PROJECT REPORT
ON

TOURISM MANAGEMENT SYSTEM

Submitted in partial fulfillment of requirements to

IT 353 – Summer Internship

By

I. Avinash (Y20IT041)
G. Divya (Y20IT036)
K. Kartheek (Y20IT045)

FEBRUARY 2023

BATCH - 18

R.V.R & J.C.COLLEGE OF ENGINEERING(AUTONOMOUS)
(NAAC A+ GRADE) (Approved by A.I.C.T.E)
(Affiliated to Acharya Nagarjuna University)
Chandramoulipuram : : Chowdavaram
GUNTUR – 522 019

R.V.R & J.C.COLLEGE OF ENGINEERING

DEPARTMENT OF INFORMATION TECHNOLOGY

BONAFIDE CERTIFICATE

This is to certify that this project work titled **TOURISM MANAGEMENT SYSTEM** is the bonafide work of (I. Avinash(Y20IT041), G. Divya(Y20IT036), K. Kartheek(Y20IT045)) who have carried out the work under my supervision, and submitted in partial fulfillment of the requirements to **IT-353, Summer Internship** during the year 2022-2023.

Dr.M.Pompapathi
Lecturer Incharge

Dr. A.Srikrishna
Prof.&HOD, Dept. of IT

ACKNOWLEDGEMENTS

The successful completion of any task would be incomplete without a proper suggestions, guidance and environment. Combination of these three factors acts like backbone to our Project **“TOURISM MANAGEMENT SYSTEM”.**

We are very much thankful to **Dr.Kolla Srinivas**, Principal of R.V.R. & J.C. College of Engineering, Guntur for having allowed delivering this Project - I.

We express our sincere thanks to **Dr.A.Srikrishna**, Professor, Head of the Department of Information Technology for her encouragement and support to carry out this mini project successfully.

We are very glad to express our special thanks to **Dr.M.Pompapathi**, Assistant Professor, in Department of Information Technology for timely, guidance and providing us with most essential materials required for the completion of this report.

Finally we express our sincere thanks to all the **Teaching** and **Non-Teaching staff** of **IT Department** who have contributed for the successful completion of this report.

I. Avinash (Y20IT041)
G. Divya (Y20IT036)
K. Kartheek (Y20IT045)

CONTENTS

1. PROBLEM STATMENT

TOURISM MANAGEMENT SYSTEM

Program Objective:

This software is highly programmed in order to provide best services to customers and various travelling agents in the field of tourisms activities such as bookings, accommodations, tourism spot details. This integrated software offers one of the best ways of managing all the travel related businesses.

Currently:

The user needs to visit travel agency office to plan any tour. It involves a lot of manual paperwork, and customers need to stay in the queue for a long time. The present systems are inadequate in providing information and advice to the agencies and customers about tour plans. Most of the time agencies have to rely on local information sources and on their experience regarding time and cost.

The proposed system is highly automated and makes the travelling activities much easier and flexible. The user can get the very right information at the very right time. Customers can get the knowledge of the hotels and vehicles they are going to use in their trip prior to their starting of trip. The user gets complete information about the various tourist places based on users on their time, cost, source and destination places.

The user can do modifications within three days before of their start of journey, like:

- Regarding the visiting places
- Regarding the number of passengers
- Regarding the hotel residencies

User can also postpone or cancel their trip before a week and charges may apply to their according to their specification.

2. SRS DOCUMENTATION – REQUIREMENTS ELICITATION

ID	DETAILS	FUNCTIONLITIES	PRIORITIES
R1	TMS must be able to store user information	Functional Data	Must have
R2	TMS must be able to respond to user within seconds	Non-Functional performance	Must have
R3	TMS must be able to validate login credentials of user	Functional	Must have
R4	TMS must be able to display basic details of corresponding User when user logs in	Functional	Must have
R5	TMS must be able to redirect to login page after registering in the system	Non-Functional	Must have
R6	TMS must be able to logout of the system	Functional	Must have
R7	TMS must be able to provide user registration form	Functional	Must have
R8	TMS must be able to provide user login form	Functional	Could have
R9	TMS must be able to provide admin login form	Functional	Must have
R10	TMS must be able to store admin login information	Functional	Must have
R11	TMS must be able to respond to admin within 5 seconds	Non-Functional	Must have
R12	TMS must be able to display respective admin basic information after logging in	Functional	Must have
R13	TMS must be able to logout of the system	Functional	Must have
R14	TMS must be able to validate the start date and end date specified by the user	Functional	Must have
R15	TMS must be able to validate the start date and end date specified by the admin	Functional	Must have
R16	TMS must be able to change the password of user when requested	Functional	Must have
R17	TMS must be able to change the password of admin when requested	Functional	Must have
R18	TMS must be able to update profile of user when required	Functional	Must have
R19	TMS must be able to update admin profile when required	Functional	Must have

R20	TMS must be able to provide the details of vehicles to user	Functional	Must have
R21	TMS must be able to provide the details of hotels to user	Functional	Must have
R22	TMS must be able to provide information about tourist places to user	Functional	Must have
R23	TMS must be able to provide information about tourist place to user which request	Functional	Must have
R24	TMS must be able to provide information about journey time of the vehicle	Functional	Must have
R25	TMS must be able to display the cost of ticket	Functional	Must have
R25	TMS must be able to display distance from source to destination	Functional	Must have
R26	TMS must be able to select vehicles chosen by user	Functional	Must have
R27	TMS must be able to delete vehicles chosen by the user	Functional	Must have
R28	TMS must be able to update vehicles chosen by user	Functional	Must have
R28	TMS must be able to select places chosen by user	Functional	Could have
R29	TMS must be able to delete places chosen by the user	Functional	Must have
R30	TMS must be able to update places chosen by user	Functional	Must have
R31	TMS must be able to select hotel chosen by user	Functional	Must have
R32	TMS must be able to delete hotel chosen by the user	Functional	Must have
R33	TMS must be able to update hotel chosen by user	Functional	Must have
R34	TMS must be able to update the details of passengers	Functional	Must have
R35	TMS must be able to display the booking form	Functional	Must have
R36	TMS must be able to store details of the passengers	Functional	Must have
R37	TMS must be able to postpone the user trip	Functional	Must have
R38	TMS must be able to cancel the user trip	Functional	Must have
R39	TMS must be able to provide report of their user trip	Functional	Must have
R40	TMS must be able to generate the invoice of the trip	Functional	Should have

3. SYSTEM REQUIREMENT SPECIFICATION

Software Requirements:

- Operating System : Windows XP
  - Coding Language : HTML, CSS,JAVASCRIPT,
  - Web Server : Flask Server(Python Module)
  - Data Base: MYSQL

Hardware Requirements:

- Personal computer with keyboard and mouse maintained with uninterrupted power supply.
- Processor : Intel® core™ i3
- Installed Memory (RAM) :4 .00 GB
- Hard Disc : 40GB

4. REQUIREMENTS MODELLING

The most important factor for the success of an IS project is whether the software product satisfies its users' requirements. Models constructed from an analysis perspective focuses on determining these requirements. This means Requirement Model includes gathering and documenting facts and requests.

The use case model gives a perspective on many user requirements and models them in terms of what the software system can do for the user. Before the design of software which satisfies user requirements, we must analyze both the logical structure of the problem situation, and also the ways that its logical elements interact with each other. We must also need to verify the way in which different, possibly conflicting, requirements affect each other. Then we must communicate this under standing clearly and unambiguously to those who will design and build the software.

Use-case diagrams graphically represents system behavior (use cases). These diagrams present a high level view of how the system is used as viewed from an outsider's (actor's) perspective.

A use-case diagram may contain all or some of the use cases of a system. A use-case diagram can contain:

- ➢ actors ("things" outside the system)
- ➢ use cases (system boundaries identifying what the system should do)
- ➢ interactions or relationships between actors and use cases in the system including the associations, dependencies, and generalizations.

Use-case diagrams can be used during analysis to capture the system requirements and to understand how the system should work. During the design phase, you can use use-case diagrams to specify the behavior of the system as implemented.

5. IDENTIFICATION OF ACTORS AND USECASES

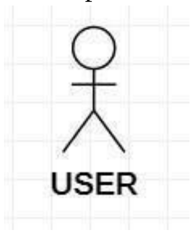
Identification of ACTORS :
Actors represent system users. They are NOT part of the system .They represent anyone or anything that interacts with the system.
An actor is someone or something that:

- Interacts with or uses the system
- Provides input to and receives information from the system
- Is external to the system and has no control over the use cases

Actors are discovered by examining:

- Who directly uses the system
- Who is responsible for maintaining the system
- External hardware used by the system
- Other systems that need to interact with the system

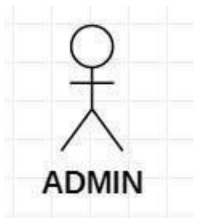
The needs of the actor are used to develop use cases. This insures that the system will be what the user expected.
Graphical depiction:
An actor is a stereotype of a class and is
depicted as a "stickman" on a use-case diagram. For example,



Actors identified in the information system are:
1) Admin
2) User

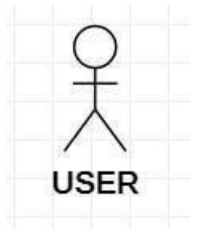
1) Admin: Admin has to login into his account
    - to view his/her profile,
    - to mange the admin/user profiles,
    - to create a new tour packages,
    - to update current tour packages,
    - to change his/her password request

  and he has to logout the account after the desired actions complete.



2) User: User has to login into his account
   - to view his/her profile,
   - to view tour packages,
   - book and make payment of the tour packages,
   - to update his/her profile,
   - to change his/her password request

and he has to logout the account after the desired actions complete.

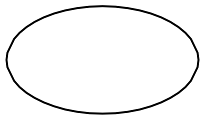


Identification of Use-Cases Or Sub Use-Cases

Use case can be described as a specific way of using the system from a user's perspective. A more detailed description might characterize a use case as:

- A pattern of behavior the system exhibits
- A sequence of related transactions performed by an actor and the system

The UML notation for use case is:



Login

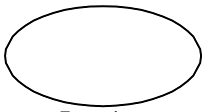
Purpose of usecases:

- Well structured use cases denote essential system or subsystem behaviours only, and are neither overly general nor too specific.
- A use case represents a functional requirement of the system as a whole
- Use cases represent an external view of the system
- A use case describes a set of sequences, in which each sequence represents the interaction of the things outside the system with the system itself.

Use-cases identified for tourism management system are:

1 .**Use-case name:**LOGIN

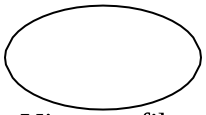
This is a use case which is used by actor to log on to the system and view the available set of operations that he can perform



Login

2. **Use-case name :**VIEW PROFILE

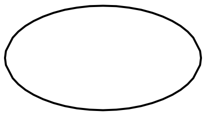
System allows user or admin to view his/her profile and can be able to select available functionalities respectively.



View profile

3. **Use-case name:** UPDATE PROFILE

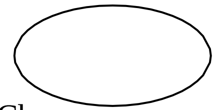
This use case allows user or admin to update his/her profile like updating existing email id, phone number, date of birth etc.,



Update profile

4.**Use-case name:** CHANGE PASSWORD

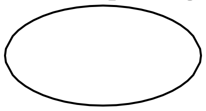
This use case allows the user or admin to change the password and secure delivery contact information.



Change password

5.**Use-case name:**VIEW TOUR PACKAGES

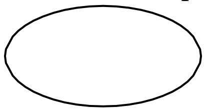
This use case allows the registered to view tour packages list.



View tour packages

6**.Use-case name:** MANAGE TOUR PACKAGES

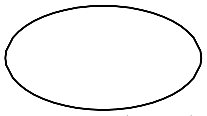
This use case allows admin to add/update/delete tour packages.



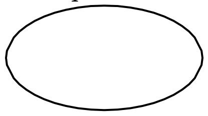
Manage tour packages

7.**Use-case name:** USER REGISTRATION

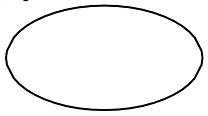
This usecase allows a person to register as a user into the system by supplying basic details.


User registration

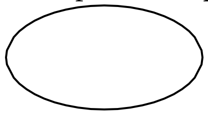
8. **Use-case name:** BOOKING TOUR PACKAGE
This usecase allows a person to select the required tour package.


Booking tour package

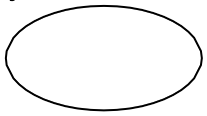
9.**Use-case name:**MAKING PAYMENET
This usecase allows a person to make payment according to their slected tour package.


Making payment

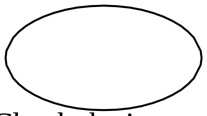
10.**Use-case name:**GENERATING INVOICES
This use case generate invoice after completion of payment.


Generating invoices

11.**Use-case name:** RAISE THE ISSUES
This use case allows user to raise any issues about the system.

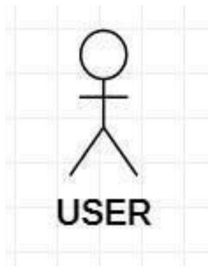

Raise the issues

12.**Use-case name:** CKECK THE ISSUES
This use case allows admin to check the issues raised by the user.


Check the issues

Identification of RELATIONS

Association Relationship:
An association provides a pathway for communication. The communication can be between use cases, actors, classes or interfaces. If two objects are usually considered independently, the relationship is an association.



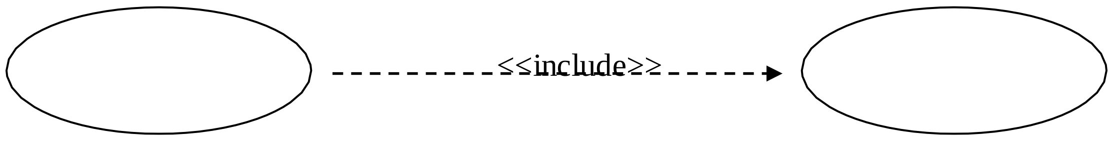
Dependency Relationship:

A dependency is a relationship between two model elements in which a change to one model element will affect the other model element. Use a dependency relationship to connect model elements with the same level of meaning.

We can provide here

1.Include relationship:

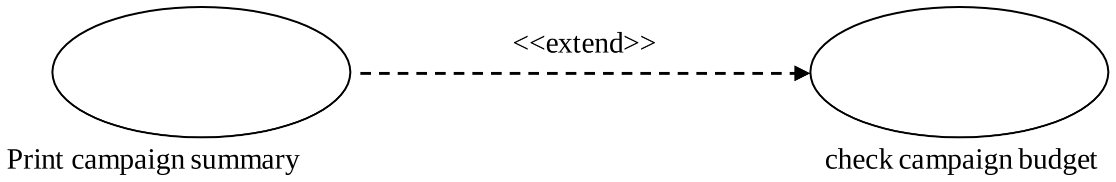
It is a stereotyped relationship that connects a base use case to an inclusion use case .An include relationship specifies how the behavior in the inclusion use case is used by the base use case.



Extend relationship:

It is a stereotyped relationship that specifies how the functionality of one use case can be inserted into the functionality of another use case.

<<extend>> is used when you wish to show that a use case provides additional functionality that may be required in another use case.

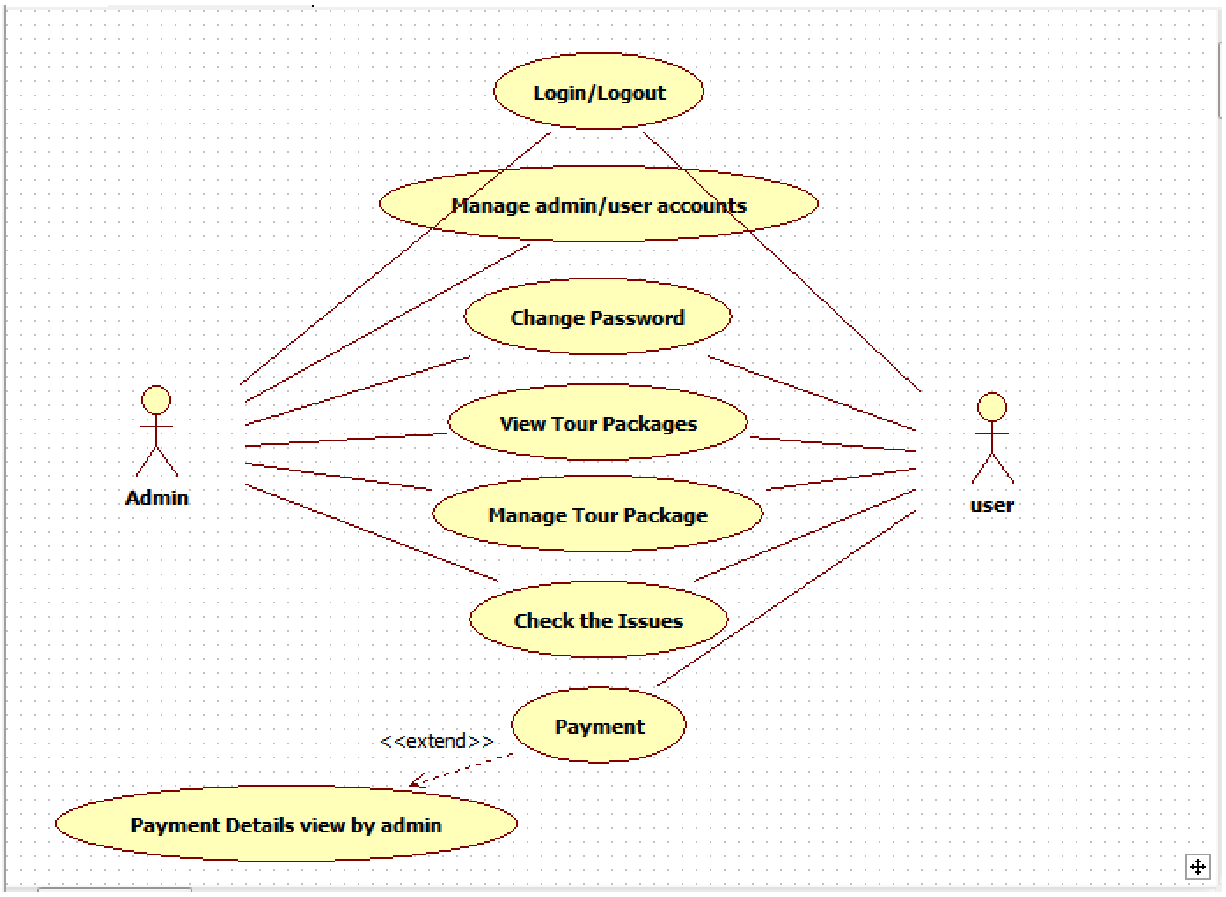


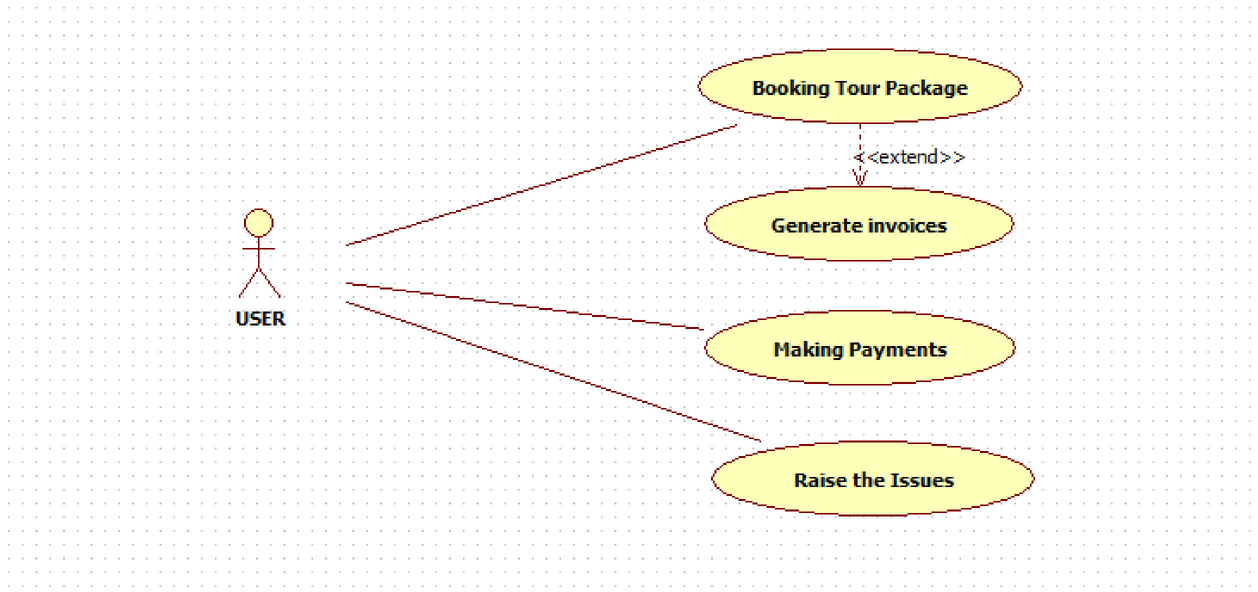
6. CONSTRUCTION OF USE CASE DIAGRAM AND FLOW OF EVENTS.

Use-case diagrams graphically represent system behavior. These diagrams present a high level view of how the system is used as viewed from an outsider's perspective.

Use-case diagrams can be used during analysis to capture the system requirements and to understand how the system should work. During the design phase, you can use use-case diagrams to specify the behavior of the system as implemented.

USE CASE DIAGRAMS FOR TMS:





FLOW OF EVENTS

A flow of events is a sequence of transactions performed by the system. They typically contain very detailed information .Flow of events document is typically created in the elaboration phase.

Each use case is documented with flow of events

- A description of events needed to accomplish required behaviour
- Written in terms of what the system should do, NOT how it should do it
- Written in the domain language , not in terms of the implementation

A flow of events should include

- When and how the use case starts and ends
- What interaction the use case has with the actors
- What data is needed by the use case
- The description of any alternate or exceptional flows

The flow of events for a use case is contained in a document called the use case specification. Each project should use a standard template for the creation of the use case specification. Includes the following

1. Use case name – Brief Description
2. Flow of events –
    1. Basic flow
    2. Alternate flow
    3. Special requirements
    4. Pre conditions
    5. Post conditions
    6. Extension points

FLOW OF EVENTS FOR MANAGE TOUR PACKAGES

1. Use case: Manage Tour Package.
Brief Description: This use case is started by Admin. It provides the capability for admin to add/delete/update the tour package.

2.Actor: Admin

3.Flow events:

Basic Flow:
***** This use case begins when admin logs into the system and enters his/her password. The system verifies that the password is valid.

- If the password is invalid alternate flow 3.3.1 is executed.

And displays the current tour packages. The system prmopts the admin to select the desired activity.

ADD / DELETE / UPDATE the Tour Package.

*If the activity selectes is **ADD**, the system displays the package screen contain package name, place1, place2, place3. The admin enters the package name, places.

-If the invalid place is entered, alternate flow 3.3.2 is executed.

*If the activity selected is **DELETE**, the system displays the package screen containing a field for package name. The admin enters the package name. Then an alert is given that the package is deleted.

-If the invalid package name is entered, alternate flow 3.3.3 is executed.

*If the activity selected is **UPDATE**. The system displays the screen containing name, place1, place2, place3 which are admin already assigned. So that admin can edit any of those details. Then an alert is given that the package is updated successfully.

-If the invalid package name or place name is entered, alternate flow 3.3.4 is executed.

Alternate Flow:

Invalid Password : Invalid password is entered by admin to login. Admin can re-enter a password or terminate the use case.

Invalid Place name : The system informs the admin that the place is invalid. The admin can re-enter the place or terminate the use case.

Invalid Package name : The system informs the admin that the package you want to delete is invalid. The admin can re-enter the package name or terminate the use case.

Invalid Package name or Place name : The system informs the admin that the pavkage name or place name is invalid. The admin can re-enter the package name or place name or terminate the use case.

4.Pre condition : Admin should first login to the website.

5.Post condition : Next time, Admin can be able to login through new password.

FLOW OF EVENTS FOR MAKING PAYMENT BY USER

1. Use case :. Making payment
Brief Description : This use case is started by user. It provides the capability for the user to make payment and generate bill.

2. Actor : User

3. Flow of Event:

Basic Flow:

***** This use case begins when admin logs into the system and enters his/her password. The system verifies that the password is valid.

- If the password is invalid alternate flow 3.3.1 is executed.

And displays the current tour packages. The system prompts the user to select the desired tour package.

*If the any one tour package is selected the system displays the payment screen containing of details selected package name, places, from date, end date.

The system displays the cost of the tour package which is selected.

-If the user thinks cost is not reasonable then the alternate flow 3.3.1 is executed.

The user make the payment. The user make the payment. And system generate the bill.

-If the transaction is failed then the alternate flow 3.3.2 is executed.

Alternate Flow:

Invalid Password : Invalid password is entered by user to login. User can re-enter a password or terminate the use case.

Un-reasonable cost : If the user thinks cost is high or not reasonable then user can change the tour package or terminate the use case.

Transaction Failed : If the transaction is failed, then user can again make the paymentor terminate the use acse.

4.Pre condition : User should login to the website.

5.Post condition : User can be able to view the generate bill of their payment.

7. BUILDING A BUSINESS PROCESS MODEL USING UML ACTIVITY DIAGRAM

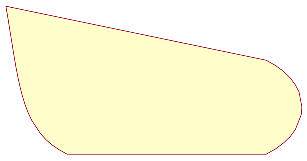
An Activity diagram is a variation of a special case of a state machine, in which the states are activities representing the performance of operations and the transitions are triggered by the completion of the operations. The purpose of Activity diagram is to provide a view of flows and what is going on inside a use case or among several classes. You can also use activity diagrams to model code-specific information such as a class operation. Activity diagrams are very similar to a flowchart because you can model a workflow from activity to activity. An activity diagram is basically a special case of a state machine in which most of the states are activities and most of the transitions are implicitly triggered by completion of the actions in the source activities.

- Activity Diagrams also may be created at this stage in the life cycle. These diagrams represent the dynamics of the system. They are flow charts that are used to show the workflow of a system; that is, they show the flow of control from activity to activity in the system, what activities can be done in parallel, and any alternate paths through the flow.
- At this point in the life cycle, activity diagrams may be created to represent the flow across use cases or they may be created to represent the flow within a particular use case.
- Later in the life cycle, activity diagrams may be created to show the workflow for an operation.

The following tools are used on the activity diagram toolbox to model activity diagrams:

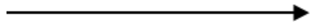
Activities:

An activity represents the performance of some behavior in the workflow.



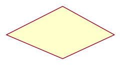
Transitions:

Transitions are used to show the passing of the flow of control from activity to activity. They are typically triggered by the completion of the behavior in the originating activity.



Decision Points:

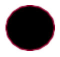
When modeling the workflow of a system it is often necessary to show where the flow of control branches based on a decision point. The transitions from a decision point contain a guard condition, which is used to determine which path from the decision point is taken. Decisions along with their guard conditions allow you to show alternate paths through a work flow.



Decision point

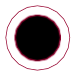
Start state:

A start state explicitly shows the beginning of a workflow on an activity diagram or the beginning of the execution of a state machine on a state chart diagram.



End state:

An End state represents a final or terminal state on an activity diagram or state chart diagram. Place an end state when you want to explicitly show the end of a workflow on an activity diagram or the end of a state chart diagram. Transitions can only occur into an end state; however, there can be any number of end states per context.



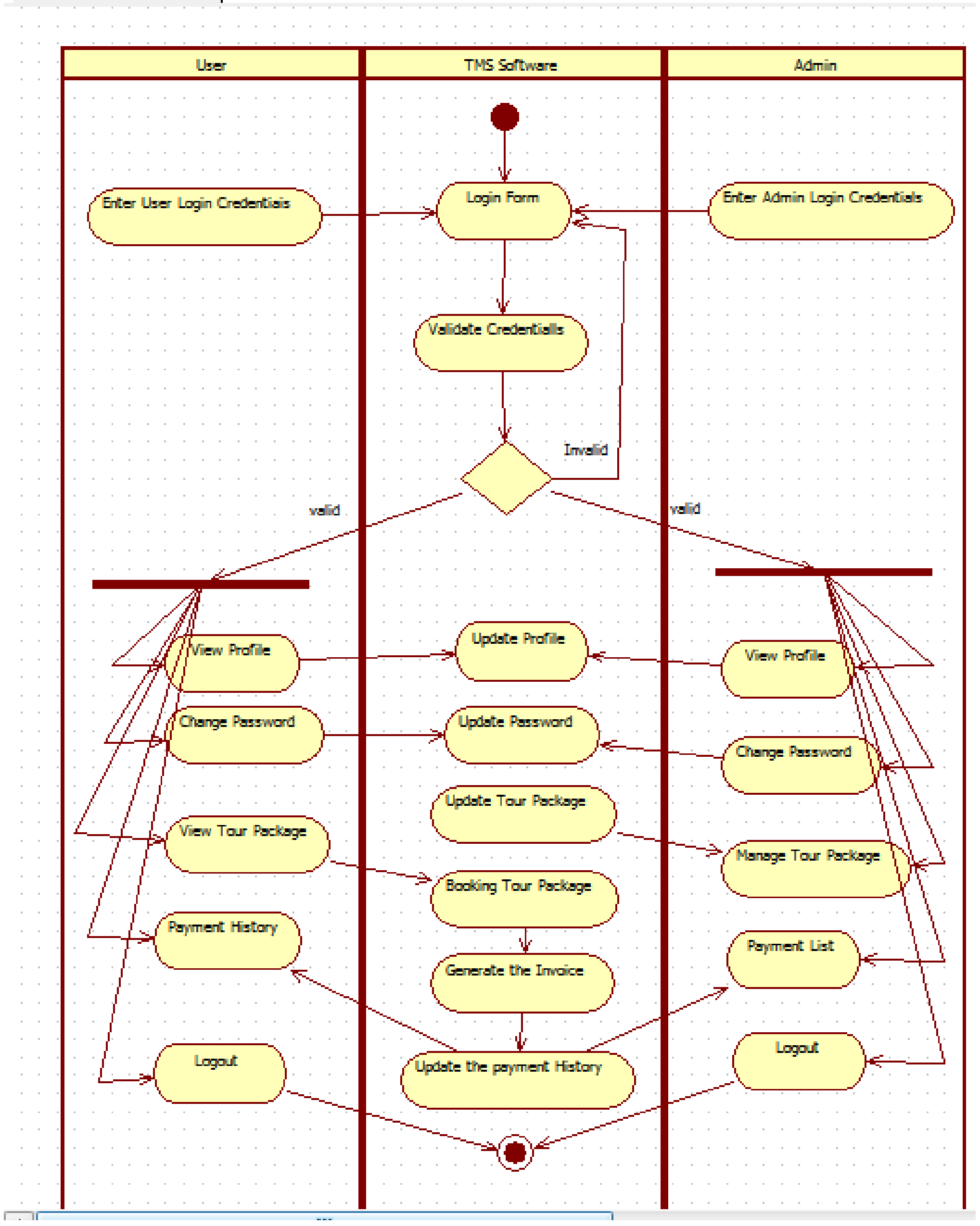
End state

Swim Lanes:

Swim lanes may be used to partition an activity diagram. This typically is done to show what person or organization is responsible for the activities contained in the swim lane.

- Horizontal synchronization
- Vertical synchronization.

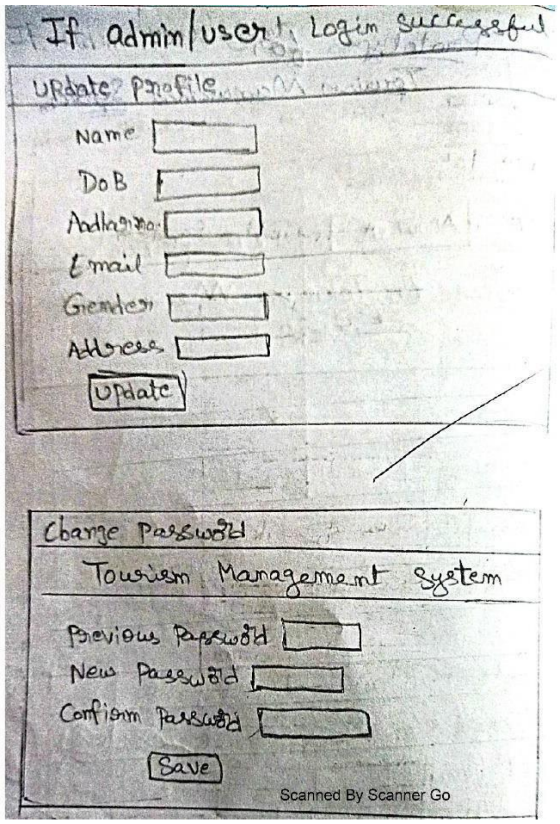
ACTIVITY DIGRAM FOR USER AND ADMIN :

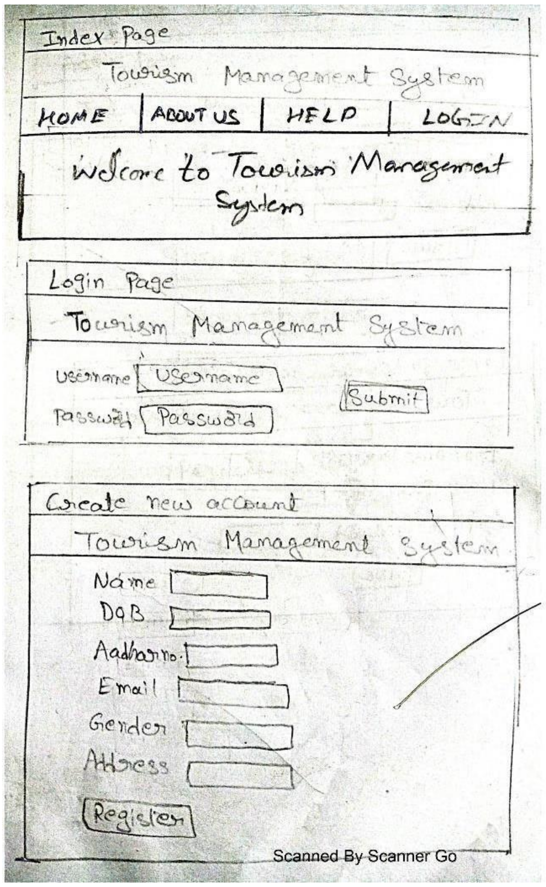


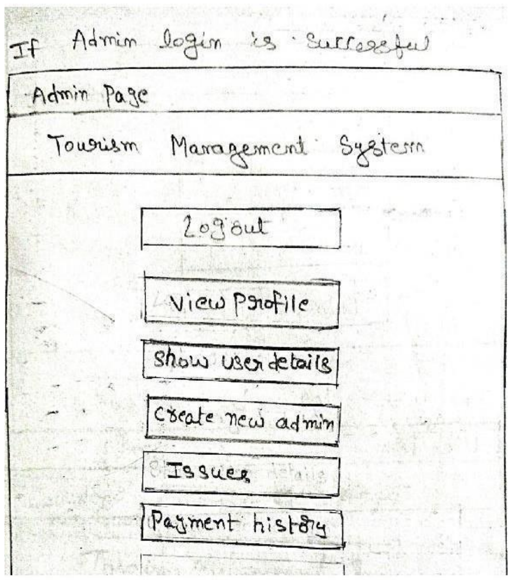
8. CONSTRUCTION OF PROTOTYPES

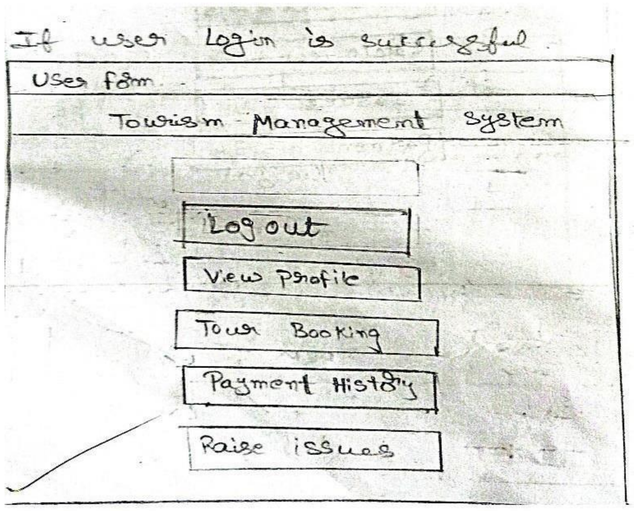
As the requirements for a system emerge in the form of use cases, it is sometimes helpful to build simple prototypes of how some of the use cases will work. A prototype is a working model of part of the system usually a program with limited functionality that is built to test out some aspect of how the system will work. Prototypes can be used to help elicit requirements. Showing users how the system might provide some of the use cases often produces a stronger reaction than showing them a series of abstract diagrams. Their reaction may contain useful information about requirements.

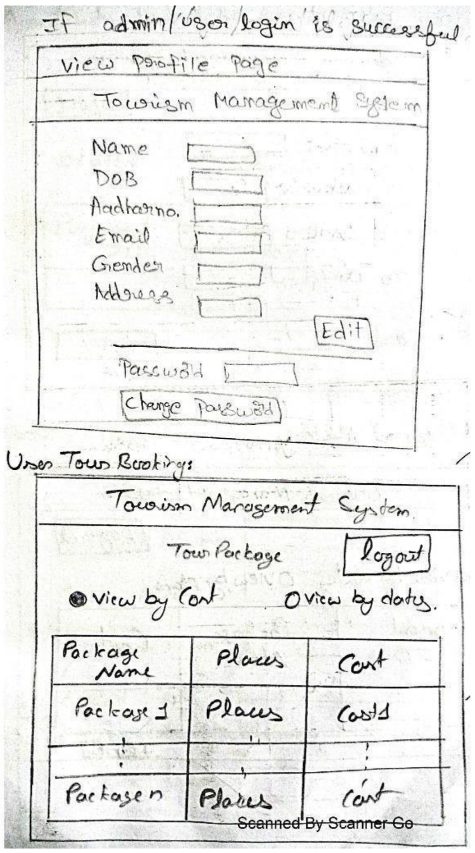
The following prototype is for TMS :

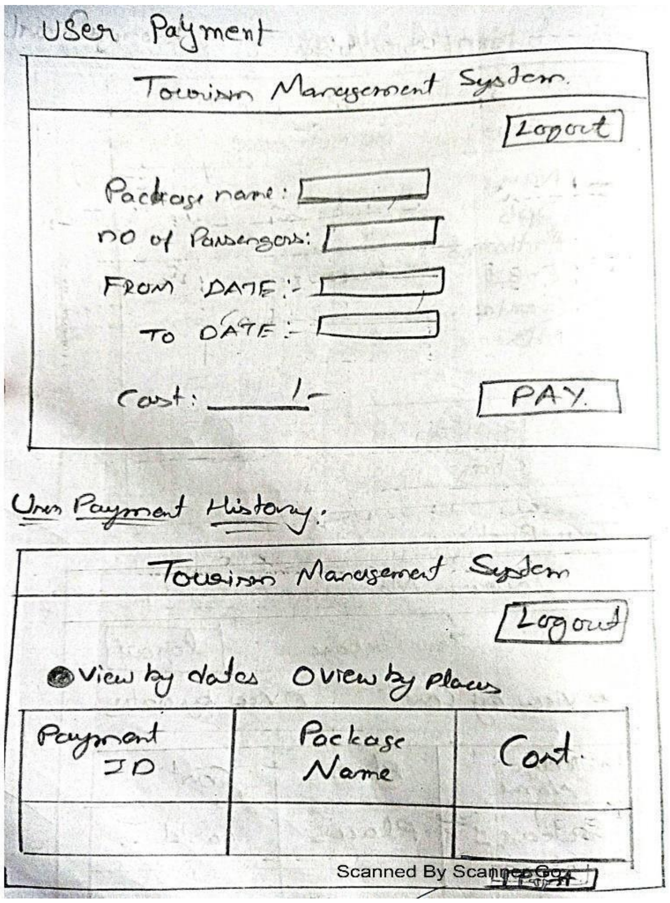


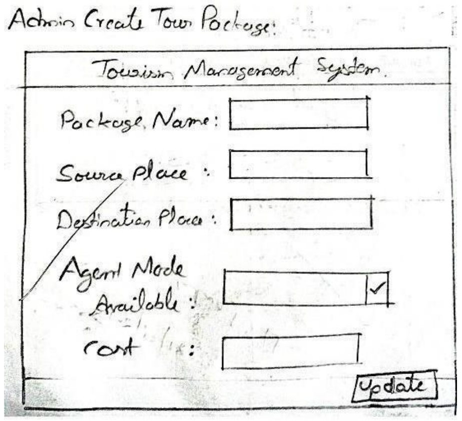



If Admin login is successful
Admin Page
Tourism Management System
Logout
View Profile
Show user details
Create new admin
Issues
Payment history

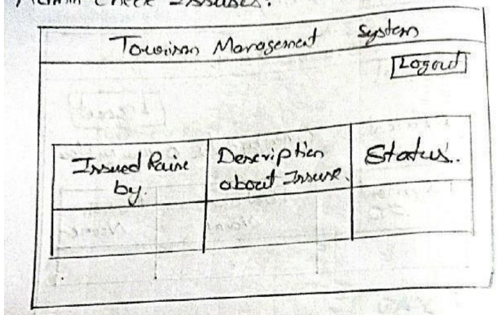

If user Login is successful.
User form.
Tourism Management System
Logout
View Profile
Tour Booking
Payment History
Raise issues


If admin/user login is successful
View Profile Page
Tourism Management System
Name
DOB
Aadharno.
Email
Gender
Address
Edit
Password
Change Password
Users Tour Bookings:
Tourism Management System
Tour Package
logout
View by Cost
View by dates.
Package Name
Places
Cost
Package 1
Places
Cost1
Package n
Places
Cost
Scanned By Scanner Go

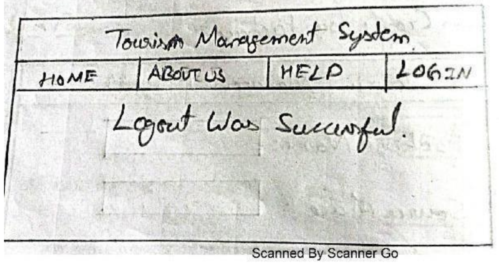

User Payment
Tourism Management System
Logout
Package name:
No of Passengers:
FROM DATE:-
TO DATE:-
Cost: ____ /-
PAY
User Payment History:
Tourism Management System
Logout
● View by dates ○ View by places
Payment ID
Package Name
Cost
Scanned By Scanner Go


Admin Create Tour Package:
Tourism Management System
Package Name:
Source Place:
Destination Place:
Agent Mode Available:
Cost:
Update

Admin Check Issues:



Admin user logout after work.



Scanned By Scanner Go

9. CONSTRUCTION OF SEQUENCE DIAGRAMS.

A sequence diagram is a graphical view of a scenario that shows object interaction in a time based sequence--what happens first, what happens next… Sequence diagrams establish the roles of objects and help provide essential information to determine class responsibilities and interfaces.

A sequence diagram has two dimensions: the vertical dimension represents time; the horizontal dimension represents different objects. The vertical line is called the object's lifeline. The lifeline represents the object's existence during the interaction. Steps:

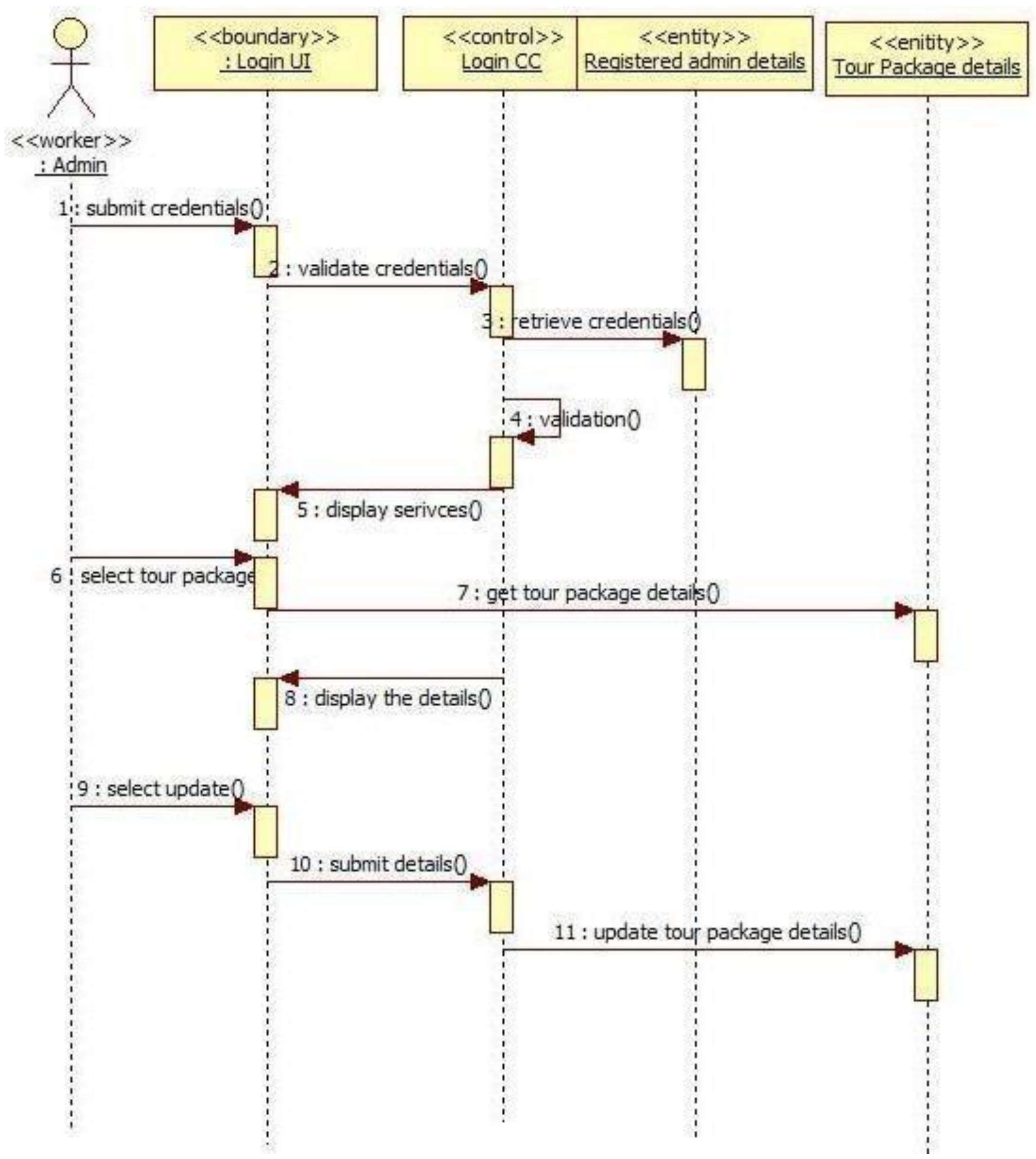
1. An object is shown as a box at the top of a dashed vertical line. Object names can be specific (e.g., Algebra 101, Section 1) or they can be general (e.g., a course offering). Often, an anonymous object (class name may be used to represent any object in the class.)
2. Each message is represented by an Arrow between the lifelines of two objects. The order in which these messages occur is shown top to bottom on the page. Each message is labeled with the message name.

There are two main differences between sequence and collaboration diagrams: sequence diagrams show time-based object interaction while collaboration diagrams show how objects associate with each other. A sequence diagram has two dimensions: typically, vertical placement represents time and horizontal placement represents different objects.

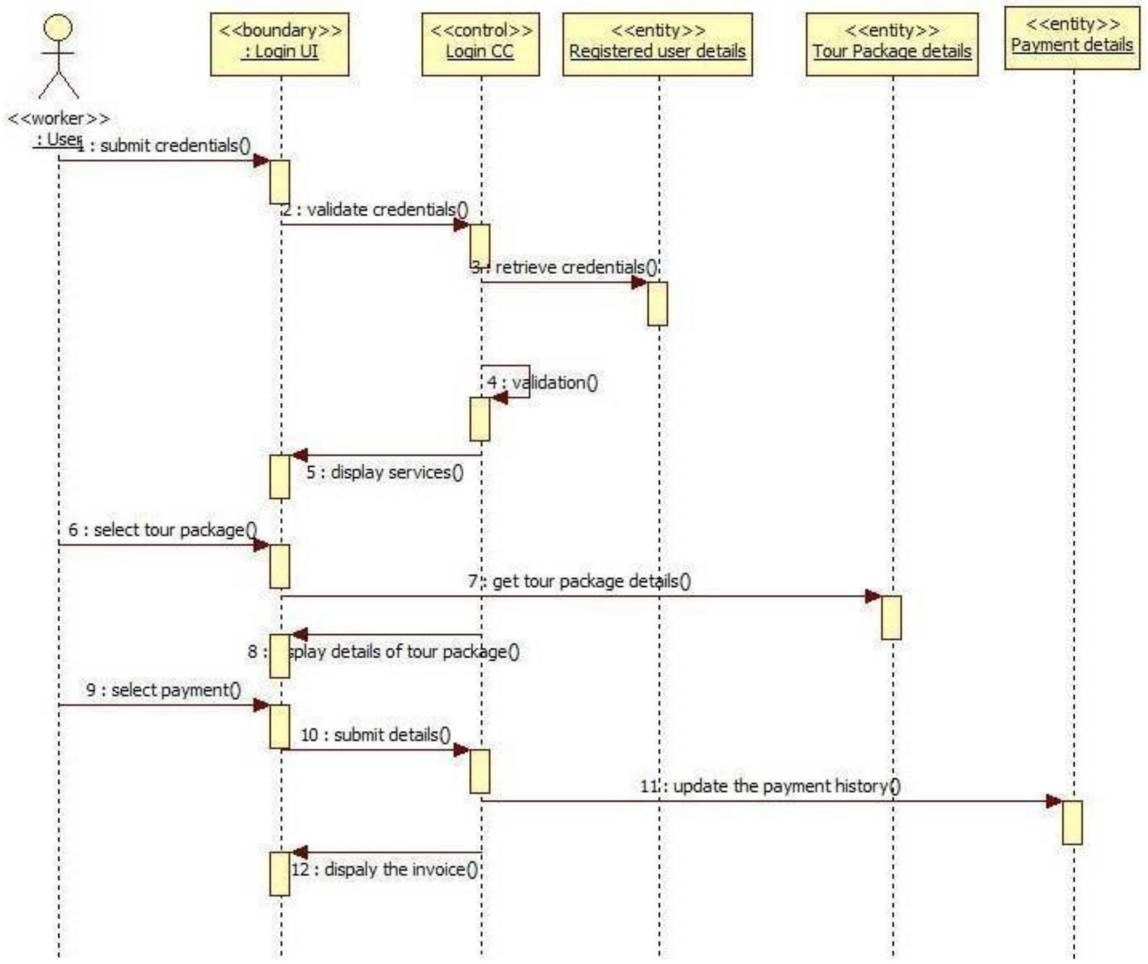
ELEMENTS OF SEQUENCE DIAGRAM:

- Objects
- Links
- Messages
- Focus of control
- Object life line

SEQUENCE DIAGRAM FOR MANAGING TOUR PACKAGES BY THE ADMIN :



SEQUENCE DIAGRAM FOR MAKING PAYMENT BY THE USER FOR BOOKING TOUR PACKAGES :



10. CONSTRUCTION OF COLLABORATION DIAGRAMS

Collaboration diagrams are the second kind of interaction diagram in the UML diagrams. They are used to represent the collaboration that realizes a use case. The most significant difference between the two types of interaction diagram is that a collaboration diagram explicitly shows the links between the objects that participate in a collaboration , as in sequence diagrams, there is no explicit time dimension. **Message labels in collaboration diagrams:**

Messages on a collaboration diagram are represented by a set of symbols that are the same as those used in a sequence diagram, but with some additional elements to show sequencing and recurrence as these cannot be inferred from the structure of the diagram. Each message label includes the message signature and also a sequence number that reflects call nesting, iteration, branching, concurrency and synchronization within the interaction.

The formal message label syntax is as follows:
[predecessor] [guard-condition] sequence-expression [return-value ':='] message-name' (' [argument-list] ')'

A predecessoris a list of sequence numbers of the messages that must occur before the current message can be enabled. This permits the detailed specification of branching pathways. The message with the immediately preceding sequence number is assumed to be the predecessor by default, so if an interaction has no alternative pathways the predecessor list may be omitted without any ambiguity. The syntax for a predecessor is as follows:
sequence-number { ',' sequence-number} *'I'*

The *'I'* at the end of this expression indicates the end of the list and is only included when an explicit predecessor is shown.

Guard conditionsare written in Object Constraint Language (OCL) ,and are only shown where the enabling of a message is subject to the defined condition. A guard condition may be used to represent the synchronization of different threads of control.

A sequence-expressionis a list of integers separated by dots ('.') optionally followed by a *name* (a single letter), optionally followed by a *recurrence* term and terminated by a colon. A sequence-expression has the following syntax:
integer { '.' integer } [name] [recurrence] ':'
In this expression *integer* represents the sequential order of the message. This may be nested within a loop or a branch construct, so that, for example, message 5.1 occurs after message 5.2 and both are contained within the activation of message 5.

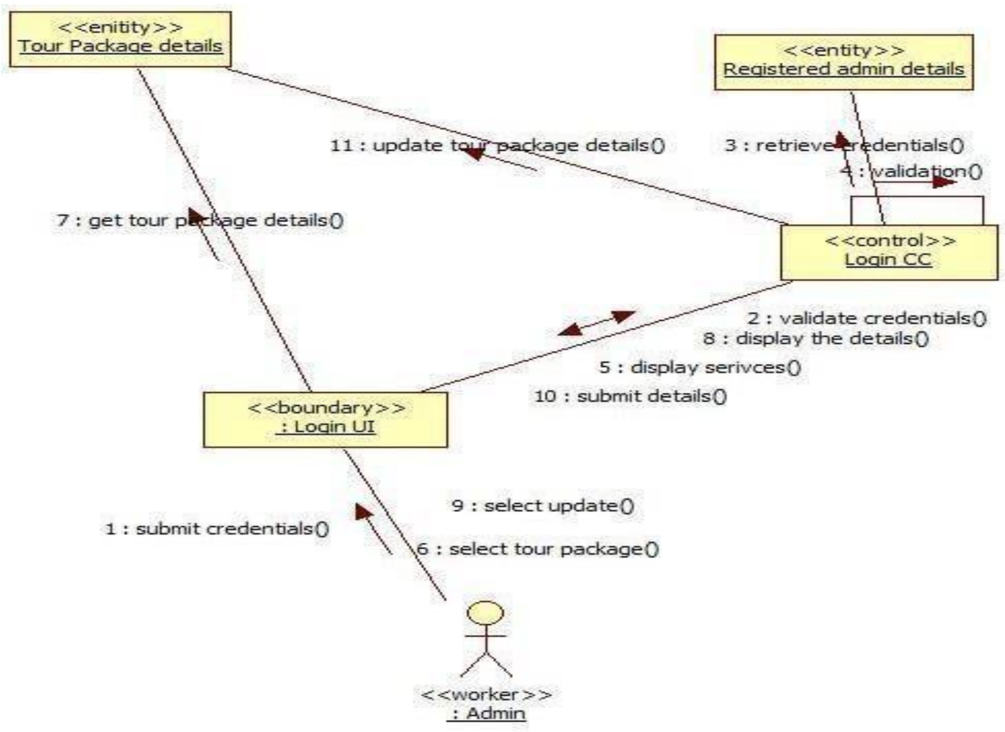
The *name* of a sequence-expression is used to differentiate two concurrent messages since these are given the same sequence number. For example, messages 3.2.1a and 3.2.1b are concurrent within the activation of message 3.2.
Recurrence reflects either iterative or conditional execution and its syntax is as follows:
Branching:'['condition-clause‘] ,

Iteration:‘ * ‘ ‘ [‘ iteration-clause ‘] '

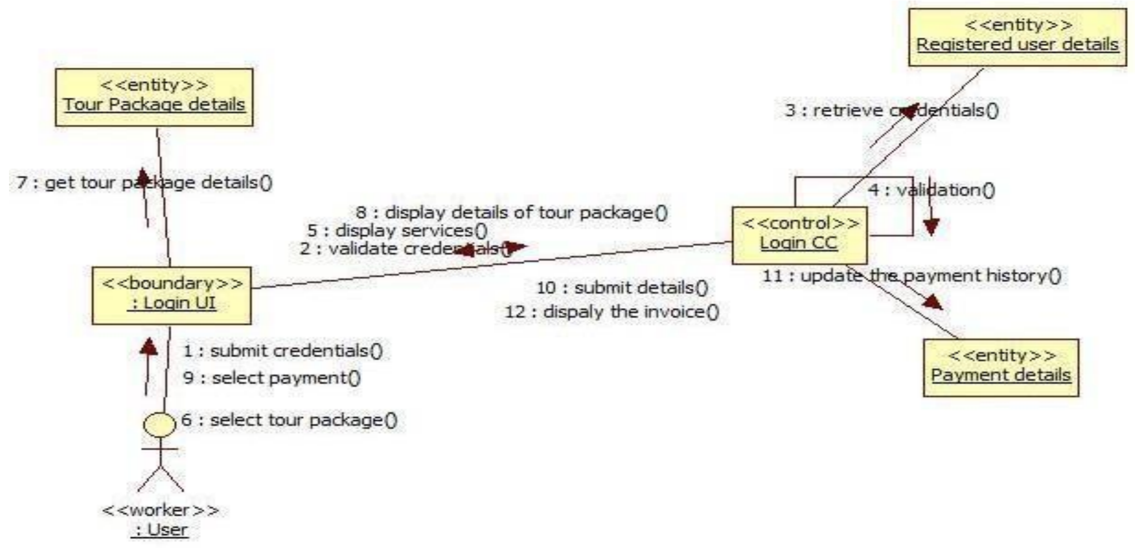
Elements:

• Objects ,Messages ,Path,SequenceNumbers,Links

COLLABORATION DIAGRAM FOR MANAGING TOUR PACKAGES BY THE ADMIN :



COLLABORATION DIAGRAM FOR MAKING PAYMENT BY THE USER FOR BOOKING TOUR PACKAGE :



11. CONSTRUCTION OF UML STATIC CLASS DIAGRAM

Class diagrams contain icons representing classes, packages, interfaces, and their relationships. You can create one or more class diagrams to depict the classes at the top level of the current model; such class diagrams are themselves contained by the top level of the current model.

Class:

A Class a description of a group of objects with common properties (attributes), common behavior (operations), common relationships to other objects, and common semantics.

Thus, a class is a template to create objects. Each object is an instance of some class and objects cannot be instances of more than one class.

Classes should be named using the vocabulary of the domain.

For example, the CourseOffering class may be defined with the following characteristics:

Attributes - location, time offered

Operations - retrieve location, retrieve time of day, add a student to the offering .

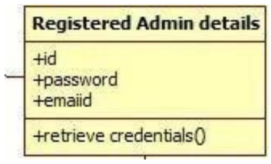
Each object would have a value for the attributes and access to the operations specified by the CourseOffering class.

In the UML, classes are represented as compartmentalized rectangles.

The top compartment contains the name of the class.

The middle compartment contains the structure of the class (attributes).

The bottom compartment contains the behavior of the class (operations) as shown below.



OBJECT :

- AN OBJECT IS a representation of an entity, either real-world or conceptual.

- An object is a concept, abstraction, or thing with well defined boundaries and

meaning for an application.

- Each object in a system has three characteristics: state, behavior, and identity.

STATE : THE STATE OF an object is one of the possible conditions in which it may exist. The state of an object typically changes over time, and is defined by a set of properties (called attributes), with the values of the properties, plus the relationships the object may have with other objects.

For example, a course offering object in the registration system may be in one of two states: *open* and *closed*. It is available in the open state if value is < 10 otherwise closed.

Behavior :

- Behavior determines how an object responds to requests from other objects .

- Behavior is implemented by the set of operations for the object.

For example , In the registration system, a course offering could have the behaviors *add* a *student* and *delete* a *student.*

Identity :

- Identity means that each object is unique even if its state is identical to that of another object.

Attributes :

Attributes are part of the essential description of a class. They belong to the class, unlike objects, which instantiate the class. Attributes are the common structure of what a member of the class can 'know'. Each object will have its own, possibly unique, value for each attribute.

Guidelines for identifying attributes of classes are as follows:

- Attributes usually correspond to nouns followed by prepositional phrases ➢Keep the class simple; state only enough attribute to defineobject state.
- Attributes are less likely to be fully described in the problem statement.
- Omit derived attributes.
- Do not carry discovery attributes to excess.

STEREOTYPES AND CLASSES :

As like stereotypes for relationships in use case diagrams. Classes can also have stereotypes. Here a stereotype provides the capability to create a new kind of modeling element. Here, we can create new kinds of classes. Some common stereotypes for a class are entity Class, boundary Class, control class, and exception.

Entity Classes

- An **entity class** models information and associated behavior that is generally long lived.
- This type of class may reflect a real-world entity or it may be needed to perform tasks internal to the system.
- They are typically independent of their surroundings; that is, they are not sensitive to how the surroundings communicate with the system.

Boundary Classes :

Boundary classes handle the communication between the system surroundings and the inside of the system. They can provide the interface to a user or another system (i.e., the interface to an actor). They constitute the surroundings dependent part of the system. Boundary classes are used to model the system interfaces.

Boundary classes are also added to facilitate communication with other systems. During design phase, these classes are refined to take into consideration the chosen communication protocols.

Control Classes

- Control classes model sequencing behavior specific to one or more use cases.

- Control classes coordinate the events needed to realize the behavior specified in the use case.
- Control classes typically are application-dependent classes.

In the early stages of the Elaboration Phase, a control class is added for each actor/use case pair. The control class is responsible for the flow of events in the use case.

NEED FOR RELATIONSHIPS AMONG CLASSES:

All systems are made up of many classes and objects. System behaviour is achieved through the collaborations of the objects in the system.

Two types of relationships in CLASS diagram are:

1. Association Relationship
2. Aggregation Relationship

1. Association Relationship:

An association is a bidirectional semantic connection between classes. It is not a data flow as defined in structured analysis and design data may flow in either direction across the association. An association between classes means that there is a link between objects in the associated classes.

2. Aggregation Relationship:

An aggregation relationship is a specialized form of association in which a whole is related to its part(s). Aggregation is known as a "part-of" or containment relationship. The UML notation for an aggregation relationship is an association with a diamond next to the class denoting the aggregate(whole).

3. Super-sub structure (Generalization Hierarchy):

These allow objects to be build from other objects. The super-sub class hierarchy is a relationship between classes, where one class is the parent class of another class.

NAMING RELATIONSHIP:

An association may be named. Usually the name is an active verb or verb phrase that communicates the meaning of the relationship. Since the verb phrase typically implies a reading direction, it is desirable to name the association so it reads correctly from left to right or top to bottom. The words may have to be changed to read the association in the other direction (e.g., Buses are allotted to Routes). It is important to note that the name of the association is optional.

ROLE NAMES:

The end of an association where it connects to a class is called an association role. Role names can be used instead of association names.

A role name is a noun that denotes how one class associates with another. The role name is placed on the association near the class that it modifies, and may be placed on one or both ends of an association line.

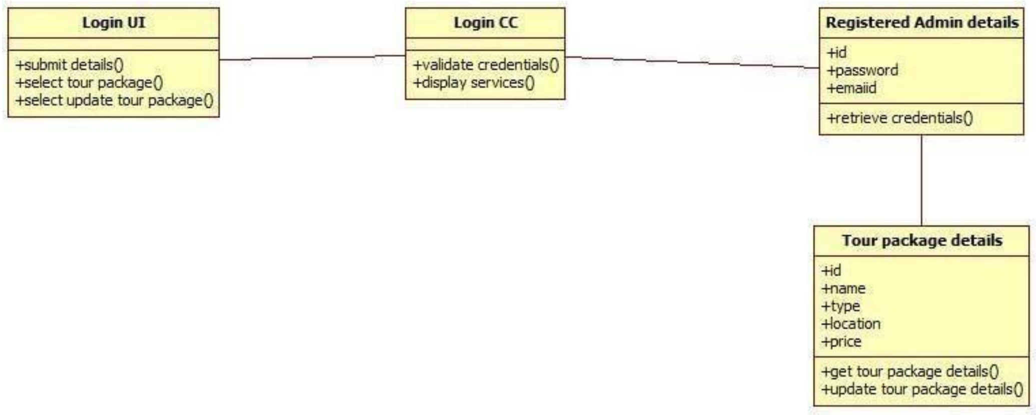
- It is not necessary to have both a role name and an association name.
- Associations are named or role names are used only when the names are needed for clarity.

MULTIPLICITY INDICATORS:

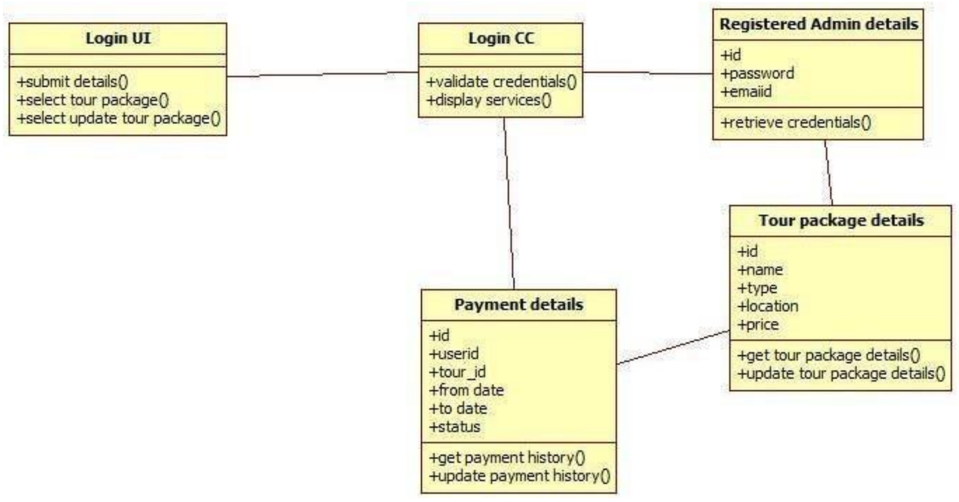
Although multiplicity is specified for classes, it defines the number of objects that participate in a relationship. Multiplicity defines the number of objects that are linked to one another. There are two multiplicity indicators for each association or aggregation one at each end of the line. Some common multiplicity indicators are

1	Exactly one
0... *	Zero or more
1... *	One or more
0... 1	Zero or one
5... 8	Specific range (5, 6, 7, or 8)
4... 7, 9	Combination (4, 5, 6, 7, or 9)

CLASS DIAGRAM FOR MANAGING TOUR PACKAGE BY ADMIN :



CLASS DIAGRAM FOR TOUR PACKAGE PAYMENT BY THE USER :



12. ANALYZING THE OBJECT BEHAVIOR BY CONSTRUCTING THE UML STATE CHART DIAGRAM

Use cases and scenarios provide a way to describe system behavior; in the form of interaction between objects in the system. Sometimes it is necessary to consider inside behavior of an object.

A state chart diagram shows the **states** of a single object, the events or messages that cause a **transition** from one state to another, and the **actions** that result from a state change. As in Activity diagram , state chart diagram also contains special symbols for start state and stop state.

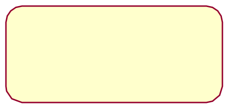
State chart diagram cannot be created for every class in the system , it is only for those class objects with significant behavior.

State chart diagrams are closely related to activity diagrams. The main difference between the two diagrams is state chart diagrams are state centric, while activity diagrams are activity centric. A state chart diagram is typically used to model the discrete stages of an object's lifetime, whereas an activity diagram is better suited to model the sequence of activities in a process.

STATE:

A state represents a condition or situation during the life of an object during which it satisfies some condition, performs some action or waits for some event.

UML notation for STATE is



To identify the states for an object its better to concentrate on sequence diagram. In an ESU the object for Course Offering may have in the following states, initialization, open and closed state. These states are obtained from the attribute and links defined for the object. Each state also contains a compartment for actions.

Actions:

Actions on states can occur at one of four times:

- on entry
- on exit
- do
- on event.

- **on entry :**What type of action that object has to perform after entering into the state. **on exit :** What type of action that object has to perform after exiting from the state. **Do :**The task to be performed when object is in this state, and must to continue until it leaves the state. **on event :** An on event action is similar to a state transition label with the following
- syntax: event(args)[condition] : the Action **State Transition:**
- A state transition indicates that an object in the source state will perform certain specified actions and enter the destination state when a specified event occurs or when certain conditions are satisfied. A state transition is a relationship between two states, two activities, or between an activity and a state. You can show one or more state transitions from a state as long as each transition is unique. Transitions

originating from a state cannot have the same event, unless there are conditions on the event.

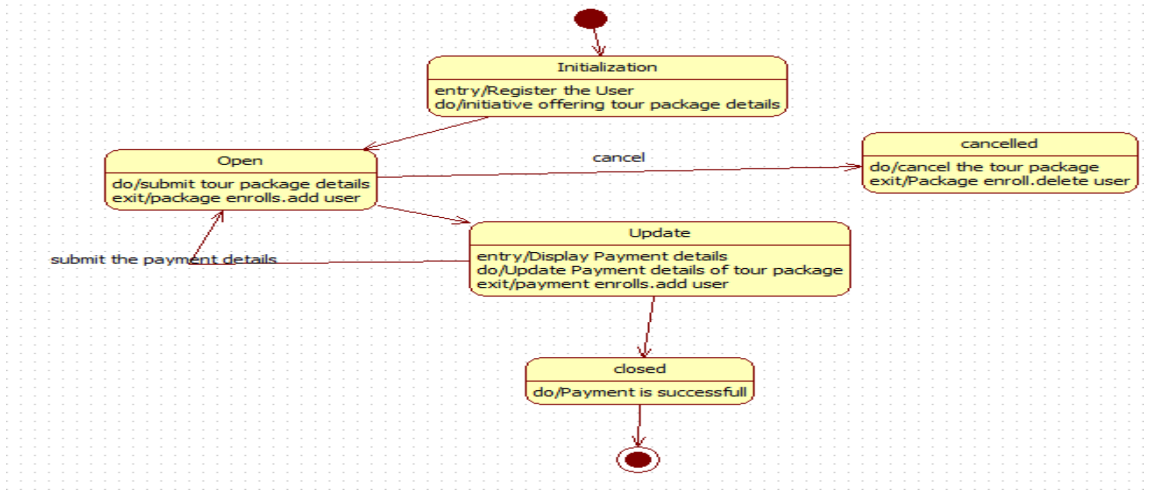
- **Transitions are labeled with the following syntax:**event arguments) [condition] / action ^ target. send Event (arguments) Only one event is allowed per transition, and one action per event.
- State Details :
- Actions that accompany all state transitions into a state may be placed as an entry action within the state. Like wise that accompany all state transitions out of a state may be placed as exit actions within the state. Behavior that occurs within the state is called an activity.
- An activity starts when the state is entered and either completes or is interrupted by an outgoing state transition. The behavior may be a simple action or it may be an event sent to another object.
-
- UML notation for State Details:

StateName
entry/ simple action
entry/ class name.eventname
do/ simple action
do/ class name.event name
exit/ class name.event name

Purpose of State chart diagram:

- State chart diagrams are used to model dynamic view of a system.
- State chart diagrams are used to modelling lifetime of an object.
- State chart diagrams are used to focus on the changing state of a system driven by events.
- It will also be used when showing the behavior of a class over several use cases.

STATE CHART DIAGRAM FOR MAKING PAYMENT


Initialization
entry/Register the User
do/initiative offering tour package details
Open
do/submit tour package details
exit/package enrolls.add user
cancel
cancelled
do/cancel the tour package
exit/Package enroll.delete user
Update
entry/Display Payment details
do/Update Payment details of tour package
exit/payment enrolls.add user
submit the payment details
closed
do/Payment is successfull

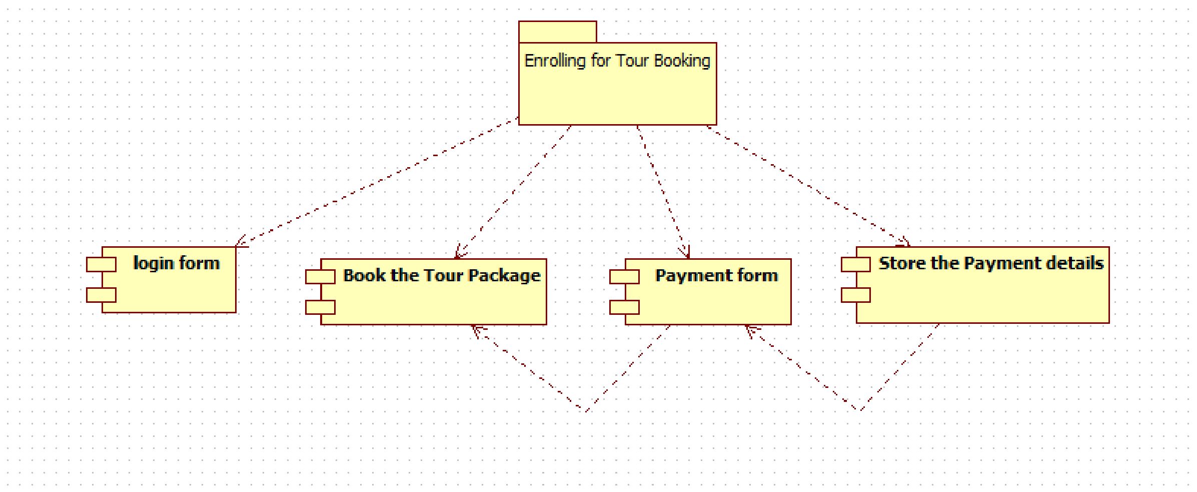
13. CONSTRUCTION OF IMPLEMENTATION DIAGRAMS

Component diagrams:

In a large project there will be many files that make up the system. These files will have dependencies on one another. The nature of these dependencies will depend on the language or languages used for the development and may exist at compile-time, at link-time or at run-time. There are also dependencies between source code files and the executable files or byte code files that are derived from them by compilation. Component diagrams are one of the twotypes of implementation diagram in UML. Component diagrams show these dependencies between software components in the system. Stereotypes can be used to show dependencies that are specific to particular languages also.

A component diagram shows the allocation of classes and objects to components in the physical design of a system. A component diagram may represent all or part of the component architecture of a system along with dependency relationships.

COMPONENT DIAGRAM FOR TMS



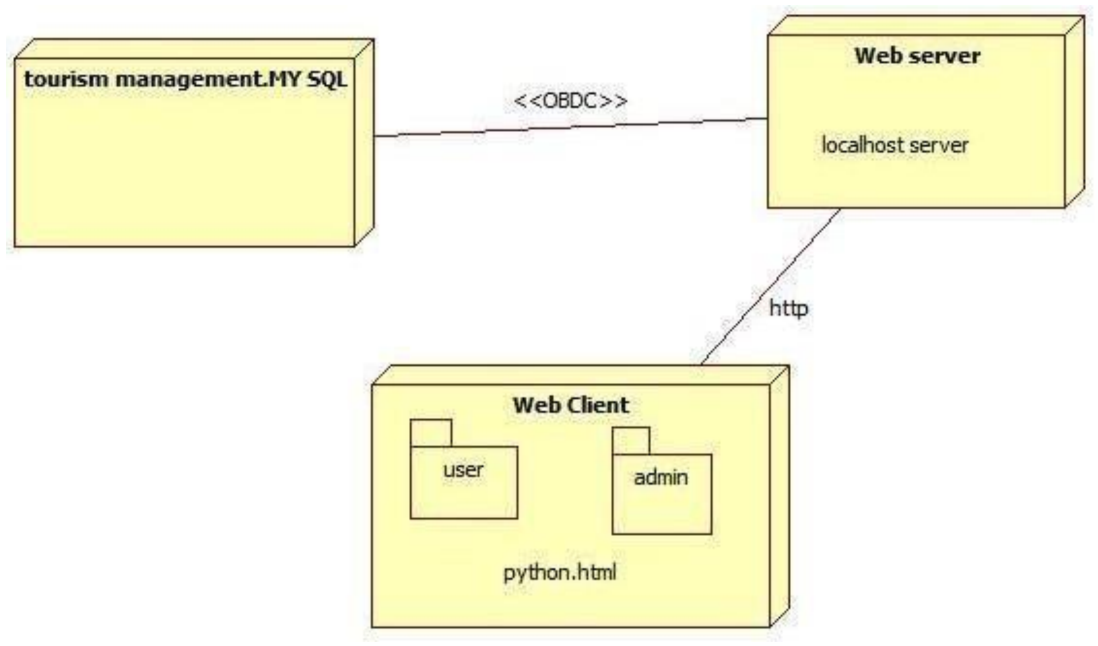
Deployment diagrams:

The second type of implementation diagram provided by UML is the deployment diagram. Deployment diagrams are used to show the configuration of run-time processing elements and the software components and processes that are located on them.

Deployment diagrams are made up of nodes and communication associations. Nodes are typically used to show computers and the communication associations show the network and protocols that are used to communicate between nodes. Nodes can be used to show other processing resources such as people or mechanical resources.

Nodes are drawn as 3D views of cubes or rectangular prisms, and the following figure shows a simplest deployment diagram where the nodes connected by communication associations.

DEPLOYEMENT DIAGRAM FOR TMS

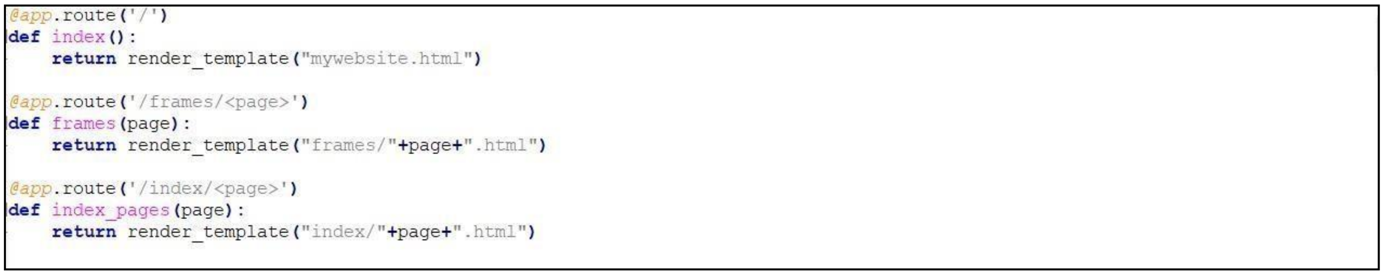


14. SAMPLE APPLICATION CODE AND DATABASE

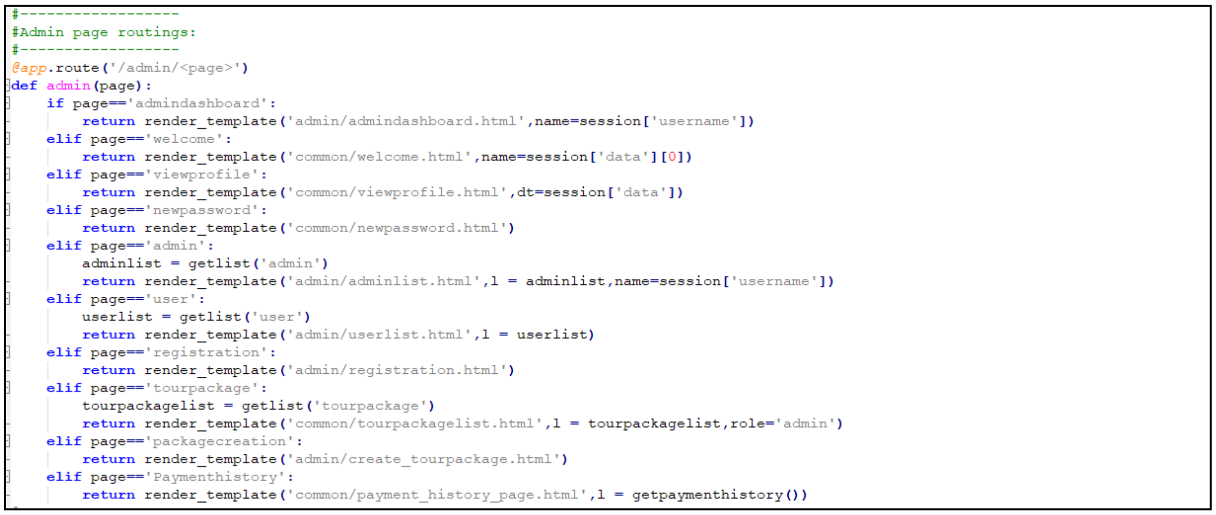
Various modules in the system are

1. Registration
2. Login
3. View Profile
4. Change Password
5. Update Profile
6. Tour Package Management
7. Payment Operations

CODE FOR INDEX PAGE

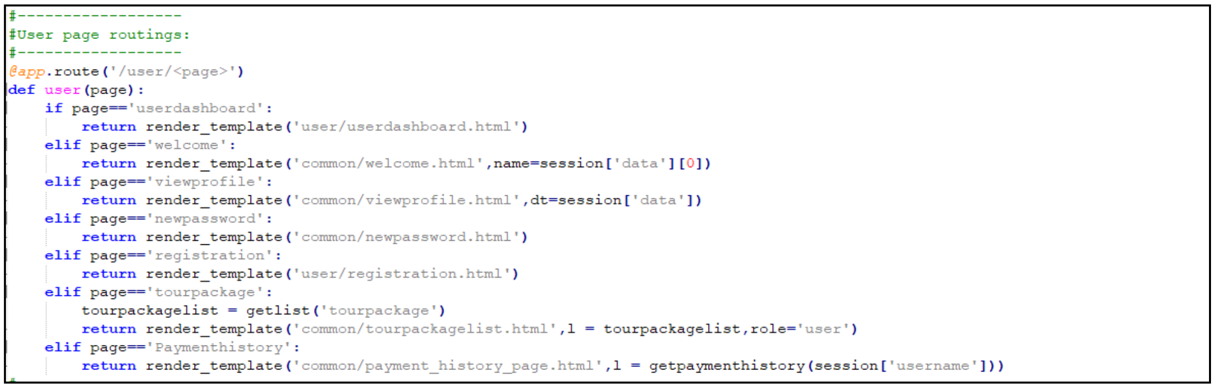


```
@app.route('/')
def index():
    return render_template("mywebsite.html")

@app.route('/frames/<page>')
def frames(page):
    return render_template("frames/"+page+".html")

@app.route('/index/<page>')
def index_pages(page):
    return render_template("index/"+page+".html")
```



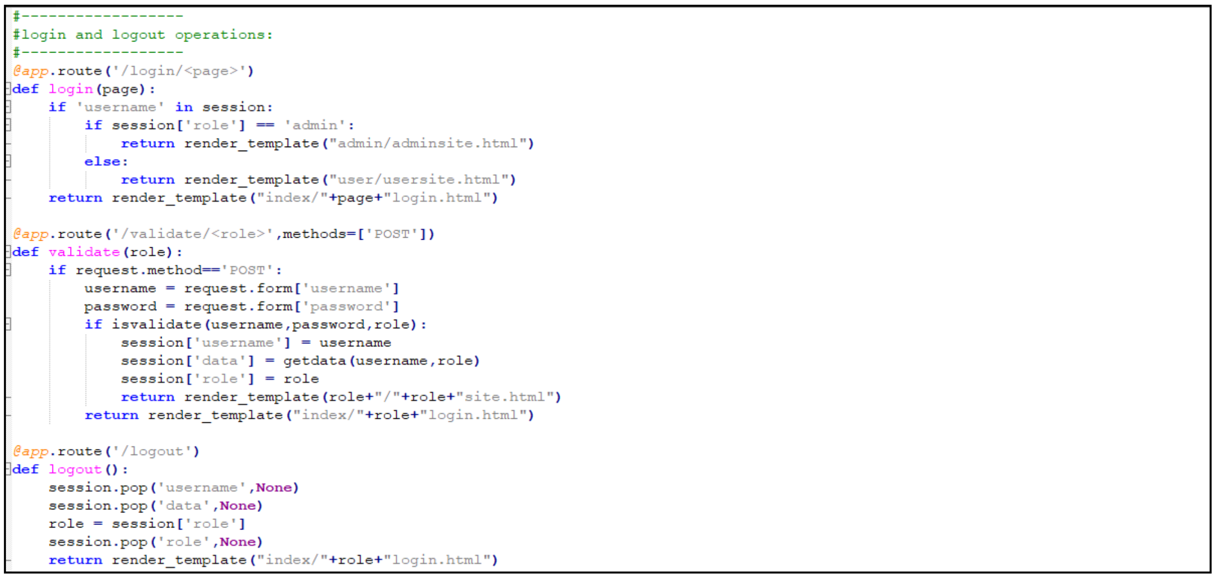
```
#------------------
#Admin page routings:
#------------------
@app.route('/admin/<page>')
def admin(page):
    if page=='admindashboard':
        return render_template('admin/admindashboard.html',name=session['username'])
    elif page=='welcome':
        return render_template('common/welcome.html',name=session['data'][0])
    elif page=='viewprofile':
        return render_template('common/viewprofile.html',dt=session['data'])
    elif page=='newpassword':
        return render_template('common/newpassword.html')
    elif page=='admin':
        adminlist = getlist('admin')
        return render_template('admin/adminlist.html',l = adminlist,name=session['username'])
    elif page=='user':
        userlist = getlist('user')
        return render_template('admin/userlist.html',l = userlist)
    elif page=='registration':
        return render_template('admin/registration.html')
    elif page=='tourpackage':
        tourpackagelist = getlist('tourpackage')
        return render_template('common/tourpackagelist.html',l = tourpackagelist,role='admin')
    elif page=='packagecreation':
        return render_template('admin/create_tourpackage.html')
    elif page=='Paymenthistory':
        return render_template('common/payment_history_page.html',l = getpaymenthistory())
```

CODE FOR USER LOAD PAGES



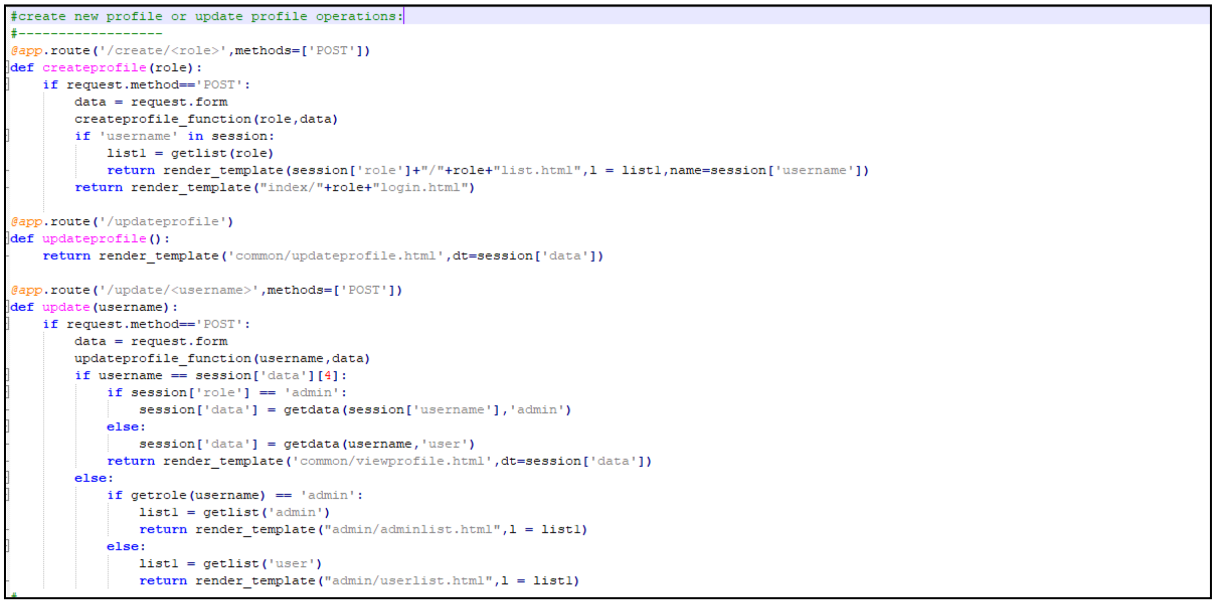
```
#------------------
#User page routings:
#------------------
@app.route('/user/<page>')
def user(page):
    if page=='userdashboard':
        return render_template('user/userdashboard.html')
    elif page=='welcome':
        return render_template('common/welcome.html',name=session['data'][0])
    elif page=='viewprofile':
        return render_template('common/viewprofile.html',dt=session['data'])
    elif page=='newpassword':
        return render_template('common/newpassword.html')
    elif page=='registration':
        return render_template('user/registration.html')
    elif page=='tourpackage':
        tourpackagelist = getlist('tourpackage')
        return render_template('common/tourpackagelist.html',l = tourpackagelist,role='user')
    elif page=='Paymenthistory':
        return render_template('common/payment_history_page.html',l = getpaymenthistory(session['username']))
```

CODE FOR SIGN IN/SIGN OUT OPERATIONS

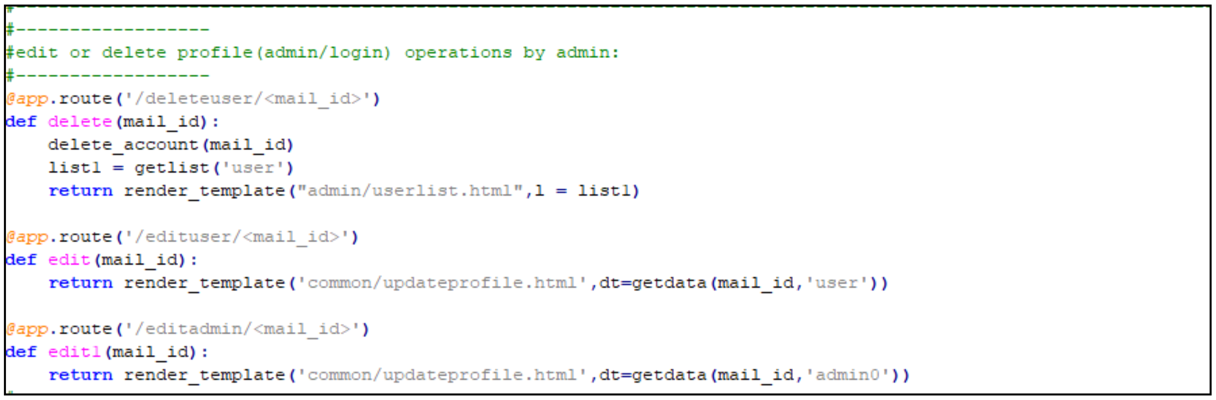


```
#------------------
#login and logout operations:
#------------------
@app.route('/login/<page>')
def login(page):
    if 'username' in session:
        if session['role'] == 'admin':
            return render_template("admin/adminsite.html")
        else:
            return render_template("user/usersite.html")
    return render_template("index/"+page+"login.html")

@app.route('/validate/<role>',methods=['POST'])
def validate(role):
    if request.method=='POST':
        username = request.form['username']
        password = request.form['password']
        if isvalidate(username,password,role):
            session['username'] = username
            session['data'] = getdata(username,role)
            session['role'] = role
            return render_template(role+"/"+role+"site.html")
        return render_template("index/"+role+"login.html")

@app.route('/logout')
def logout():
    session.pop('username',None)
    session.pop('data',None)
    role = session['role']
    session.pop('role',None)
    return render_template("index/"+role+"login.html")
```

CODE FOR PROFILE MANAGEMENT

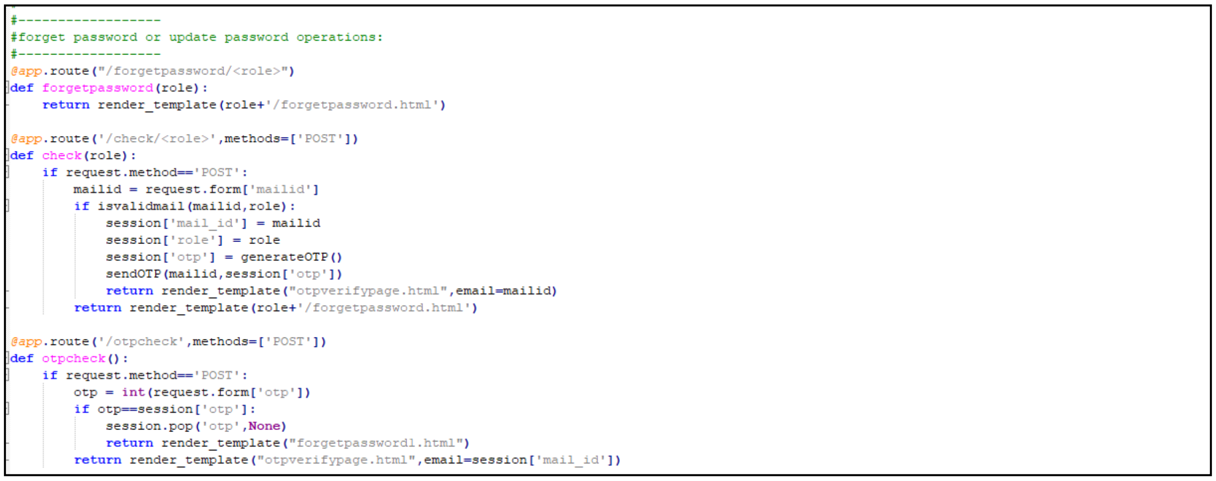


```python
#create new profile or update profile operations:
#------------------
@app.route('/create/<role>',methods=['POST'])
def createprofile(role):
    if request.method=='POST':
        data = request.form
        createprofile_function(role,data)
        if 'username' in session:
            list1 = getlist(role)
            return render_template(session['role']+"/"+role+"list.html",l = list1,name=session['username'])
        return render_template("index/"+role+"login.html")

@app.route('/updateprofile')
def updateprofile():
    return render_template('common/updateprofile.html',dt=session['data'])

@app.route('/update/<username>',methods=['POST'])
def update(username):
    if request.method=='POST':
        data = request.form
        updateprofile_function(username,data)
        if username == session['data'][4]:
            if session['role'] == 'admin':
                session['data'] = getdata(session['username'],'admin')
            else:
                session['data'] = getdata(username,'user')
            return render_template('common/viewprofile.html',dt=session['data'])
        else:
            if getrole(username) == 'admin':
                list1 = getlist('admin')
                return render_template("admin/adminlist.html",l = list1)
            else:
                list1 = getlist('user')
                return render_template("admin/userlist.html",l = list1)
```

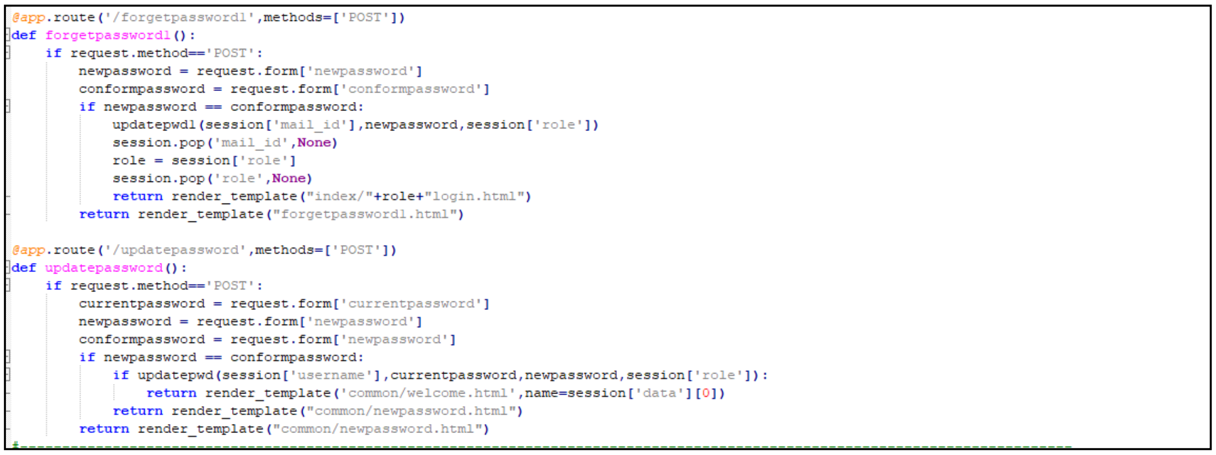


```python
#------------------
#edit or delete profile(admin/login) operations by admin:
#------------------
@app.route('/deleteuser/<mail_id>')
def delete(mail_id):
    delete_account(mail_id)
    list1 = getlist('user')
    return render_template("admin/userlist.html",l = list1)

@app.route('/edituser/<mail_id>')
def edit(mail_id):
    return render_template('common/updateprofile.html',dt=getdata(mail_id,'user'))

@app.route('/editadmin/<mail_id>')
def edit1(mail_id):
    return render_template('common/updateprofile.html',dt=getdata(mail_id,'admin0'))
```

CODE FOR CHANGE OR UPDATE PASSWORD

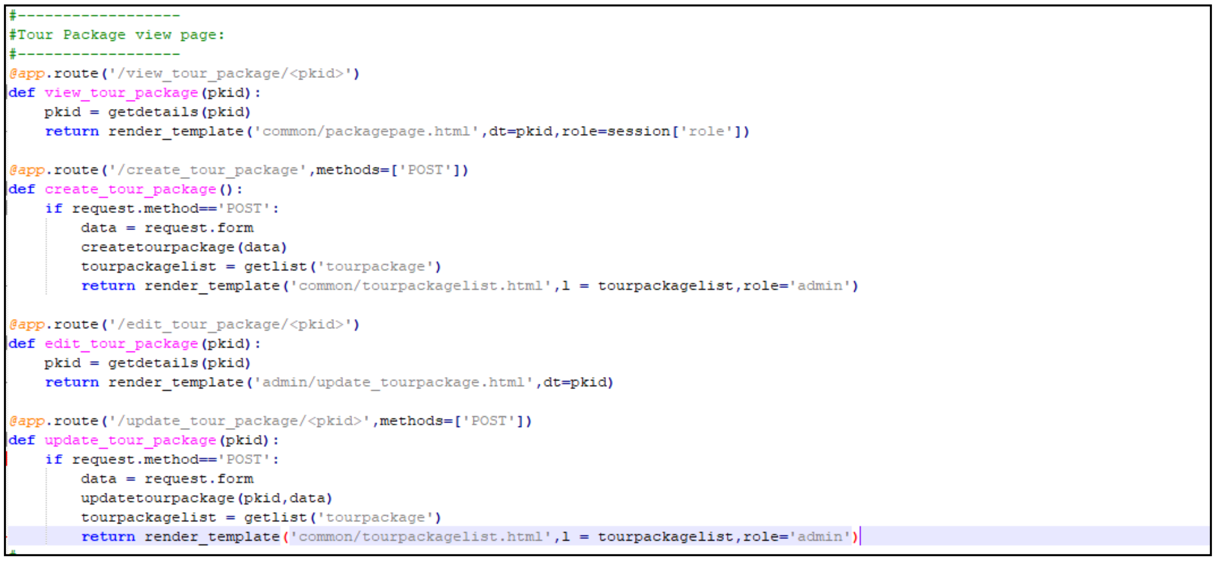


```python
#------------------
#forget password or update password operations:
#------------------
@app.route("/forgetpassword/<role>")
def forgetpassword(role):
    return render_template(role+'/forgetpassword.html')

@app.route('/check/<role>',methods=['POST'])
def check(role):
    if request.method=='POST':
        mailid = request.form['mailid']
        if isvalidmail(mailid,role):
            session['mail_id'] = mailid
            session['role'] = role
            session['otp'] = generateOTP()
            sendOTP(mailid,session['otp'])
            return render_template("otpverifypage.html",email=mailid)
        return render_template(role+'/forgetpassword.html')

@app.route('/otpcheck',methods=['POST'])
def otpcheck():
    if request.method=='POST':
        otp = int(request.form['otp'])
        if otp==session['otp']:
            session.pop('otp',None)
            return render_template("forgetpassword1.html")
        return render_template("otpverifypage.html",email=session['mail_id'])
```



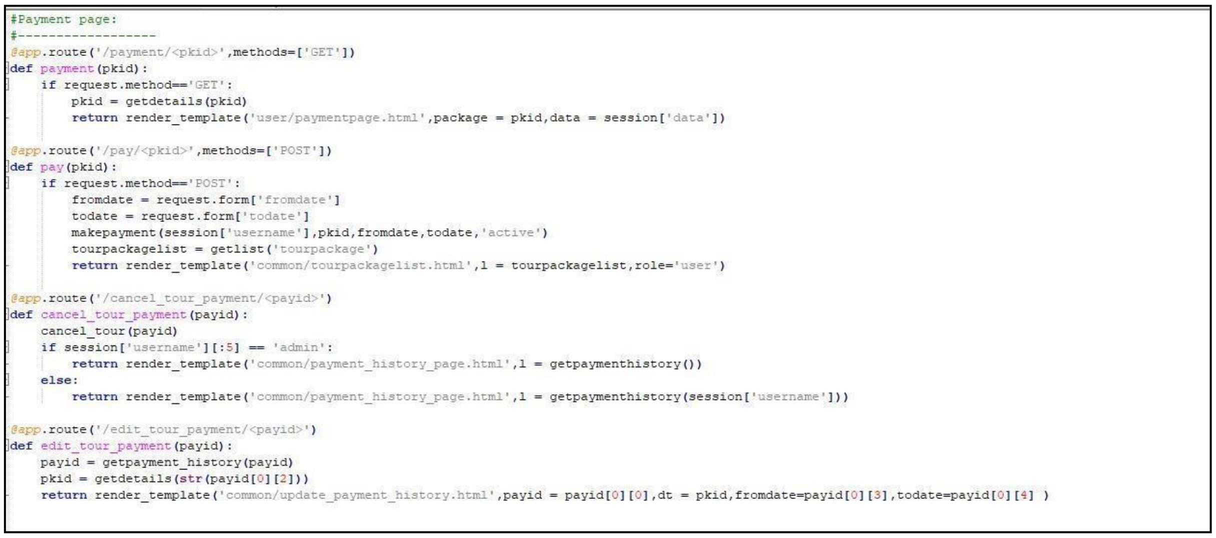
```python
@app.route('/forgetpassword1',methods=['POST'])
def forgetpassword1():
    if request.method=='POST':
        newpassword = request.form['newpassword']
        conformpassword = request.form['conformpassword']
        if newpassword == conformpassword:
            updatepwd1(session['mail_id'],newpassword,session['role'])
            session.pop('mail_id',None)
            role = session['role']
            session.pop('role',None)
            return render_template("index/"+role+"login.html")
        return render_template("forgetpassword1.html")

@app.route('/updatepassword',methods=['POST'])
def updatepassword():
    if request.method=='POST':
        currentpassword = request.form['currentpassword']
        newpassword = request.form['newpassword']
        conformpassword = request.form['newpassword']
        if newpassword == conformpassword:
            if updatepwd(session['username'],currentpassword,newpassword,session['role']):
                return render_template('common/welcome.html',name=session['data'][0])
            return render_template("common/newpassword.html")
        return render_template("common/newpassword.html")
#------------------------------------------------------------------------------------------------
```

CODE FOR PACKAGE MANAGEMENT

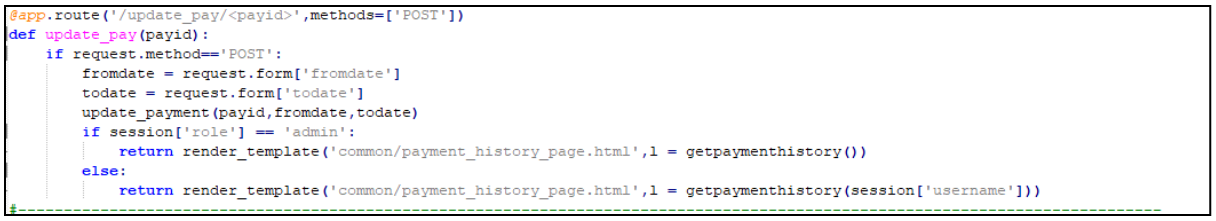


```
#------------------
#Tour Package view page:
#------------------
@app.route('/view_tour_package/<pkid>')
def view_tour_package(pkid):
    pkid = getdetails(pkid)
    return render_template('common/packagepage.html',dt=pkid,role=session['role'])

@app.route('/create_tour_package',methods=['POST'])
def create_tour_package():
    if request.method=='POST':
        data = request.form
        createtourpackage(data)
        tourpackagelist = getlist('tourpackage')
        return render_template('common/tourpackagelist.html',l = tourpackagelist,role='admin')

@app.route('/edit_tour_package/<pkid>')
def edit_tour_package(pkid):
    pkid = getdetails(pkid)
    return render_template('admin/update_tourpackage.html',dt=pkid)

@app.route('/update_tour_package/<pkid>',methods=['POST'])
def update_tour_package(pkid):
    if request.method=='POST':
        data = request.form
        updatetourpackage(pkid,data)
        tourpackagelist = getlist('tourpackage')
        return render_template('common/tourpackagelist.html',l = tourpackagelist,role='admin')
```

CODE FOR PAYMENT OPERATIONS



```
#Payment page:
#------------------
@app.route('/payment/<pkid>',methods=['GET'])
def payment(pkid):
    if request.method=='GET':
        pkid = getdetails(pkid)
        return render_template('user/paymentpage.html',package = pkid,data = session['data'])

@app.route('/pay/<pkid>',methods=['POST'])
def pay(pkid):
    if request.method=='POST':
        fromdate = request.form['fromdate']
        todate = request.form['todate']
        makepayment(session['username'],pkid,fromdate,todate,'active')
        tourpackagelist = getlist('tourpackage')
        return render_template('common/tourpackagelist.html',l = tourpackagelist,role='user')

@app.route('/cancel_tour_payment/<payid>')
def cancel_tour_payment(payid):
    cancel_tour(payid)
    if session['username'][:5] == 'admin':
        return render_template('common/payment_history_page.html',l = getpaymenthistory())
    else:
        return render_template('common/payment_history_page.html',l = getpaymenthistory(session['username']))

@app.route('/edit_tour_payment/<payid>')
def edit_tour_payment(payid):
    payid = getpayment_history(payid)
    pkid = getdetails(str(payid[0][2]))
    return render_template('common/update_payment_history.html',payid = payid[0][0],dt = pkid,fromdate=payid[0][3],todate=payid[0][4] )
```

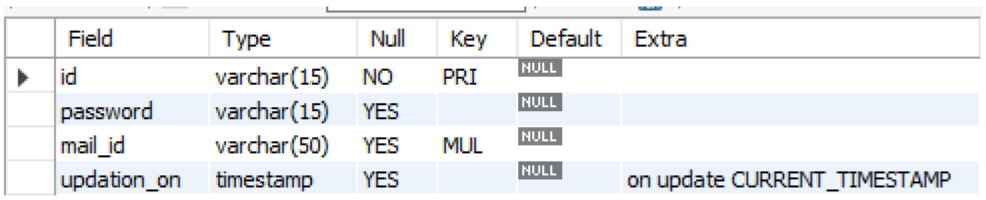


```
@app.route('/update_pay/<payid>',methods=['POST'])
def update_pay(payid):
    if request.method=='POST':
        fromdate = request.form['fromdate']
        todate = request.form['todate']
        update_payment(payid,fromdate,todate)
        if session['role'] == 'admin':
            return render_template('common/payment_history_page.html',l = getpaymenthistory())
        else:
            return render_template('common/payment_history_page.html',l = getpaymenthistory(session['username']))
#------------------------------------------------------------------------------------------------------------------------------
```

DATABASE TABLES CREATED IN APPLICATION

ADMIN TABLE

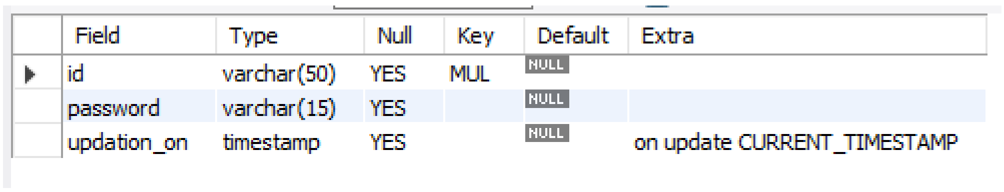
ADMIN Table stores details of admins.



Field	Type	Null	Key	Default	Extra
id	varchar(15)	NO	PRI	NULL	
password	varchar(15)	YES		NULL	
mail_id	varchar(50)	YES	MUL	NULL	
updation_on	timestamp	YES		NULL	on update CURRENT_TIMESTAMP

USER TABLE

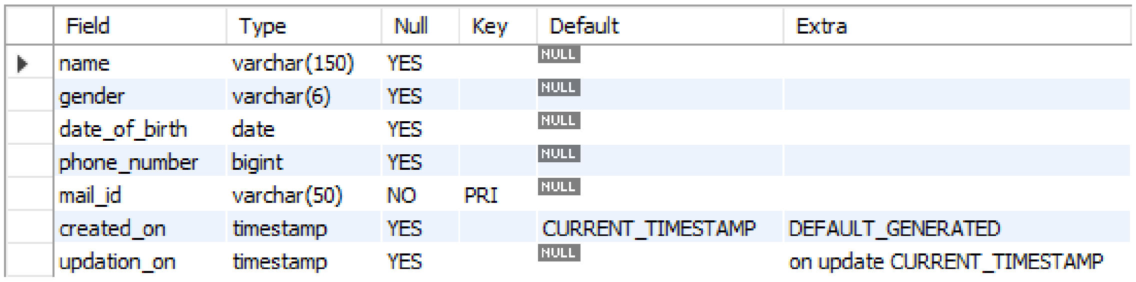
USER Table stores details of users.



Field	Type	Null	Key	Default	Extra
id	varchar(50)	YES	MUL	NULL	
password	varchar(15)	YES		NULL	
updation_on	timestamp	YES		NULL	on update CURRENT_TIMESTAMP

PERSON TABLE

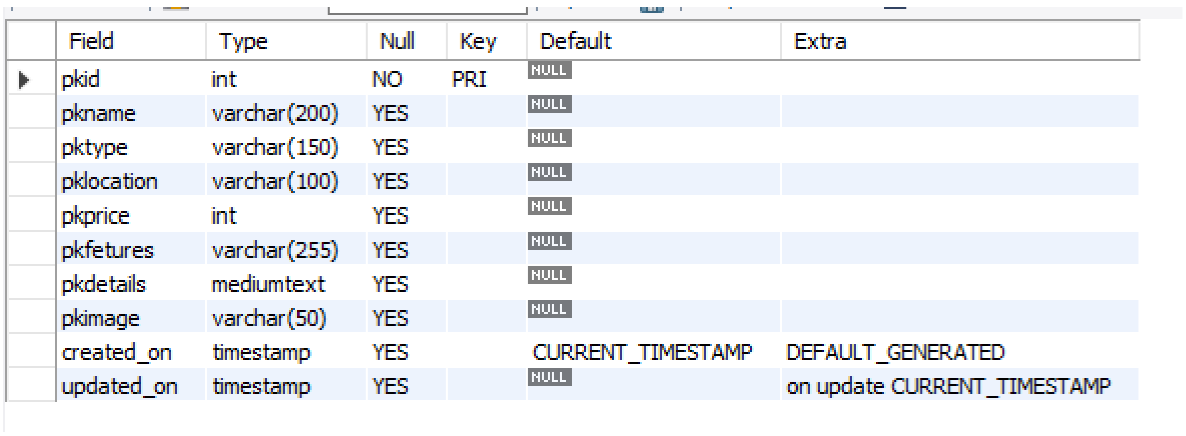
PERSON Table stores personal details of admin and user.



Field	Type	Null	Key	Default	Extra
name	varchar(150)	YES		NULL	
gender	varchar(6)	YES		NULL	
date_of_birth	date	YES		NULL	
phone_number	bigint	YES		NULL	
mail_id	varchar(50)	NO	PRI	NULL	
created_on	timestamp	YES		CURRENT_TIMESTAMP	DEFAULT_GENERATED
updation_on	timestamp	YES		NULL	on update CURRENT_TIMESTAMP

TOUR PACKAGES

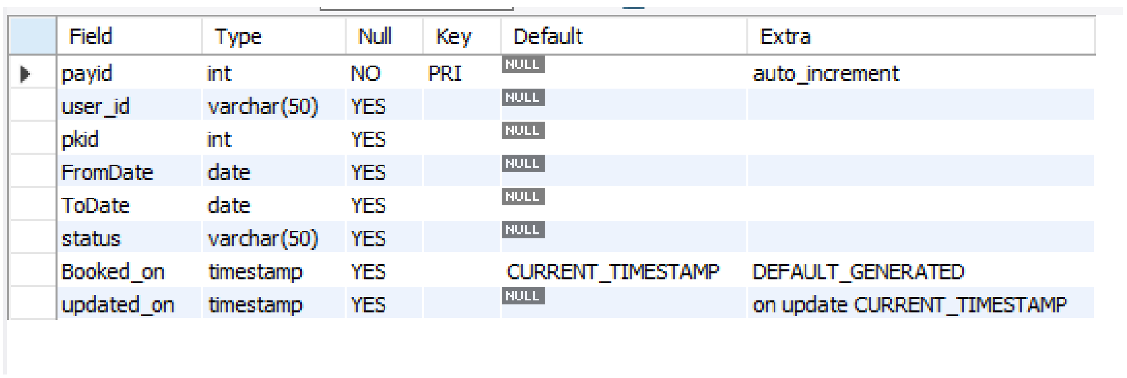
TOUR PACKAGES Table stores details of different tour packages.



Field	Type	Null	Key	Default	Extra
pkid	int	NO	PRI	NULL	
pkname	varchar(200)	YES		NULL	
pktype	varchar(150)	YES		NULL	
pklocation	varchar(100)	YES		NULL	
pkprice	int	YES		NULL	
pkfetures	varchar(255)	YES		NULL	
pkdetails	mediumtext	YES		NULL	
pkimage	varchar(50)	YES		NULL	
created_on	timestamp	YES		CURRENT_TIMESTAMP	DEFAULT_GENERATED
updated_on	timestamp	YES		NULL	on update CURRENT_TIMESTAMP

PAYMENT

PAYMENT Table stores the details of user , tour package, from date and end date.



Field	Type	Null	Key	Default	Extra
payid	int	NO	PRI	NULL	auto_increment
user_id	varchar(50)	YES		NULL	
pkid	int	YES		NULL	
FromDate	date	YES		NULL	
ToDate	date	YES		NULL	
status	varchar(50)	YES		NULL	
Booked_on	timestamp	YES		CURRENT_TIMESTAMP	DEFAULT_GENERATED
updated_on	timestamp	YES		NULL	on update CURRENT_TIMESTAMP

15. TESTING

The main purpose of testing FRAMS is to ensure that all the activities and functionalities of this software run smoothly with no errors and it remains protected.

FOR ADMIN :

- Verify admin login with valid and invalid data
- Verify admin view profile
- Verify admin update profile with valid and invalid data
- Verify admin change password with valid and invalid data
- Verify admin view tour packages
- Verify admin manage tour packages with valid and invalid data

FOR USER :

- Verify user registration with valid and invalid data.
- Verify user login with valid and invalid data
- Verify user view profile.
- Verify user update profile with valid and invalid data.
- Verify user change password with valid and invalid data.
- Verify user view tour packages
- Verify user book the tour package with valid and invalid data
- Verify user make payment with valid and invalid data.
- Verify user view invoices of payment

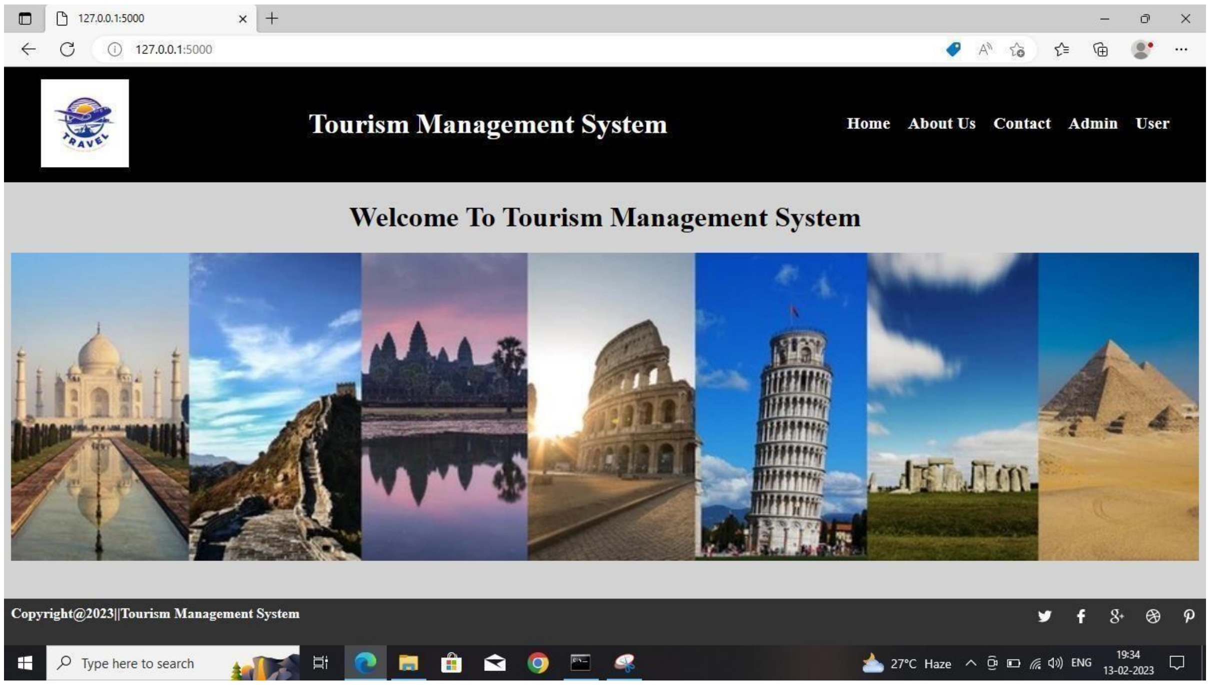
Test case no	**Functionality to be checked**	**Actual input**	**Actual output**	**Expected output**	**Status**
1	Verify admin login/verify user login	Given valid email id& valid password	Login succes s	Login success	Pass
		Given valid email id& invalid password	Deny login	Deny login	Pass
		Given invalid emailid & valid password	Deny login	Deny login	Pass
		Given invalid emailid & invalid password	Deny login	Deny login	Pass
2	Verify admin update profile/verify user updateprofile	Given new email id, new phone no, newdate of birth	updated	updated	Pass
		Given only one field	updated	updated	Pass
		Given only two fields	updated	updated	Pass
3	Verify admin change password/ verify userchange password	Given valid previouspassword, valid new password, correct confirm password	changed	Changed	Pass
		Given invalid previous password,valid new password, correct confirm password	Not change d	Not changed	Pass
		Given valid previouspassword, new password, wrong confirm password	Not change d	Not changed	Pass

4.	Verify user book tourpackage	Given valid fromdate and valid todate	Display tour package payment invoice	Display tour package payment invoice	Pass
		Given invalid fromdate and invalid todate	Deny payment	Deny payment	Pass

16. IMPLEMENTATION SCREEN SHOTS

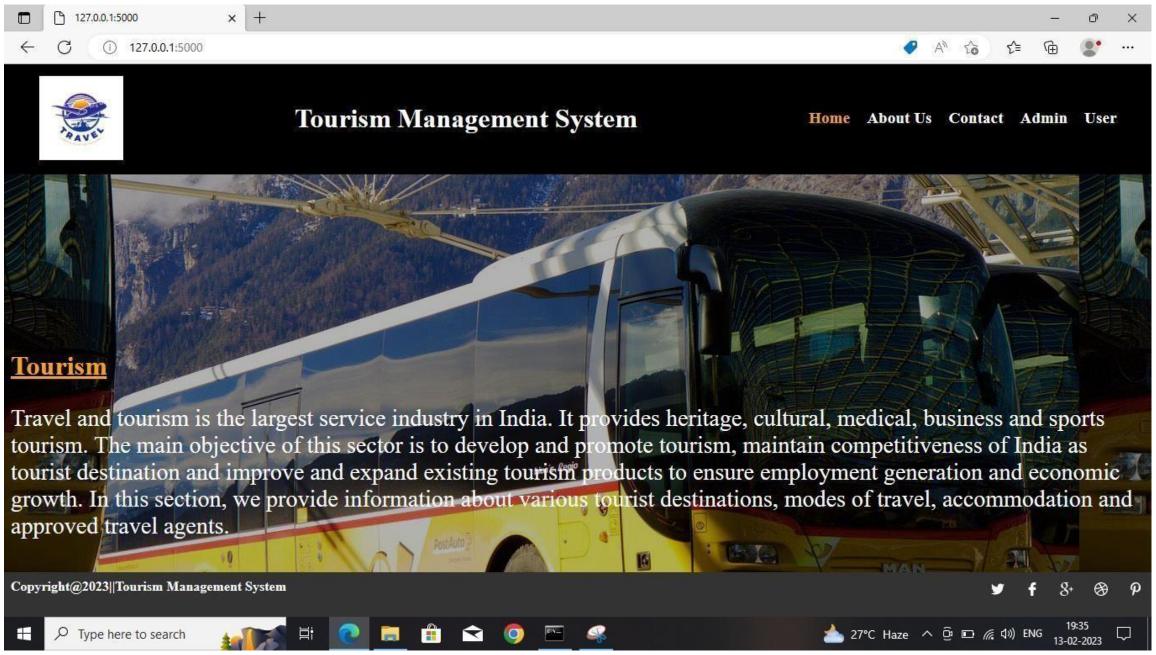
The following are runtime GUI developed in the application along with functionality

INDEX PAGE



127.0.0.1:5000
Tourism Management System
Home
About Us
Contact
Admin
User
Welcome To Tourism Management System
Copyright@2023||Tourism Management System
Type here to search
27°C Haze
ENG
19:34
13-02-2023

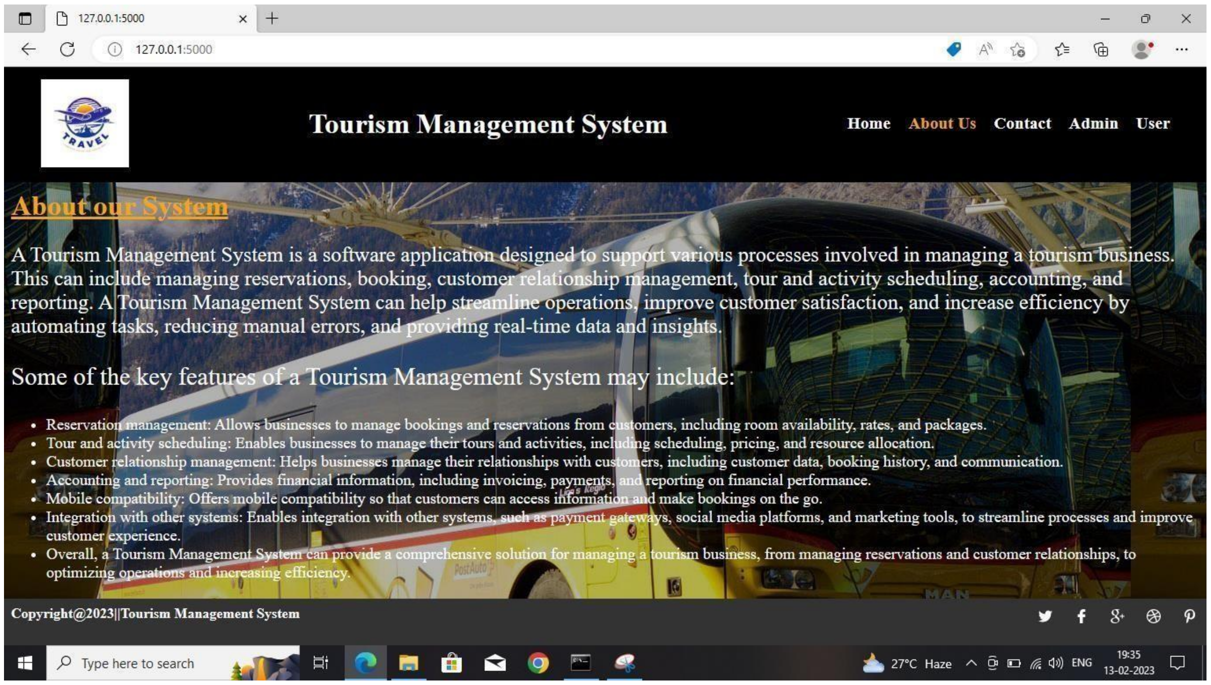
HOME PAGE



Tourism

Travel and tourism is the largest service industry in India. It provides heritage, cultural, medical, business and sports tourism. The main objective of this sector is to develop and promote tourism, maintain competitiveness of India as tourist destination and improve and expand existing tourism products to ensure employment generation and economic growth. In this section, we provide information about various tourist destinations, modes of travel, accommodation and approved travel agents.

ABOUT US PAGE



About our System

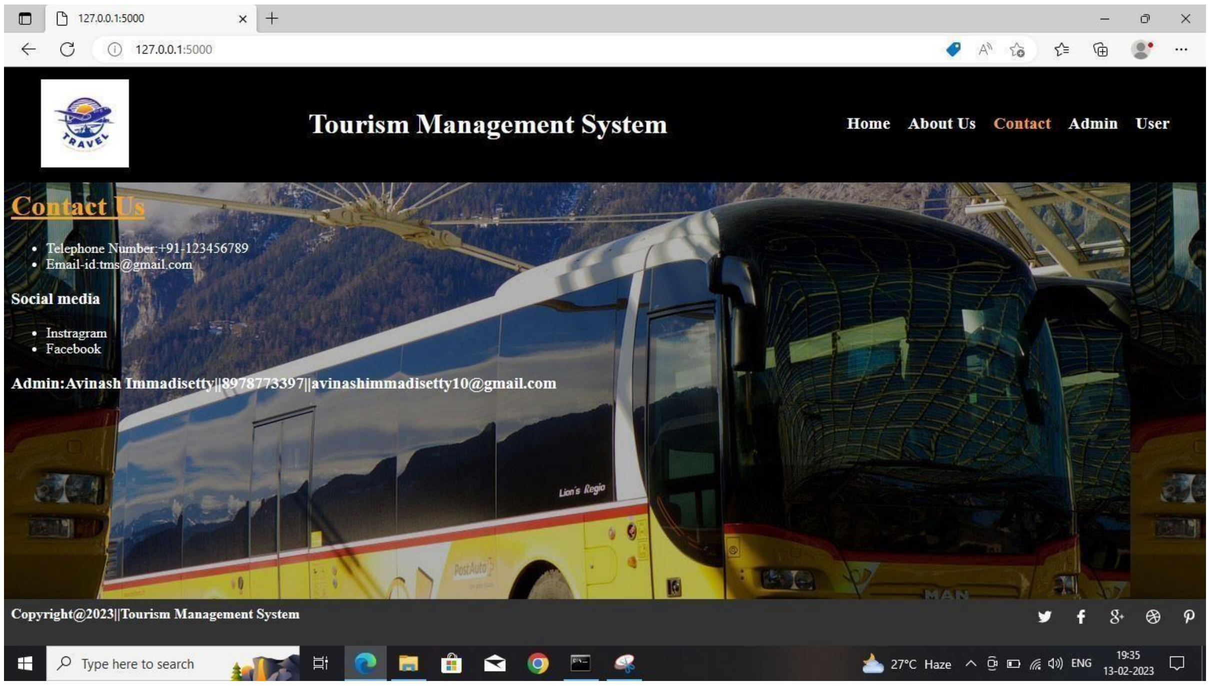
A Tourism Management System is a software application designed to support various processes involved in managing a tourism business. This can include managing reservations, booking, customer relationship management, tour and activity scheduling, accounting, and reporting. A Tourism Management System can help streamline operations, improve customer satisfaction, and increase efficiency by automating tasks, reducing manual errors, and providing real-time data and insights.

Some of the key features of a Tourism Management System may include:

- Reservation management: Allows businesses to manage bookings and reservations from customers, including room availability, rates, and packages.
- Tour and activity scheduling: Enables businesses to manage their tours and activities, including scheduling, pricing, and resource allocation.
- Customer relationship management: Helps businesses manage their relationships with customers, including customer data, booking history, and communication.
- Accounting and reporting: Provides financial information, including invoicing, payments, and reporting on financial performance.
- Mobile compatibility: Offers mobile compatibility so that customers can access information and make bookings on the go.
- Integration with other systems: Enables integration with other systems, such as payment gateways, social media platforms, and marketing tools, to streamline processes and improve customer experience.
- Overall, a Tourism Management System can provide a comprehensive solution for managing a tourism business, from managing reservations and customer relationships, to optimizing operations and increasing efficiency.

Copyright@2023||Tourism Management System

CONTACT PAGE



Contact Us

- Telephone Number:+91-123456789
- Email-id:tms@gmail.com

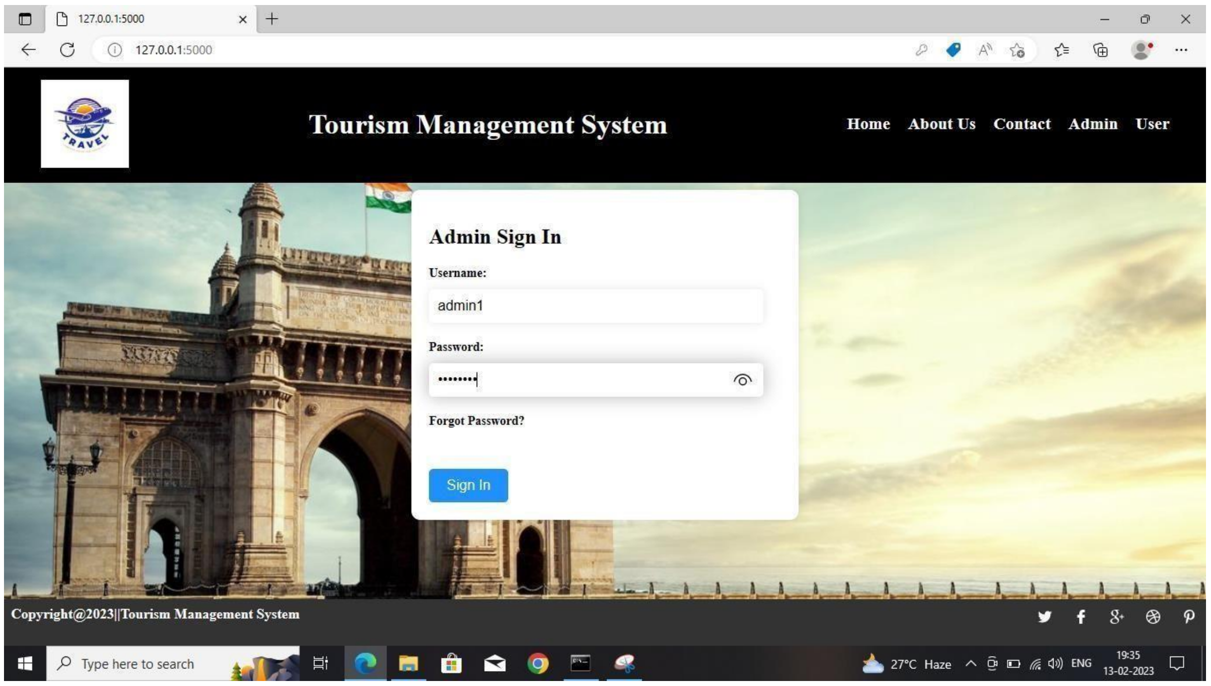
Social media

- Instragram
- Facebook

Admin:Avinash Immadisetty||8978773397||avinashimmadisetty10@gmail.com

Copyright@2023||Tourism Management System

ADMIN/USER SIGN IN PAGE



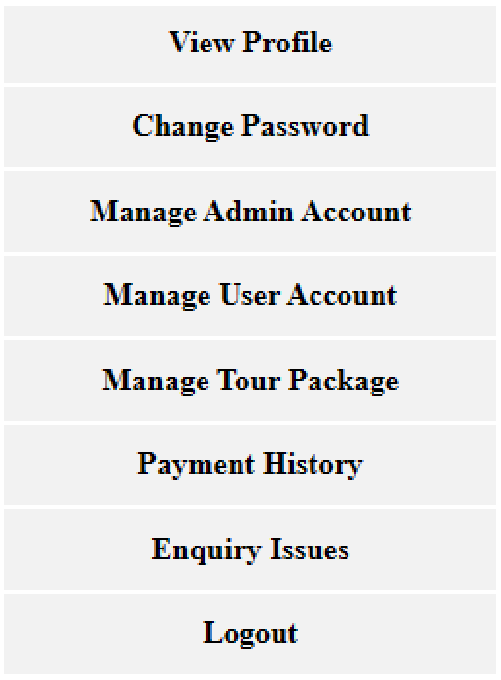
Admin Sign In

Username: admin1

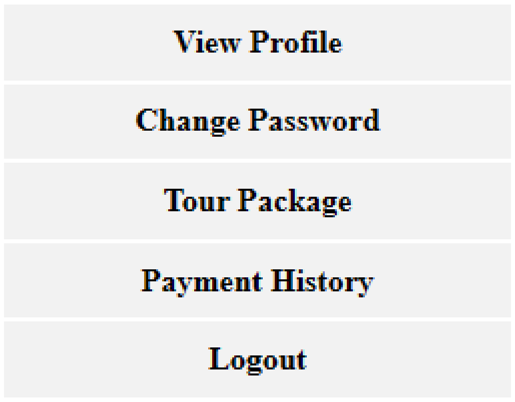
Password: ••••••••

Forgot Password?

ADMIN SERVICES

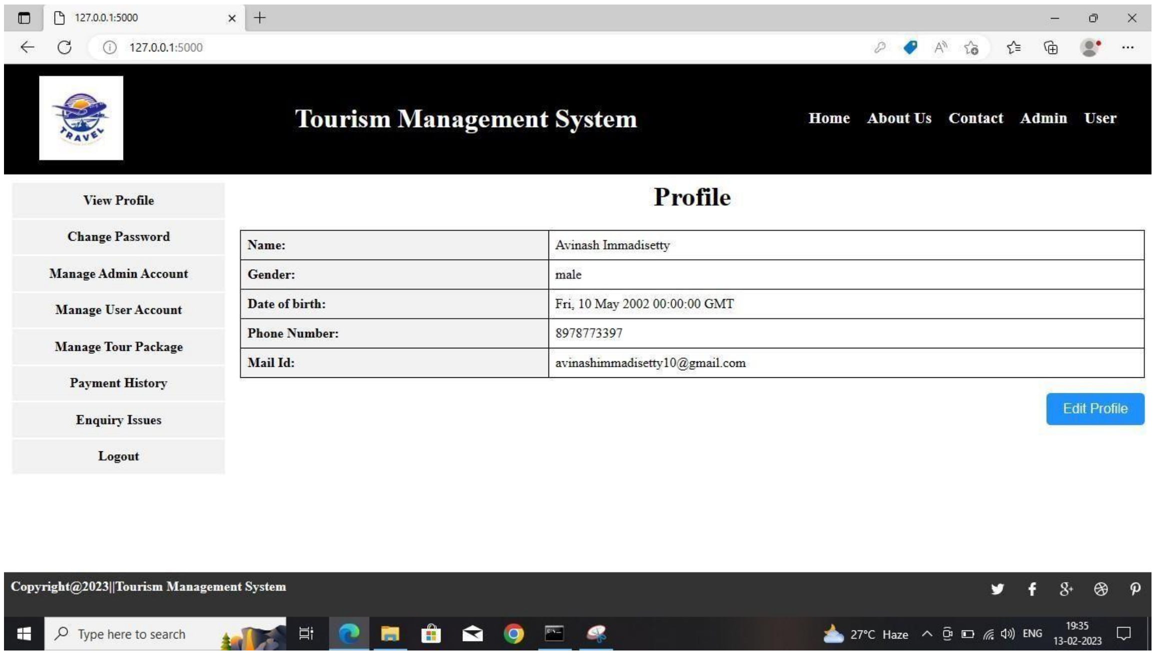


USER SERVICES



- View Profile
- Change Password
- Tour Package
- Payment History
- Logout

VIEW PROFILE PAGE

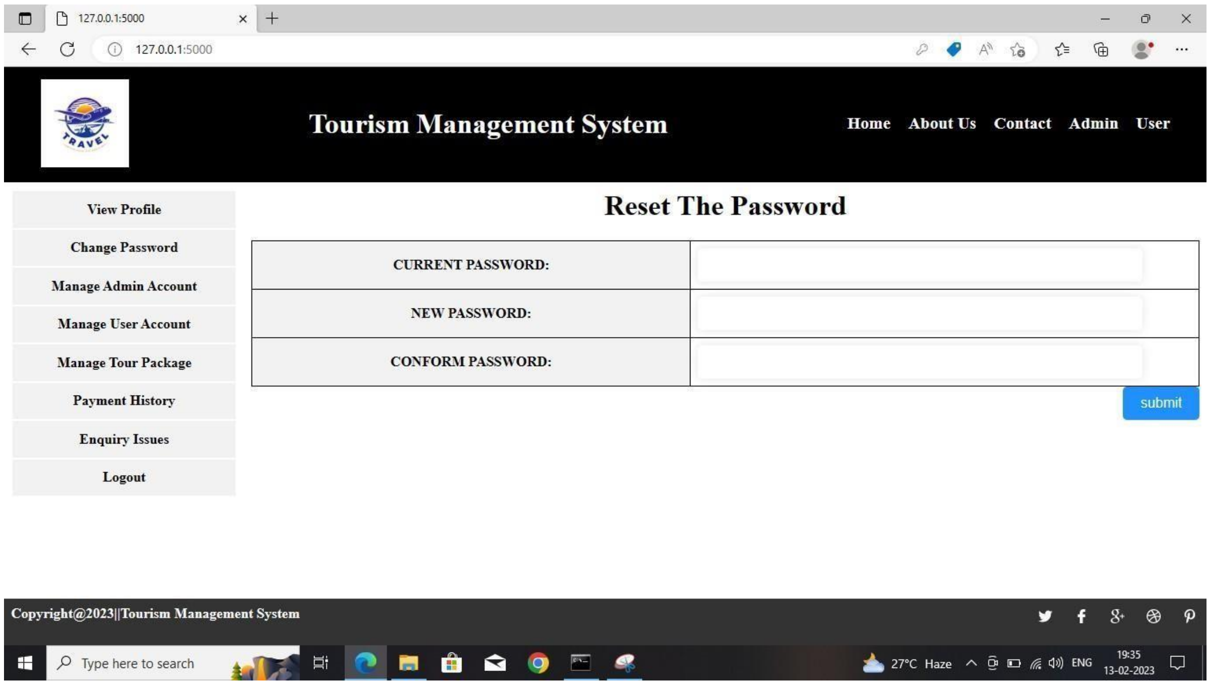


Profile

Name:	Avinash Immadisetty
Gender:	male
Date of birth:	Fri, 10 May 2002 00:00:00 GMT
Phone Number:	8978773397
Mail Id:	avinashimmadisetty10@gmail.com

Copyright@2023||Tourism Management System

CHANGE PASSWORD PAGE



127.0.0.1:5000

Tourism Management System

Home About Us Contact Admin User

View Profile
Change Password
Manage Admin Account
Manage User Account
Manage Tour Package
Payment History
Enquiry Issues
Logout

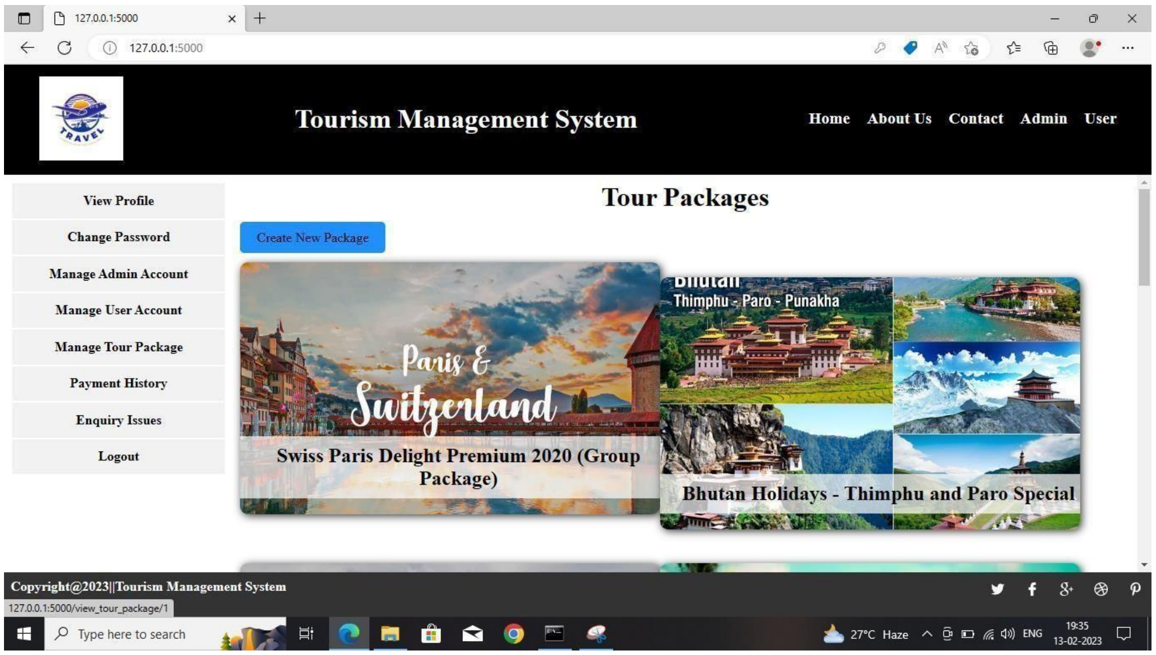
Reset The Password

CURRENT PASSWORD:	
NEW PASSWORD:	
CONFORM PASSWORD:	

submit

Copyright@2023||Tourism Management System

TOUR PACKAGE PAGE



127.0.0.1:5000

Tourism Management System

Home About Us Contact Admin User

View Profile
Change Password
Manage Admin Account
Manage User Account
Manage Tour Package
Payment History
Enquiry Issues
Logout

Tour Packages

Create New Package

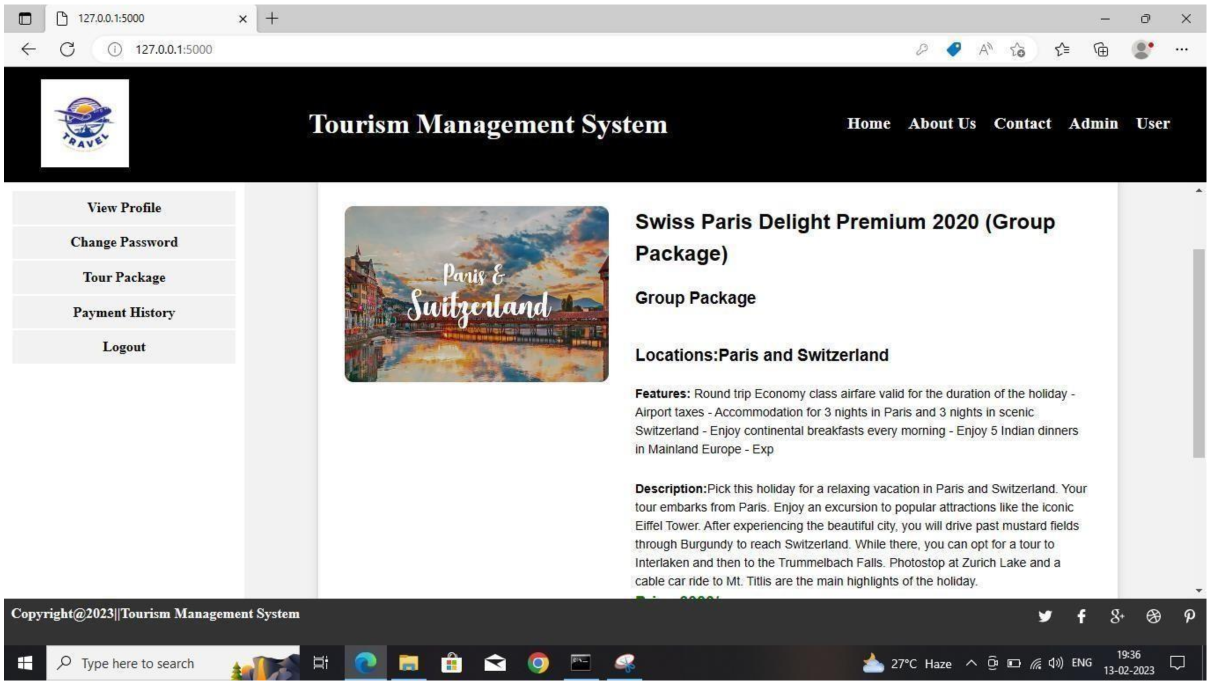
Swiss Paris Delight Premium 2020 (Group Package)

Bhutan Holidays - Thimphu and Paro Special

Copyright@2023||Tourism Management System

127.0.0.1:5000/view_tour_package/1

VIEW/BOOKING TOURPACKAGE PAGE



127.0.0.1:5000

Tourism Management System

Home About Us Contact Admin User

View Profile
Change Password
Tour Package
Payment History
Logout

Swiss Paris Delight Premium 2020 (Group Package)

Group Package

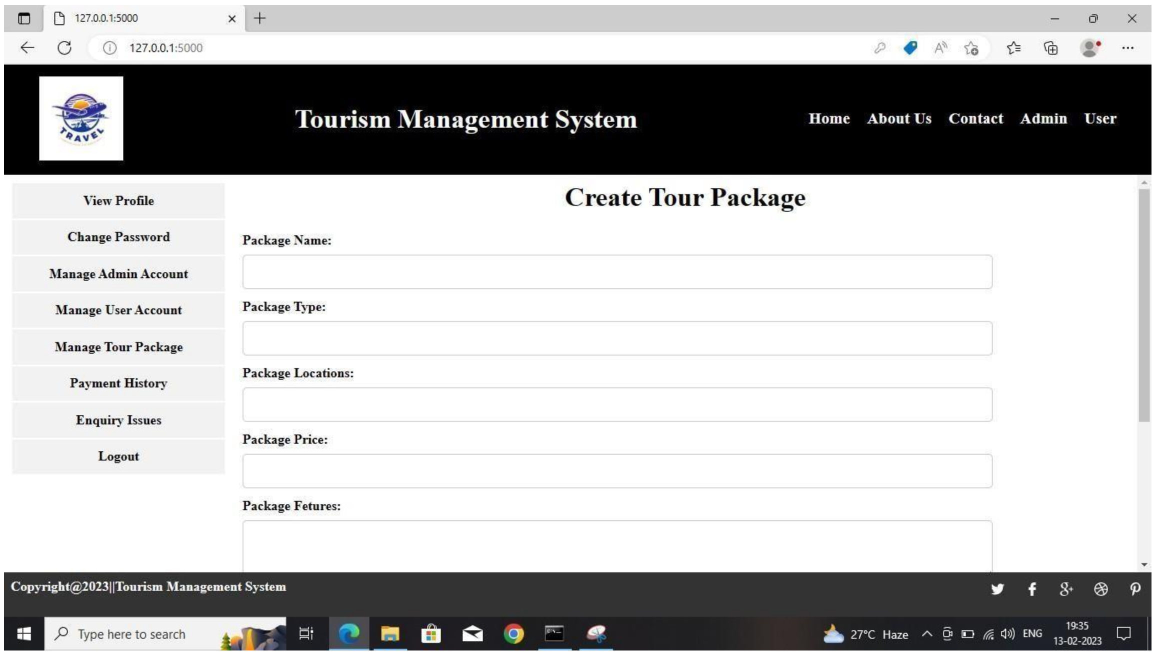
Locations:Paris and Switzerland

Features: Round trip Economy class airfare valid for the duration of the holiday - Airport taxes - Accommodation for 3 nights in Paris and 3 nights in scenic Switzerland - Enjoy continental breakfasts every morning - Enjoy 5 Indian dinners in Mainland Europe - Exp

Description: Pick this holiday for a relaxing vacation in Paris and Switzerland. Your tour embarks from Paris. Enjoy an excursion to popular attractions like the iconic Eiffel Tower. After experiencing the beautiful city, you will drive past mustard fields through Burgundy to reach Switzerland. While there, you can opt for a tour to Interlaken and then to the Trummelbach Falls. Photostop at Zurich Lake and a cable car ride to Mt. Titlis are the main highlights of the holiday.

Copyright@2023||Tourism Management System

CREATE NEW PACKAGE



127.0.0.1:5000

Tourism Management System

Home About Us Contact Admin User

View Profile
Change Password
Manage Admin Account
Manage User Account
Manage Tour Package
Payment History
Enquiry Issues
Logout

Create Tour Package

Package Name:

Package Type:

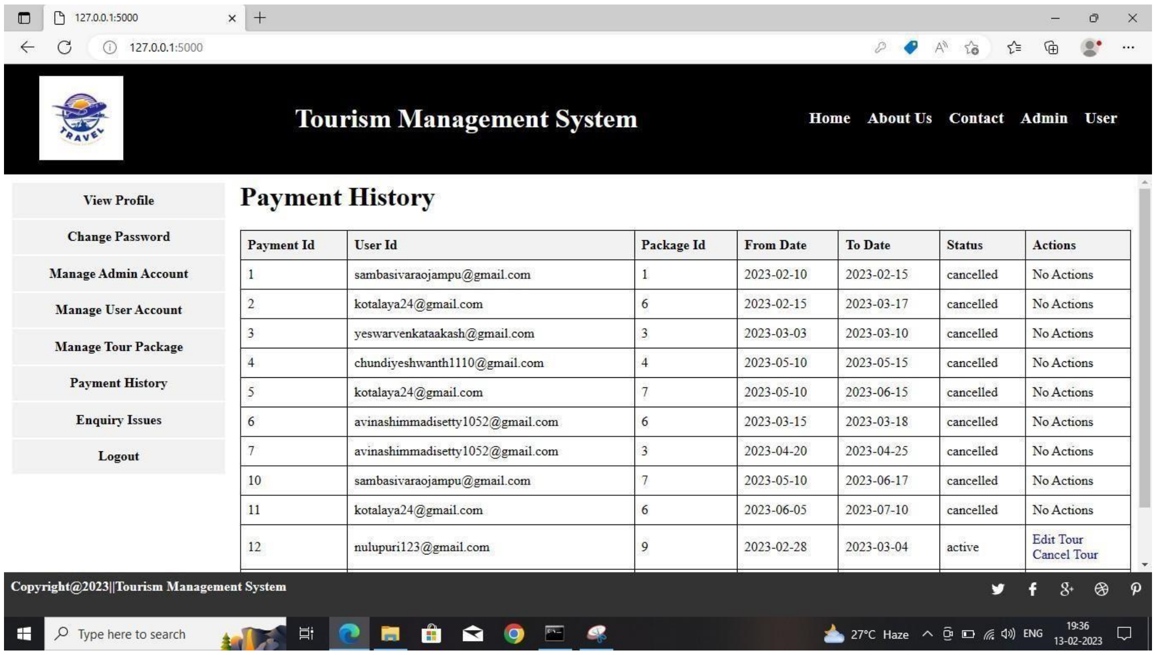
Package Locations:

Package Price:

Package Fetures:

Copyright@2023||Tourism Management System

PAYMENT HISTROY PAGE



Payment History

Payment Id	User Id	Package Id	From Date	To Date	Status	Actions
1	sambasivaraojampu@gmail.com	1	2023-02-10	2023-02-15	cancelled	No Actions
2	kotalaya24@gmail.com	6	2023-02-15	2023-03-17	cancelled	No Actions
3	yeswarvenkataakash@gmail.com	3	2023-03-03	2023-03-10	cancelled	No Actions
4	chundiyeshwanth1110@gmail.com	4	2023-05-10	2023-05-15	cancelled	No Actions
5	kotalaya24@gmail.com	7	2023-05-10	2023-06-15	cancelled	No Actions
6	avinashimmadisetty1052@gmail.com	6	2023-03-15	2023-03-18	cancelled	No Actions
7	avinashimmadisetty1052@gmail.com	3	2023-04-20	2023-04-25	cancelled	No Actions
10	sambasivaraojampu@gmail.com	7	2023-05-10	2023-06-17	cancelled	No Actions
11	kotalaya24@gmail.com	6	2023-06-05	2023-07-10	cancelled	No Actions
12	nulupuri123@gmail.com	9	2023-02-28	2023-03-04	active	Edit Tour Cancel Tour

17. CONCLUSION

TOURISM MANAGEMENT SYSTEM is the system that facilitates the user to book the tour package for their trip. This system saves time and energy for both user and admin. This proposed system can be developed in gives suggestion about tourism places that user can visit. The aim of the project entails the design and implementation of a platform that will assist tourists in gaining access to travel to various tourist locations. For a modified system, the user need to just login into the application and can find the routes, cost, hotels, transportations and book immediately and complete the booking process for a successful transaction.

REFERENCES

1. Harvey M. Deitel and Paul J.Deitel, "Internet & World Wide Web How to Program", 4/e, Pearson Education.
2. Jason Cranford Teague "Visual Quick Start Guide CSS, DHTML & AJAX", 4/ e, "Pearson Education".
3. Roger S. Pressman, Software Engineering - A Practitioner's Approach, Seventh Edition, McGraw Hill Publications.
4. Software Engineering Resources : - www.rspa.com/spi/ • Database Systems, Ramez Elmasri and Shamkant B.Navathe, Pearson Education, 6th edition.
5. James Rumbaugh, Jacobson, Booch, Unified Modeling Language Reference Manual, 2nd Edition, PHI.