flight, travelling either through Business class or Economy class.
* TMS provides admin the facility to add different new packages and/or flights.The updations will be visible to the user, thus is highly interactive.
* Proper error messages are shown ,if any invalid entries are provided by user in login section or if any of the mandatory marked fields are left blank in any of the forms.
  1. **Definitions, Acronyms and Abbreviations:**

|  |  |
| --- | --- |
| **Term** | **Definition/Acronym/Abbreviation** |
| SRS | Software Requirement Specification |
| Admin | System administrator |
| TMS | Tourism Management System |
| CSS | Cascading Style Sheet |
| S/W | Software |
| MoT | Mode of Transport |
| HTML | Hyper Text Markup Language |
| DFD | Data Flow Diagram |
| UML | Unified Modeling Language |
| ER | Entity Relationship |
| H/W | Hardware |
| RAM | Random Access Memory |

* 1. **References:**

[1]. SlideShare

[2]. Stackoverflow,quora

[3]. Lucid chart

[4].Information from -yatra.com

* 1. **Overview:**

The remaining sections of this document provide a general description, including characteristics of the users of this product, the product's hardware, ,functional and data requirements of the product.  General description of the project is discussed in section 2 of this document.

Section 3 gives the functional requirements, data requirements, constraints and assumptions made while designing the TMS.  It also gives the user viewpoint of product and description of non functional requirements.

Section 4 shows DFD , Data Dictionary ,UML and ER diagrams, algorithms and testings performed.

1. **General description:**

This section provides the detail functions of TMS with user characteristics permitted constraints, assumptions and dependencies and requirements subsets.

* 1. **Product perspective:**

The Tourism Management System consist of all the details a tourist wants about packages, their detailed information, travel and stay expenses, site attractions and transports available to reach the place. It maintains database storing details of all the places and different packages of that place available.

* 1. **Product function:**

The Tourism Management System will primarily perform following functions:

* Registration of new user. Confirming it via link sent by TMS.
* View Packages Available
* View flights available from anywhere in the world-**Global Tourism**
* View total price after giving date of tour and number of people accompanying
* Manage booking details- book via
* Credit card
* Debit card.
* Confirmation email showing details like:
* Tourist name
* Destination
* No of days
* Tour start date
* Total amount, etc.
  1. **User characteristics:**
* **Tourist:**
* A new user can register in TMS and confirm his registration by clicking on the mailed link, redirecting him to the home page of TMS.
* User can change his password too, in case he want more privacy and security.
* He can view all the packages and the detailed day wise information , provided in the package.
* Can view the flights available for the source and destination entered.
* Total amount is displayed to him at the end after which he can confirm his booking.
* **Admin:**
* Admin can login to TMS.
* He can add a new flight either for the existing source and destination or for a completely new one.
* Adds Economy and Business class amount and flight timings.
* Adds new package of same or available destination
* Adds images, sightseeings, hotels, its popularity and daywise tour visit details to the package.
  1. **General constraints:**
* Dummy card details are used for banking since there will not be any real transactions.
* Application is based on HTML, CSS, JavaScript, PHP only.
* The required database and XAMPP server should be there for its proper functioning.
  1. **Assumptions and dependencies:**
* It is assumed that the user should have some basic knowledge of computer.
* Admin will have with him, proper day-wise knowledge of whichever new package he will add.
* Admin will make all the hotel and tour arrangement facilities prior to the tour.

1. **Specific Requirements:**

This block describes all the requirements and features of the system.

* 1. **Functional requirements:**
* **Registration by user/tourist:**
* New user should be able to register in TMS.
* He should get a Registration confirmation link on email and only after clicking on it ,his details like name,email-id,password will be saved in database.
* **User Login:**
* The user will enter his details which are matched from that in the database,in case of invalid entries ,system should show error message , else redirect to the home page.
* Change password option is also available.
* **View flight:**
* User can view flights available from his specified source to destination.
* Each available flight’s name,timing,Economy and Business class amount is displayed.
* User should book the one he desire to choose.

* **View packages:**
* Packages list and their starting amount are displayed based on the destination place chosen.
* Packages vary in price according to the no. of days of stay, hotel, no. of sightseeings etc.
* Detailed day-wise information is shown of the chosen package.
* No of people and tour start date are entered by user to view the total amount.
* **Payment Management**
* Two separate payments for booking a package or booking flight (MoT).
* Sending a email on the registered mail id for each payment , showing final booking details .
* **Admin Login:**
* Enter valid credentials (name, email, password) to login, showing error message otherwise.
* Redirected to admin home page showing options of either adding a new package or new flight transport.

* **Add Flight:**
* Admin adds flight name, source , destination, timings of flight and amount of Economy and Business classes.
* **Add Package:**
* Admin adds package of new or existing destination.
* Destination place, no of days and nights, travel expenses are provided.
* Each day’s detailed information is entered now comprising:
* Hotel name
* Stars of hotel
* Amount of hotel
* Sightseeing’s on that day
* Image of hotel and place

* 1. **Performance requirements:**

Following features, if kept in mind ,results in better performance of the system.

* Proper internet connectivity should be there for uminterrupted functioning of mailing features.
* Admin should provide the url’s of hotel and place images.
* Mail-id provided during registration should be correct, else registration will fail.
* The timings of flight should be in 24-hour format and use of am/pm should be avoided.
* Admin should provide all the details asked by the TMS while adding package/flight.
* For booking, dummy card details should be entered by the user.
* All the required database existence is must, prior to the use of TMS.
  1. **External interface requirements:**

This section shall describe the interface requirements for the TMS. They specify the way user shall interact with the system as well as defines the necessary H/W and communication interfaces by the S/W.

* + 1. **User interfaces:**
* The user login page pops up asking for name, mail id and password. New user can register as well.
* The user page has two tabs in the navigation bar which allow users to easily switch between the different parts of the program.
* The first tab is the “Holiday” tab which ask user to enter destination place and shows all the available packages of that destination alongwith their starting price.
* On clicking on any of the package, user can view detailed info of packages and the hotel services provided.
* He can proceed to the booking process of any of the paakage and get all the booking details via email after payment.
* The second tab, named “Flights”, prompts user to enter his departure place and destination place ,r edirecting him to the new page showing all the available flights for the specified route.
* User can book either Business or Economy class via clicking on the respective buttons.

* + 1. **Hardware interface:**
* Operating system : Windows or Linux
* Hard disk : 40 GB
* RAM : 1 GB
* Processor used : i5 quad-core CPU
  + 1. **Software interface :**

* Web server :Apache Tomcat Server
* Database: My SQL
* XAMPP server
* Development end: PHP,MySQL,CSS,JavaScript
* Supported Browser: Google Chrome,Internet Explorer,Mozilla Firefox
* IDE used: Sublime Text
* Images are in .jpg/.jpeg/.png format.

**3.3.4 Communication interface:-**

* Windows or Linux
* Internet connectivity.

1. **Non Functional Requirements:**

Following list shows various attributes of TMS:

* **Efficiency:** It uses the resources efficiently to achieve any given task.
* **Portability:** The system should run on any Windows/Linux environment.
* **Correctness:** Application is correct in terms of data supplied and inner calculations done (like in calculating total amount).It adheres to functional requirements.
* **Integrity or Security:** System is secure enough to prevent unauthorized users from viewing the system details. Only admin can add new functionalities and tourist/users are not allowed to register in admin section.
* **Flexibility:** TMS is highly flexible and new features can be added to it with much ease without changing the functionality of existing ones since there are different units (modules) for each functionality.
* **Reliability:** The product is reliable enough to sustain in any condition and gives consistently correct results.
* **Testability:** Unit testing can be done easily which leads to effective system testing.
* **Maintainability:** The system is maintainable and can be upgraded to new features effectively. Any error can also be removed easily by the developer.

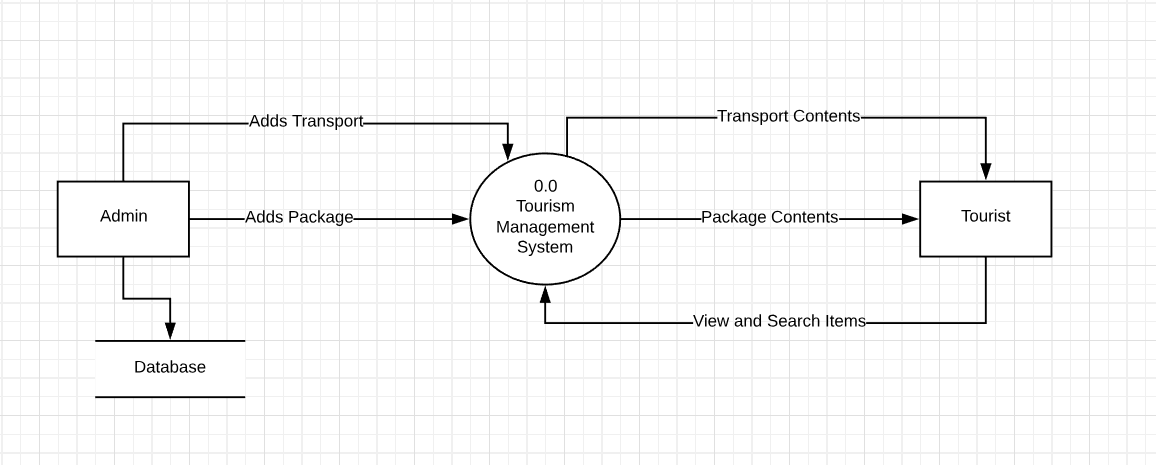
1. **Appendices:**

* **DATA FLOW DIAGRAM:**

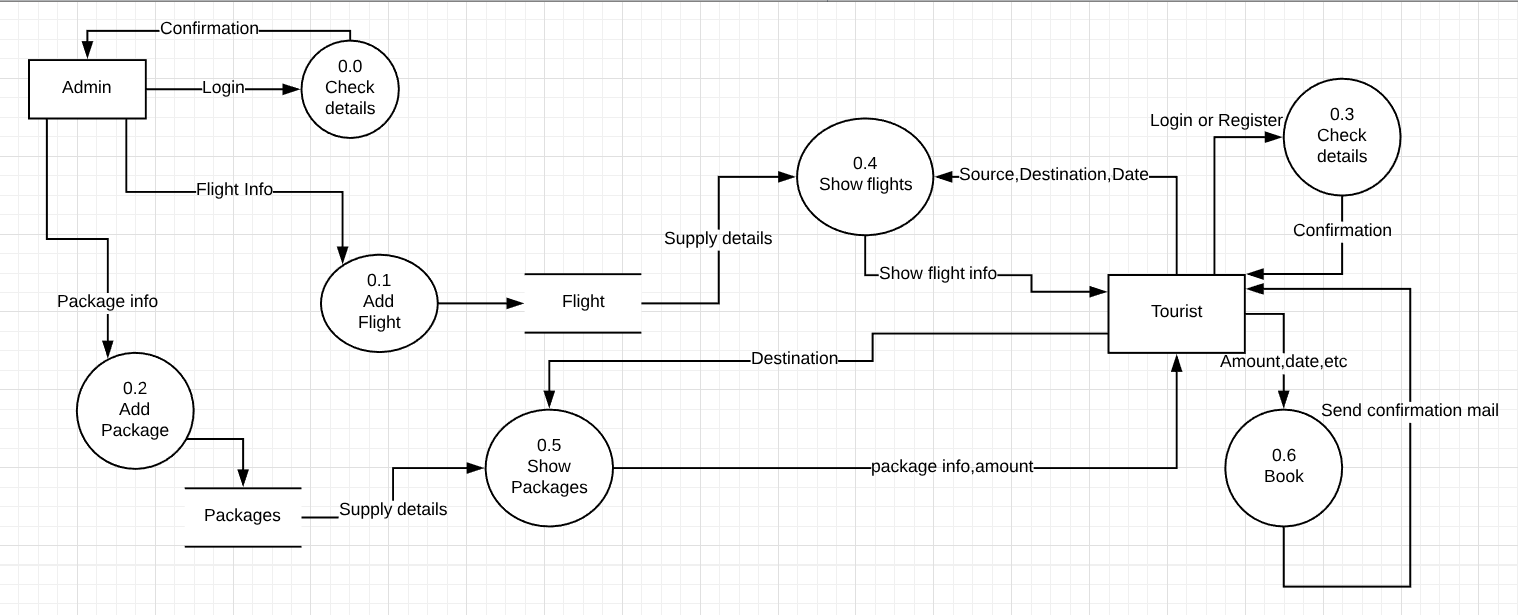
**Data Dictionary:**

|  |  |
| --- | --- |
| **Term** | **Definition/Type** |
| Username | Name of user/  String |
| Password | Password of user or admin/  String |
| Email-id | Valid mail id to send mails/  email |
| Source | Flight departure place/  String |
| Destination | (Flight|Package)  (Flight arrival place or place tourist wants to visit)/  String |
| Card No. | 16 digit card no of user/  Number |
| Date | Date in dd-mm-yyyy format/  Date |
| CVV | 3 digit CVV no of user/  Number |

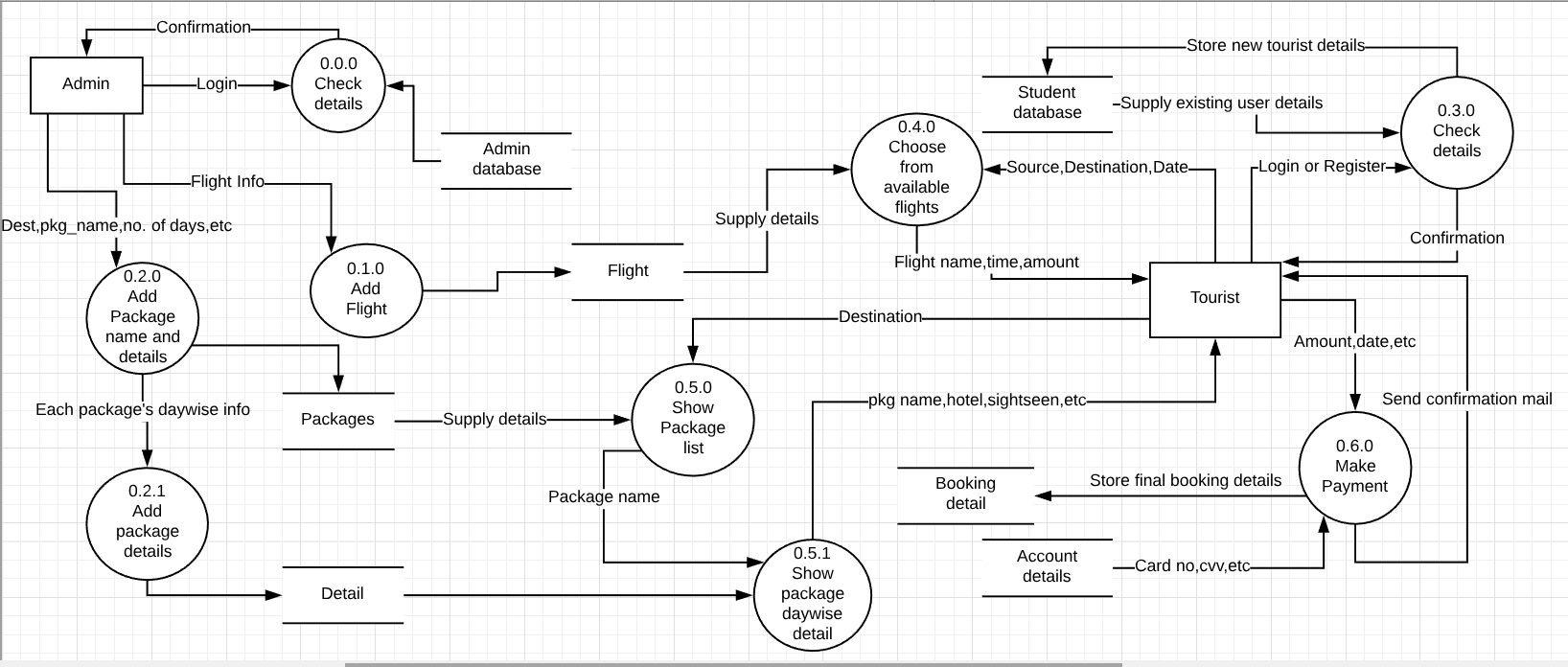
* **Level 0:**



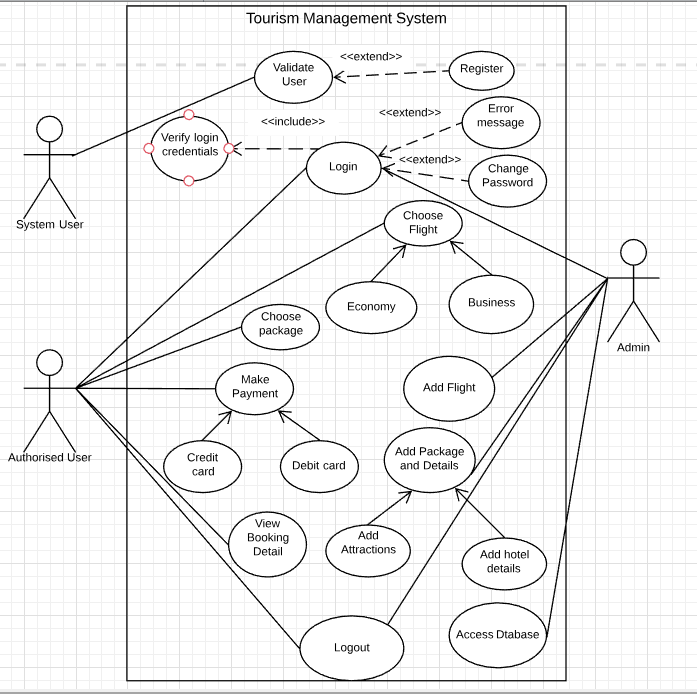
* **Level1:**



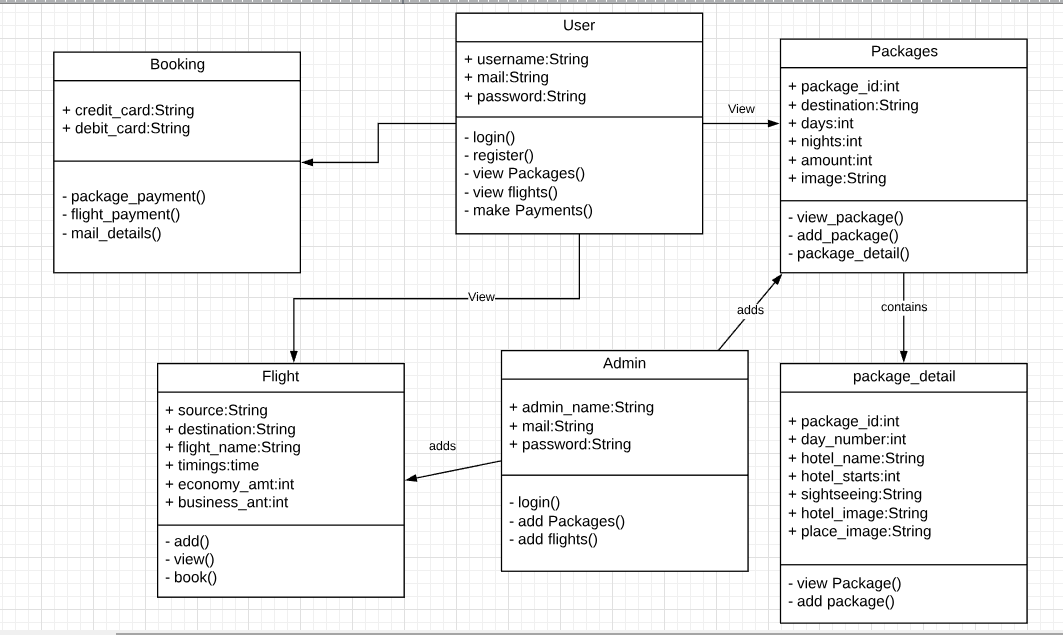
* **Level 2:**



* **UML DAIGRAMS**
* **Use-Case**

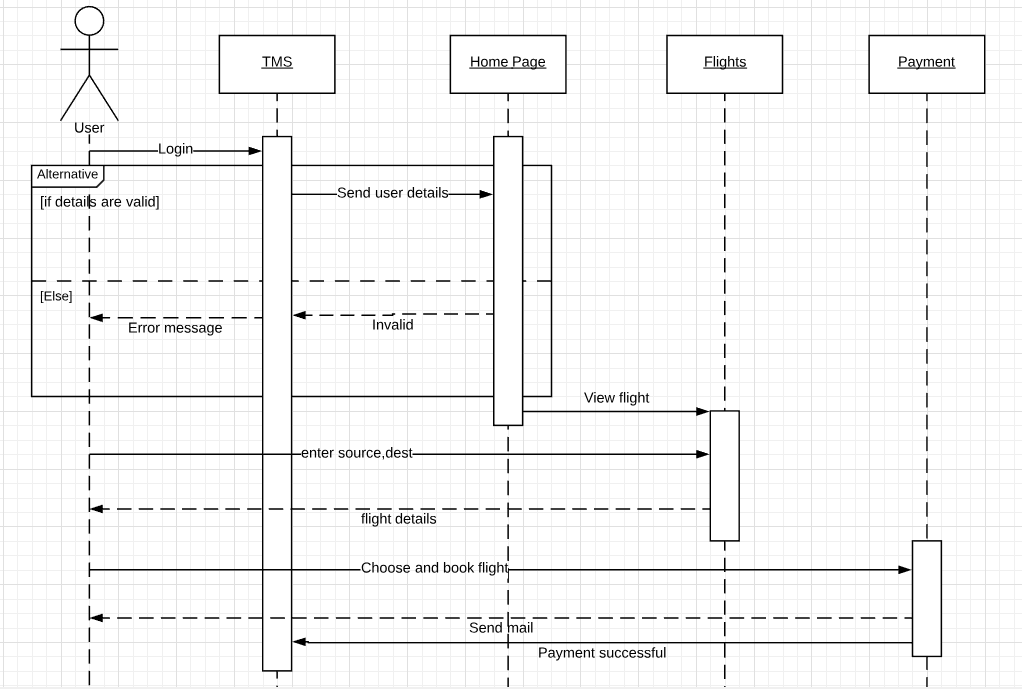


* **Class Diagram**

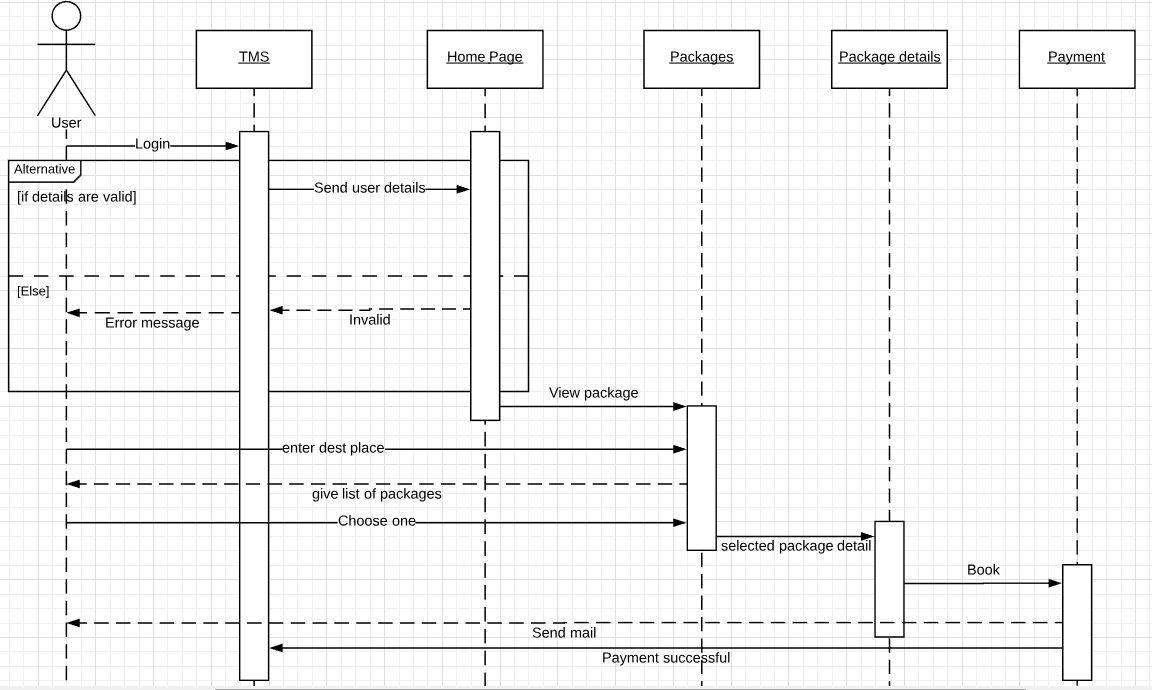


* **Sequence Diagram**

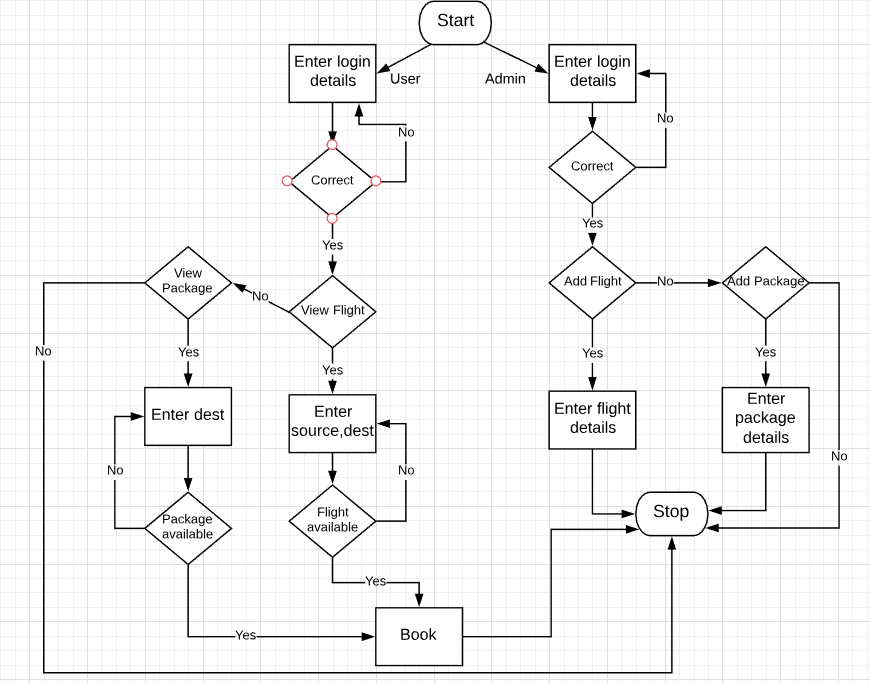
**Showing Flights:**



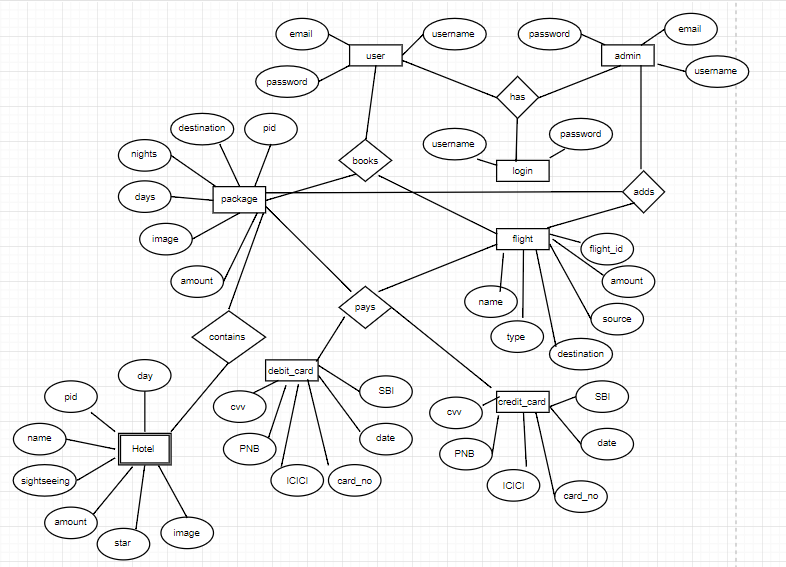
**Showing packages:**



* **Activity Diagram**

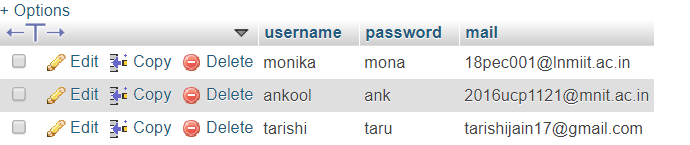


* **ER DIAGRAM:**

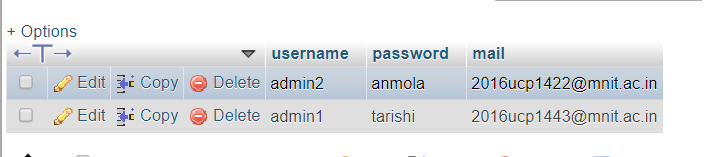
****

1. **Database:**

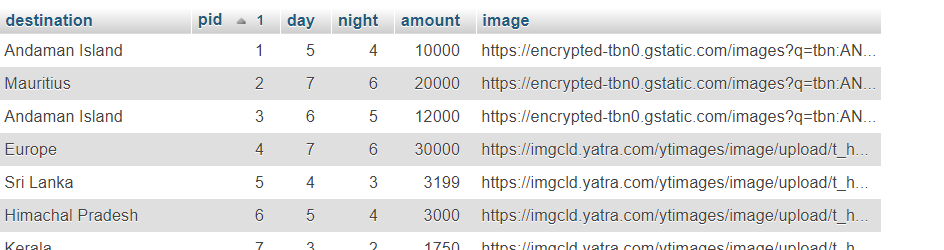
* **User:**

****

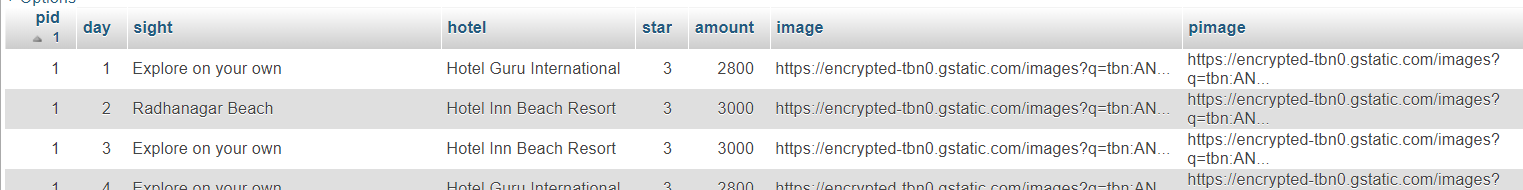
* **Admin**

****

* **Package**



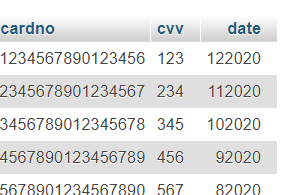
* **Package\_detail**



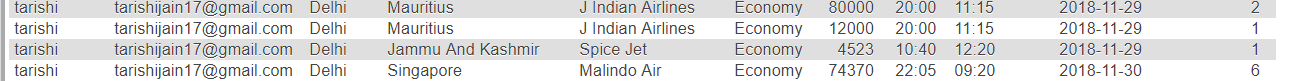
* **Flight**



* **Card Details**



* **Flight booking detail**



* **Package Booking detail**



1. **Algorithms**
   1. **User Login**

user\_login ( ) {

user\_name = input.name;

user\_pwd = input.pwd;

user\_mail = input.mail;

if ( user\_name == null || user\_pwd == null || user\_mail == null){

Output “Marked Fields mandatory”

}

else {

row = the entry having same ‘e-mail’ as user\_mail from ‘log’ relation.

if ( row.name == user\_name && row.pwd == user\_pwd ){

Output “Login successful”

Redirect to user home page

}

else {

Output “Invalid entries”

}

}

}

* 1. **Admin Login**

admin\_login ( ) {

admin\_name = input.name;

admin\_pwd = input.pwd;

admin\_mail = input.mail;

if ( admin\_name == null || admin\_pwd == null || admin\_mail == null){

Output “Marked Fields mandatory”

}

else {

row = the entry having same ‘e-mail’ as admin\_mail from ‘admin’ relation.

if ( row.name == admin\_name && row.pwd == admin\_pwd ){

Output “Login successful”

Redirect to admin home page

}

else {

Output “Invalid entries”

}

}

}

* 1. **User Registration**

user\_register ( ) {

user\_name = input.name;

user\_pwd = input.pwd;

user\_retype\_pwd=input.repwd;

user\_mail = input.mail;

if ( user\_name == null || user\_pwd == null || ser\_retype\_pwd == null || user\_mail == null) {

Output “Marked Fields mandatory”

}

else if ( user\_pwd != user\_retype\_pwd ) {

Output “Password didn’t match”

}

else {

Send a confirmation activation link to the mail id.

Store the details into log relation.

Redirect to user home page

}

}

* 1. **Change Password**

user\_pwd\_change ( ) {

name = user\_name;

mail = user\_mail;

old\_pwd = input.oldpwd;

new\_pwd=input.new\_pwd;

if ( old\_pwd == null || new\_pwd == null ) {

Output “Marked Fields mandatory”

}

Else {

row = the entry having same email as ‘mail’ in ‘log’ relation

if (old\_pwd == row[pwd]){

update ( row[pwd] = new\_pwd) in ‘log’ relation;

Output “Password changed successfully”’;

}

Else{

Output “Old password didn’t match;

}

}

* 1. **View Flights**

view\_flight ( ) {

source = input.src;

dest = input.dest;

if ( source == null || dest == null){

Output “Enter valid entries”

}

else {

row = Get list and details of flights from ‘source’ to ‘dest’ from ‘flight’ relation.

if ( row.length == 0){

Output “No flight found for this route”

}

else {

for ( int i=1 till row.length ( ) ){

flight\_name = row[i].flight\_name;

flight\_dep = row[i].flight\_departure;

flight\_arr = row[i].flight\_arrival;

flight\_economy\_amt = row[i].economy;

flight\_business\_amt = row[i].businesss;

Output above details;

}

}

}

}

* 1. **Book Flight**

book\_flight ( ){

source = src;

destination = dest;

name = user\_name;

f\_name = flight\_name;

departure = dep\_time;

arrival = arr\_time;

date = input.date ;

no\_of\_people = input.person;

amount = amt;

type = type; /////Business/Economy

/////// Enter card details

card = input.cardno;

cvv = input.cvv;

expiredate=input.expire;

if ( card.length!=16 || cvv.length != 3){

Output “Enter suitable length”

}

/////User paying money, system sending mail

Else if ( card,cvv,expiredate matches with any of the tuple in relation ‘card’){

Send mail giving above details of booking;

Output “Payment successful”;

}

Else {

Output “Invalid entries”;

}

* 1. **View Packages**

view\_pkg ( ) {

dest = input.dest;

if ( dest == null){

Output “Enter valid entries”;

}

else {

////Showing the list of available packages of the selected destination

row = Get list of packages to the ‘dest’ from ‘holiday’ relation.

for ( int i=1 till row.length ( ) ){

pkg\_id = row[i].pkg\_id;

pkg\_image = row[i].img;

pkg.days = row[i].days;

pkg.nights = row[i].nights;

pkg.amount = row[i].amt;

Output above details ;

}

}

}

* 1. **View Package detail**

view\_pkg\_detail ( ) {

pkg = pkg\_id; //Getting the selected package

row = Details of package with package id = ‘pkg’ from detail relation

///Showing the selected package’s details

for ( int i=1 to pkg.days) {

day\_no = i;

hotel\_name = row[i].hotel;

hotel\_stars = row[i].star;

hotel\_amount = row[i].amt;

hotel\_image = row[i].img;

sightseeing = row[i].sight;

Output above details

}

show\_images ();

}

* 1. **Book Package**

book\_pkg ( ){

destination = dest;

name = user\_name;

days = pkg.days;

nights = pkg.nights;

date = input.date ;

no\_of\_people = input.person;

amount = pkg.amt;

/////// Enter card details

card = inut.cardno;

cvv = input.cvv;

expiredate=input.expire;

if ( card.length!=16 || cvv.length != 3){

Output “Enter suitable length”

}

/////User paying money, system sending mail

Else if ( card,cvv,expiredate matches with any of the tuple in relation ‘card’){

Send mail giving above details of trip booking;

Output “Payment successful”;

}

Else {

Output “Invalid entries”;

}

* 1. **Add Flight**

add\_flight ( ){

source = input.src;

destination = input.dest;

flight\_name = input\_name;

economy\_amt = input.eco;

business\_amt = input.busi;

departure = input.dep ;

arrival = input.arr;

if ( any of the field is empty){

Output “Entry valid entries “;

}

else{

Store the above details in flight relation;

Updations succeeded;

}

**}**

* 1. **Add Packages**

add\_package ( ){

pkg\_id = input.id; //Autogenerated

destination = input.dest;

days = input\_days;

nights = input.nights;

amt = input.amt;

if ( any of the field is empty){

Output “Entry valid entries “;

}

//////// Storing day-wise details

else{

for ( int i=1 to days) {

hotel\_name = input.row[i].hotel;

hotel\_stars = input.row[i].star;

hotel\_amount = input.row[i].amt;

hotel\_image = input.row[i].img;

sightseeing = input.row[i].sight;

places\_images = input.row[i].place\_image;

Store in relation ‘detail’ ;

}

}

1. **Testing**
   1. **Black Box Testing:**

* **Decision Table Testing:**

Decision table testing is a testing technique used to test system behavior for different input combinations. This is a systematic approach where the different input combinations and their corresponding system behavior (Output) are captured in a tabular form.

**Ex1. Decision table for User/Admin Login form**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Conditions | Rule1 | Rule2 | Rule3 | Rule4 |
| Username | T | T | F | F |
| Password | T | F | T | F |
| Output | H | E | E | E |

**T** T: Correct username/password

F: Incorrect username/password

E: Apt error message is shown

H: Redirected to home page

**Ex2. Decision Table for mailing**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | Rule1 | Rule2 | Rule3 | Rule4 |
| Conditions | Package Payment | T | T | F | F |
| Flight Payment | T | F | T | F |
| Actions | Package mail sent | Y | Y | N | N |
| Flight mail sent | Y | N | Y | N |

* **Equivalence Class Partitioning :**

Ex.1

Equivalence class for no. of persons:

|  |  |  |
| --- | --- | --- |
| Invalid | Valid | Invalid |
| <1 | >=1 and <=10 | >10 |

Ex.2

Equivalence class for Amount:

|  |  |
| --- | --- |
| Invalid | Valid |
| <100 | >=100 |

Ex.3

Equivalence class for CVV:

|  |  |  |
| --- | --- | --- |
| Invalid | Valid | Invalid |
| <3 | =3 | >3 |

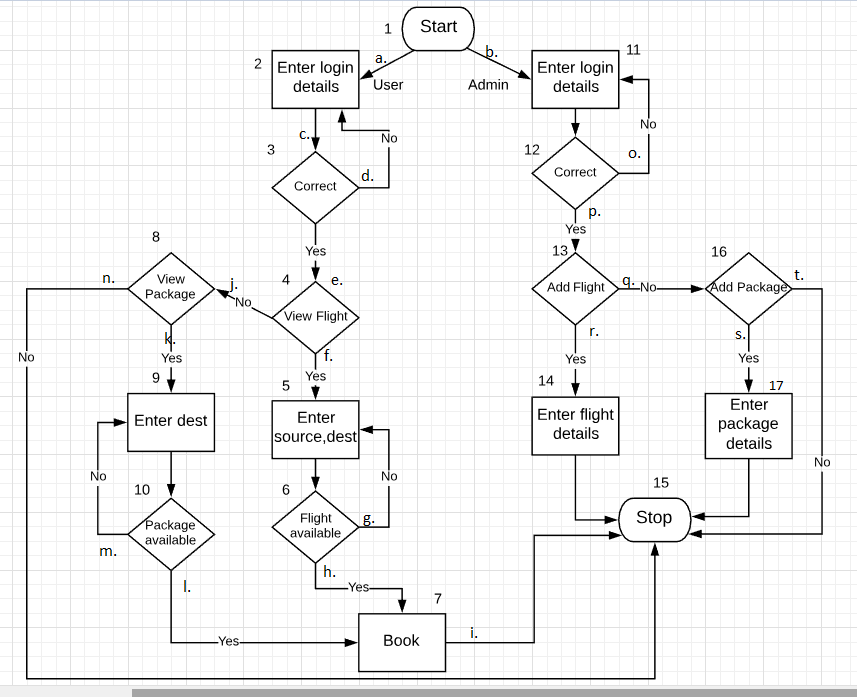
Ex.4

Equivalence class for Card Number:

|  |  |  |
| --- | --- | --- |
| Invalid | Valid | Invalid |
| <16 | =16 | >16 |

**7.2 White Box Testing:**

**Control Flow Graph:**



* **Statement Coverage Testing**

In Statement Coverage [testing](http://www.testingbrain.com/software-testing-tutorial), the code is executed in such a manner that every statement of the application is executed at least once. It helps in assuring that all the statements execute without any side effect.

* **Test Case 1:** User, Invalid Login
* **Test Case 2:** Admin, Invalid Login
* **Test Case 3:** User, Valid Login, View Flight
* **Test Case 4:** User, Valid Login, View Package
* **Test Case 5:** Admin, Valid Login, Add Flight
* **Test Case 6:** Admin, Valid Login, Add Package
* **Decision Coverage Testing**

Decision coverage or Branch coverage is a testing method, which aims to ensure that each one of the possible branch from each decision point is executed at least once and thereby ensuring that all reachable code is executed.

It helps in validating all the branches in the code making sure that no branch leads to abnormal behavior of the application.

|  |  |  |
| --- | --- | --- |
| **Number** | **Test Cases** | **Nodes Traversed** |
| **Test Case 1** | User, Invalid Login | acd |
| **Test Case 2** | Admin, Invalid Login | bo |
| **Test Case 3** | User, Valid Login, View Flight, Flight available | acefghi |
| **Test Case 4** | User, Valid Login, View Package, Not available | acejkmli |
| **Test Case 5** | User, Valid Login, Not willing to view | acejn |
| **Test Case 6** | Admin, Valid Login, Add Flight | bpr |
| **Test Case 7** | Admin, Valid Login, Add Package | bpqs |
| **Test Case 8** | Admin, Valid Login, Not willing to add | bpqt |

* **Basis Path Testing**

Basis path testing involves execution of all possible blocks in a program and achieves maximum path coverage with least number of test cases.

The objective behind basis path testing is that it defines the number of independent paths, thus the number of test cases needed can be defined explicitly .

Cyclomatic Complexity V(G) of flow Graph = No of edges-No of nodes+2

=25-17+2=10

Independent Paths=10

* acdc..
* acefg..
* acefhi
* acejn
* acejkm..
* acejkli
* bo..
* bpr
* bpqt
* bpqs