Royal Education Society's

College of Computer Science and Information Technology, Latur.

Affiliated to

Swami Ramanand Teerth Marathwada University, Nanded.

A Project Report
on

Tour Package Booking System

Submitted for the award of degree of

Bachelor of Science in Software Engineering

By:

1. Gundre Renuka Dilip : 'A' 09
2. Mithagare Rushikesh Rajkumar: 'A'33

In

Year 2024 – 2025

MR. GOPAL SHINDE
Project Guide

MR.B.M.SONTAKKE
HOD

ACKNOWLEDGEMENT

We would like to convey our gratitude to **Dr. V. A. Jadhav**, Principal of **College of Computer Science and Information Technology, Latur** who gave us necessary information and guidance for project.

We are grateful to **Swami Ramanand Teerth Marathwada University, Nanded** for giving an opportunity to deliver project.

We would like to thank Project In-Charge and Project Guide **Mr. Gopal Shinde** who guided us through doing these project development process, provided with invaluable advice, helped us in difficult periods and provided practical assistant for our project. Their willingness to motivate us contributed tremendously to the success of this project.

We would like to express our special thanks of gratitude to our Head of the Department of Computer Science Department of Software Engineering **Mr. B. M. Sontakke sir** who helped us a lot in finalizing this project

Besides we would like to thank all staff members who helped us by giving advice and providing equipment which we needed.

Last but not in least we would like to thank all who helped and motivated us.

With Sincere Thanks,

1. Gundre Renuka Dilip : 'A' 09

2. Mithagare Rushikesh Rajkumar: 'A'33

Index

1. Abstract :

The **Tour Package Booking System** is an advanced e-commerce platform developed using Django to streamline the process of booking travel packages. It caters to both users and administrators, offering an intuitive interface to browse, select, and book tours based on various preferences like destination, budget, and package duration. This web-based platform digitizes the traditional tour booking process, providing users with an easy, secure way to explore tour packages, check real-time availability, and make payments from any location.

The system includes essential features such as user account management, package filtering, and booking history. On the administrative side, it supports package management, customer inquiry handling, and reporting tools for analyzing booking trends and user behavior. Administrators have the capability to manage tour packages efficiently by adding, modifying, and removing packages based on availability or demand. This ensures flexibility in managing offers and user interactions.

Built using modern web technologies, the system emphasizes security, scalability, and user-friendliness. Payment handling is integrated using secure payment gateways, ensuring that users can pay online with confidence. Furthermore, the system supports notifications for booking confirmations and updates, offering a seamless communication loop between users and the platform.

The primary objective of this system is to create an efficient, all-in-one solution for managing travel bookings. It reduces manual overhead for tour operators, enhances the customer experience, and allows for growth in functionality, such as integrating dynamic pricing or personalized travel recommendations in the future.

In summary, the **Tour Package Booking System** aims to enhance the way customers and operators interact by creating an optimized and secure platform for booking travel packages, leading to better operational efficiency and user satisfaction. This project serves as a scalable foundation for future enhancements, including mobile compatibility and third-party integrations

2 Introduction:

The **Tour Package Booking System** is a comprehensive web application designed to digitize the tour booking experience for both customers and tour operators. With the increasing demand for convenience and speed in today's digital world, this platform allows users to browse through various travel packages, check availability, and make bookings in a secure and hassle-free manner from any location. The system aims to address the limitations of traditional tour booking methods, which often involve manual processes, delayed responses, and a lack of transparency in pricing and availability.

This system provides users with access to detailed tour package information, including itinerary, pricing, destination, and duration, along with the option to customize their travel experience. It also integrates secure payment gateways to facilitate online transactions, making the entire booking process seamless. In addition to offering an exceptional user experience, the platform equips administrators with the tools to manage tour packages, bookings, and customer inquiries efficiently through an intuitive dashboard.

The project is built using Django, a powerful Python-based web framework that ensures rapid development, scalability, and security. For the front-end, technologies such as HTML, CSS, JavaScript, and AJAX have been employed to create a responsive and interactive user interface. The backend is powered by SQLite3, which serves as the database to store and manage user data, package details, and bookings. The system is designed to support future expansion, allowing for potential integration with third-party APIs, mobile applications, and advanced features like real-time price adjustments and personalized recommendations.

The primary objective of the **Tour Package Booking System** is to create a user-friendly, scalable solution that can handle the complexities of tour bookings and streamline operations for administrators. This platform not only offers convenience and ease of use but also enhances the overall efficiency of tour operators in managing customer data and bookings.

2.1 Project overview:

The **Tour Package Booking System** is a **Django-based web application** designed to provide an efficient platform for browsing, booking, and managing tour packages. This system allows users to explore various travel packages, make secure reservations, and track their bookings, while administrators can efficiently manage tour listings, monitor bookings, and generate insightful reports.
The project follows a **modular and scalable architecture**, ensuring easy maintenance and future enhancements, such as **payment gateway integration and database migration to PostgreSQL**. The system is **successfully deployed on PythonAnywhere**, making it accessible for demonstration and real-world usage.

System Architecture & Modules :

1. User Module

- **User Authentication**: Secure registration, login, and profile management using Django's built-in authentication system.

- **Tour Package Browsing**: Users can explore various tour packages, view itineraries, and check pricing.

- **Booking Management**: Users can book, cancel, and modify reservations, with a **history section** for tracking past bookings.

2. Admin Module

- **Tour Package Management**: Admins can **add, update, or remove tour packages** efficiently.

- **Booking & Inquiry Management**: Track and manage **user bookings, cancellations, and customer inquiries** in real time.

- **Reports & Insights**: Generate reports on **package popularity, booking trends, and customer engagement** for better decision-making.

3. Payment Module ***(Planned for Future Enhancement)***

- **Secure Payment Gateway Integration**: Future updates will include integration with **UPI, net banking, and card payments**.
- **Automated Booking Confirmations**: Once payment is processed, users will receive **real-time booking confirmations**.

4. Website Module

- **Frontend Development**: Built using **HTML, CSS, Bootstrap, and JavaScript** for a modern, responsive user experience.
- **Asynchronous Data Handling**: **AJAX** ensures smooth, dynamic content updates without reloading pages.
- **Static & Media Files Management**: Efficient handling of images, stylesheets, and scripts for improved performance.

5. Deployment Details

The **Tour Package Booking System** is successfully **deployed on PythonAnywhere**, making it accessible online for demonstration and testing.

🔗 **Live Demo:** *https://rushimithagare.pythonanywhere.com/*

Deployment Setup:

- **Hosting Platform:** PythonAnywhere (Cloud-based Python hosting service)
- **Backend Framework:** Django (Python)
- **Database Used:** SQLite *(Future migration planned to PostgreSQL for better scalability)*
- **Static & Media Files Handling:** Django's built-in static file management with PythonAnywhere's file storage
- **Accessibility:** The system is publicly accessible, allowing users to register, browse tour packages, and make bookings seamlessly

Future Enhancements:

- **Custom Domain Integration** instead of PythonAnywhere's default subdomain
- **Scalability Improvements** with PostgreSQL migration and load balancing
- **Automated Deployment Pipeline** using GitHub and PythonAnywhere's scheduled tasks

6. Technology Stack & Project Structure

Backend & Database

- **Framework:** Django (Python)
- **Database:** SQLite (default) *(Planned migration to PostgreSQL for scalability and reliability)*
- **Authentication:** Django's built-in user authentication system
- **Front-end:** HTML, CSS, Bootstrap, JavaScript

Project Structure :

```
├── cart/                    # Shopping cart management
├── media/                   # Stores uploaded files (tour images, user uploads)
│   ├── uploads/
│   ├── product/
├── payment/                 # Payment handling module (future enhancement)
├── static/                  # Static files (CSS, JavaScript, images)
├── staticfiles/             # Collected static files after running `collectstatic`
├── tour_package_booking/    # Main Django project directory
├── ecom/                    # Main eCommerce application
│   ├── migrations/          # Handles database migrations
│   ├── models.py            # Defines database schema for products, bookings, users
│   ├── views.py             # Application logic and request handling
│   ├── templates/           # Frontend templates (HTML)
│   ├── static/              # Additional static files for the eCommerce app
│   ├── staticFile/admin/css/  # Django-generated admin panel CSS
│   └── urls.py              # URL routing configuration
├── manage.py                # Django management script
├── requirements.txt         # List of Python dependencies
└── db.sqlite3               # SQLite database for development
```

2.2 Project Plan :

Gantt Chart:

Sr.no	Task Name	15- Sep	25-sep	10-Oct	25-Oct	15-Nov
1	Requirement Gathering					
2	Planning					
3	Designing					
4	Coding					
5	Testing and Deployment					

3. Project Requirements:

This section outlines the **hardware, software, and research requirements** for the successful development and deployment of the **Tour Package Booking System**.

3.1 Hardware Requirements

To ensure smooth development and execution, the following hardware specifications are recommended:

- **Processor:** Intel Core i5 or higher (or AMD equivalent)
- **RAM:** Minimum **8GB** (16GB recommended for optimal performance)
- **Storage:** At least **500GB HDD/SSD** *(SSD preferred for faster processing)*
- **Internet Connection:** Required for **online functionalities, deployment, and API integrations**

3.2 Software Requirements

The system requires the following software tools and technologies:

- **Operating System:** Windows 10/Linux/macOS *(Cross-platform compatibility ensured)*
- **Integrated Development Environment (IDE):** PyCharm / Visual Studio Code
- **Backend Framework:** Django (Python-based web framework)
- **Database:** SQLite *(Default, with future migration planned to PostgreSQL for scalability)*
- **Frontend Technologies:** HTML, CSS, JavaScript *(Bootstrap for responsive design)*

3.3 Field Work

Before development, thorough research was conducted to align the system with industry standards and user expectations:

- **Market Research:** Studied existing tour booking platforms to analyze their functionalities and identify improvements.
- **Requirement Gathering:** Collected input from potential users, including tourists, travel agencies, and business owners.
- **User Feedback & Analysis:** Evaluated user preferences and pain points to optimize the system's user experience.

3.4 Front-End Development

The **frontend** is designed to provide an **interactive and responsive user experience** using:

- **IDE Used:** Visual Studio Code
- **Programming Languages:** HTML, CSS, JavaScript

- **Styling & Design:** Bootstrap for responsive layouts and improved aesthetics
- **Asynchronous Features:** AJAX for seamless data updates without page reloads

3.5 Back-End Development & Database

The **backend** ensures **secure and efficient** data handling with the following technologies:

- **Backend Framework:** Django (Python) for server-side logic and data processing
- **Database:** SQLite (Lightweight and efficient for development; PostgreSQL migration planned for scalability)
- **Object-Relational Mapping (ORM):** Django ORM for efficient database interactions

3.6 Dependencies & Libraries :

The project uses the following **Python dependencies** for efficient backend functionality and database management:

Library	Version	Purpose
asgiref	3.8.1	ASGI compatibility for Django's async support
Django	5.1	Main web framework for development
django-debug-toolbar	4.4.6	Debugging tool for performance analysis
django-extensions	3.2.3	Provides additional management commands
pillow	11.0.0	Image processing library for handling media uploads
psycopg	3.2.1	PostgreSQL adapter for Django
psycopg-binary	3.2.1	Binary version of Psycopg for PostgreSQL connectivity
sqlparse	0.5.1	SQL parsing library for Django
typing_extensions	4.12.2	Provides backward compatibility for Python type hints
tzdata	2024.1	Timezone data for Django's time-aware features

These dependencies ensure **optimized performance, secure database handling, and advanced debugging features** during development and deployment.

4. System Design :

User profile schema:-

Column Name	Data Type	Constraints
id	INTEGER	PRIMARY KEY AUTOINCREMENT
user_id	INTEGER	UNIQUE, FOREIGN KEY REFERENCES auth_user (id) ON DELETE CASCADE
date_modified	DATETIME	NOT NULL
phone	VARCHAR(20)	DEFAULT ''
address1	VARCHAR(200)	DEFAULT ''
address2	VARCHAR(200)	DEFAULT ''
city	VARCHAR(200)	DEFAULT ''
state	VARCHAR(200)	DEFAULT ''
zipcode	VARCHAR(200)	DEFAULT ''
country	VARCHAR(200)	DEFAULT ''
old_cart	VARCHAR(200)	DEFAULT ''

Category schema:

Column I	Data Type	Constraints
id	INTEGER	PRIMARY KEY AUTOINCREMENT
name	VARCHAR(100)	NOT NULL

Customer schema :-

Column Name	Data Type	Constraints
id	INTEGER	PRIMARY KEY AUTOINCREMENT
first_name	VARCHAR(50)	NOT NULL
last_name	VARCHAR(50)	NOT NULL
phone	VARCHAR(10)	NOT NULL
email	VARCHAR(100)	NOT NULL
password	VARCHAR(100)	NOT NULL

Product schema :-

Column Name	Data Type	Constraints
id	INTEGER	PRIMARY KEY AUTOINCREMENT
name	VARCHAR(50)	NOT NULL
price	DECIMAL(10, 2)	NOT NULL DEFAULT 0
category_id	INTEGER	NOT NULL DEFAULT 1, FOREIGN KEY REFERENCES category (id) ON DELETE CASCADE
description	VARCHAR(250)	DEFAULT ''
image	VARCHAR(100)	NOT NULL
is_sale	BOOLEAN	NOT NULL DEFAULT 0
sale_price	DECIMAL(10, 2)	NOT NULL DEFAULT 0

Order schema :-

Column Name	Data Type	Constraints
id	INTEGER	PRIMARY KEY AUTOINCREMENT
product_id	INTEGER	NOT NULL, FOREIGN KEY REFERENCES product (id) ON DELETE CASCADE
customer_id	INTEGER	NOT NULL, FOREIGN KEY REFERENCES customer (id) ON DELETE CASCADE
quantity	INTEGER	NOT NULL DEFAULT 1
address	VARCHAR(100)	DEFAULT ''
phone	VARCHAR(10)	DEFAULT ''
date	DATE	NOT NULL DEFAULT CURRENT_DATE
status	BOOLEAN	NOT NULL DEFAULT 0

Shipping Address schema(Payment module) :-

Column Name	Data Type	Constraints
id	INTEGER	PRIMARY KEY AUTOINCREMENT
user_id	INTEGER	FOREIGN KEY REFERENCES auth_user (id) ON DELETE CASCADE
shipping_full_name	VARCHAR(255)	NOT NULL
shipping_email	VARCHAR(255)	NOT NULL
shipping_address1	VARCHAR(255)	NOT NULL
shipping_address2	VARCHAR(255)	
shipping_city	VARCHAR(255)	NOT NULL
shipping_state	VARCHAR(255)	
shipping_zipcode	VARCHAR(255)	
shipping_country	VARCHAR(255)	NOT NULL

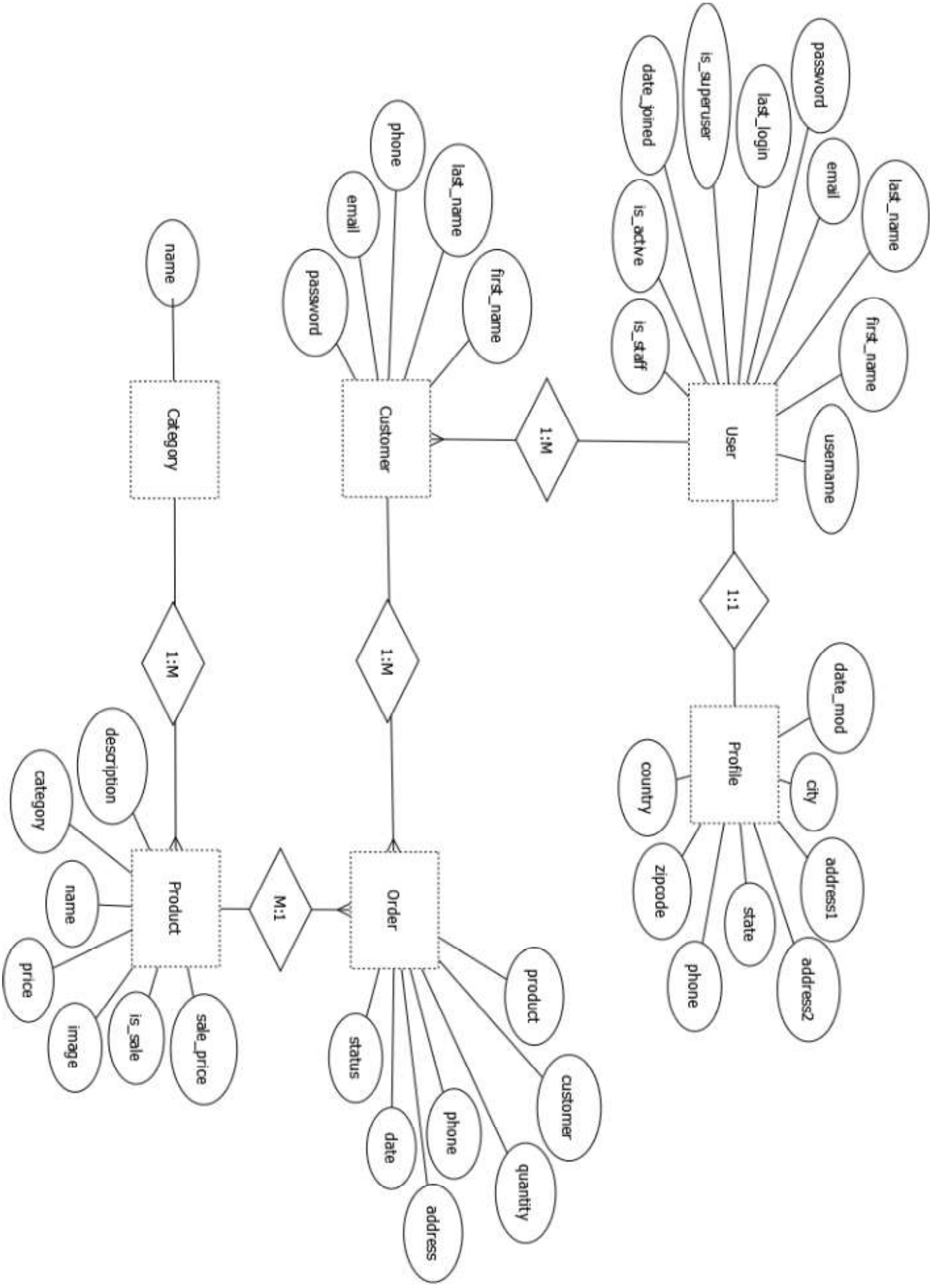
Order Schema (Payment module) : -

Column Name	Data Type	Constraints
id	INTEGER	PRIMARY KEY AUTOINCREMENT
user_id	INTEGER	FOREIGN KEY REFERENCES auth_user (id) ON DELETE CASCADE
full_name	VARCHAR(250)	NOT NULL
email	VARCHAR(250)	NOT NULL
shipping_address	VARCHAR(10000)	NOT NULL
amount_paid	DECIMAL(10, 2)	NOT NULL
date_ordered	DATETIME	NOT NULL DEFAULT CURRENT_TIMESTAMP

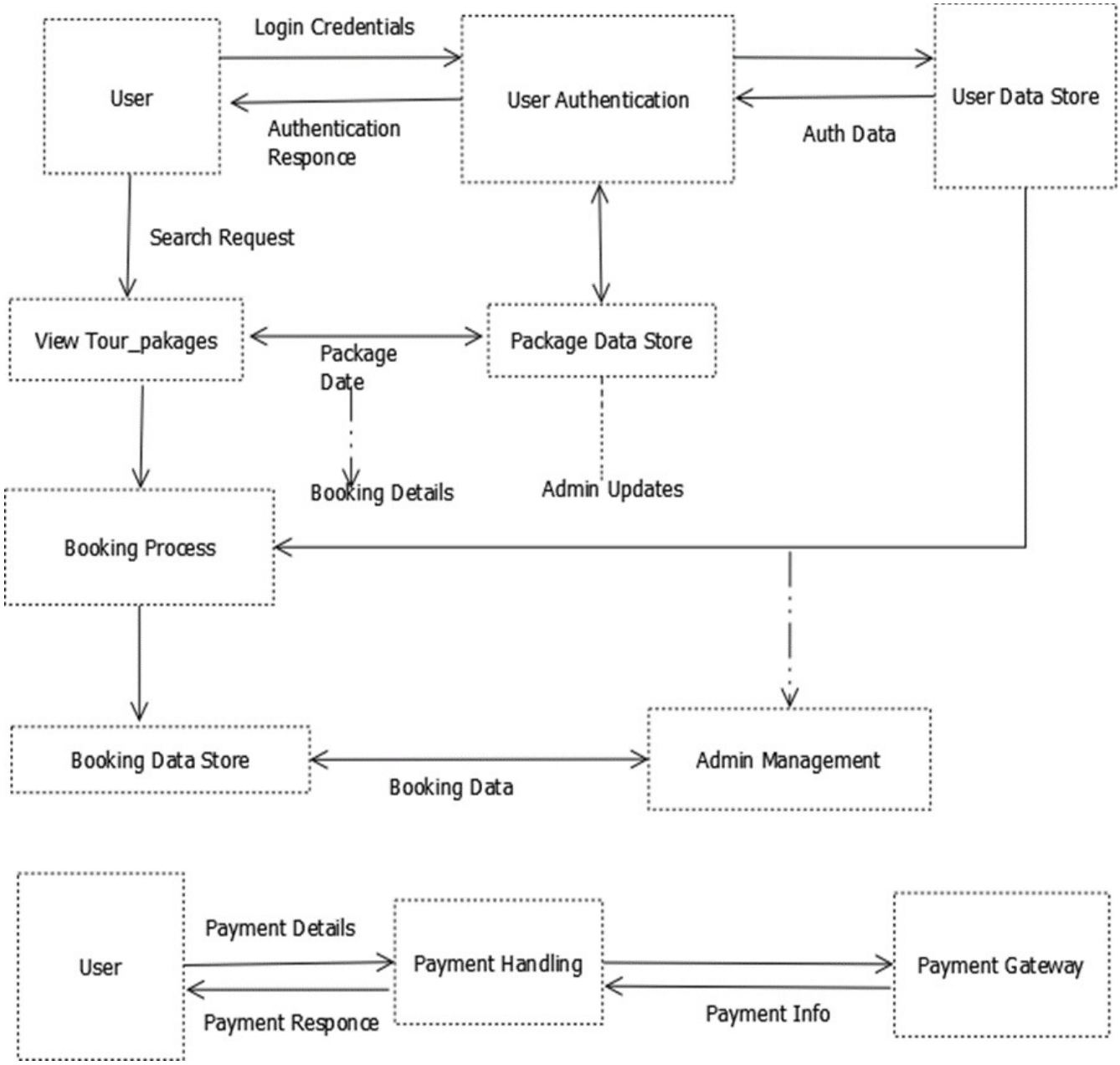
order item (Payment module):-

Column Name	Data Type	Constraints
id	INTEGER	PRIMARY KEY AUTOINCREMENT
order_id	INTEGER	FOREIGN KEY

4.1. E-R Diagram:



4.2 Data Flow Diagram (First Level)



Level 1 DFD

Processes:

1. User Authentication
   Checks user credentials and grants access.
2. View Tour Packages
   Shows available packages to users.
3. Booking Process
   Handles the actual booking of a tour package.
4. Payment Handling
   Manages payment flow between the user and payment gateway.
5. Admin Management
   Allows the admin to update package details, view bookings, etc.

Data Stores:

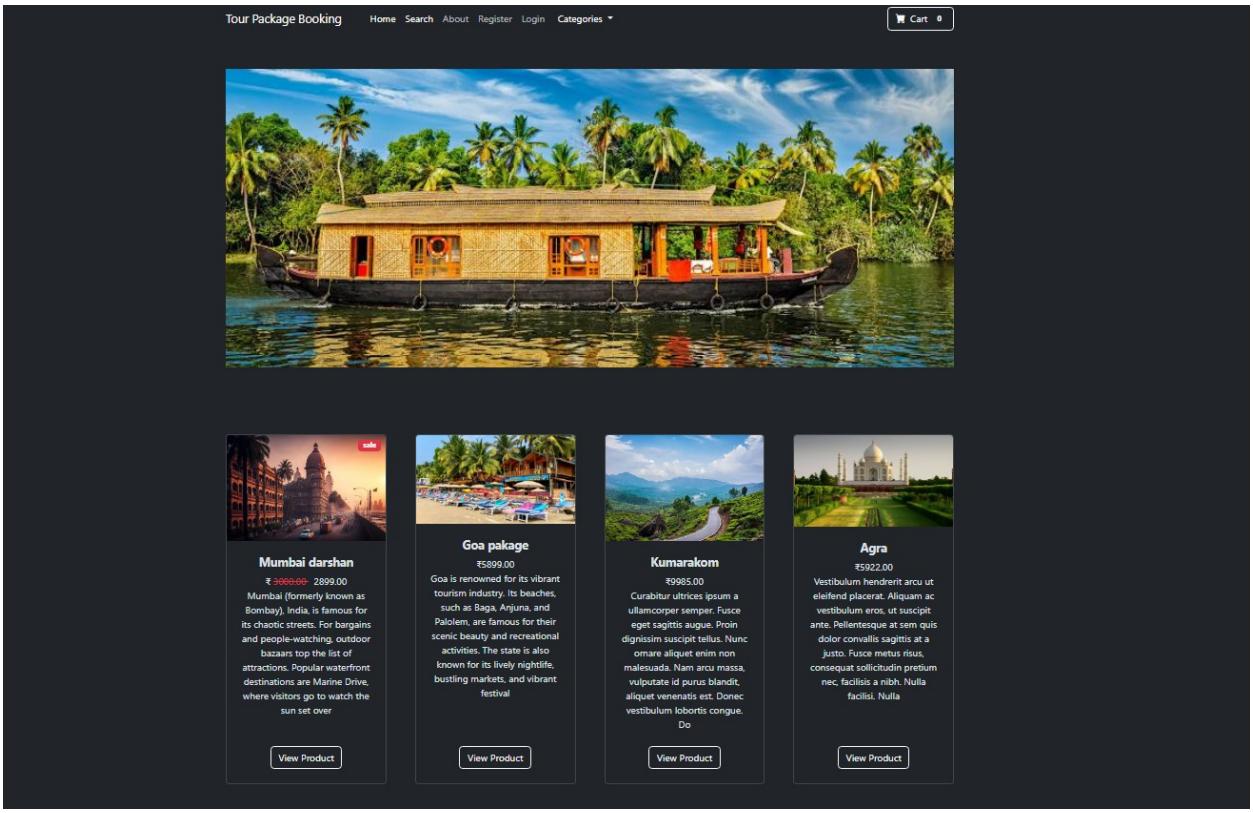
- User Data Store : Stores user credentials and account information.
- Package Data Store : Stores information about tour packages.
- Booking Data Store : Stores information about user bookings.

Data Flows:

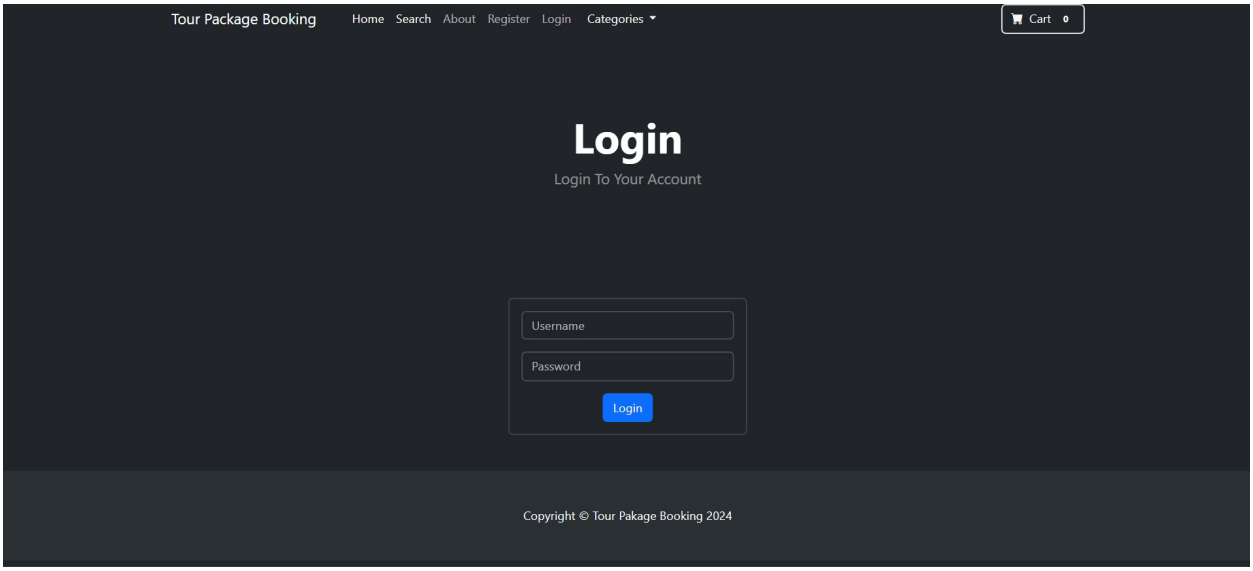
- User sends login credentials -> User Authentication process
- User Authentication process -> User Data Store for validation
- User searches for packages -> View Tour Packages process retrieves data from Package Data Store
- User selects a tour and submits booking details -> Booking Process stores booking in Booking Data Store
- Payment details -> Payment Handling interacts with Payment Gateway
- Admin updates package info -> Admin Management process updates Package Data Store

5.Designing : -

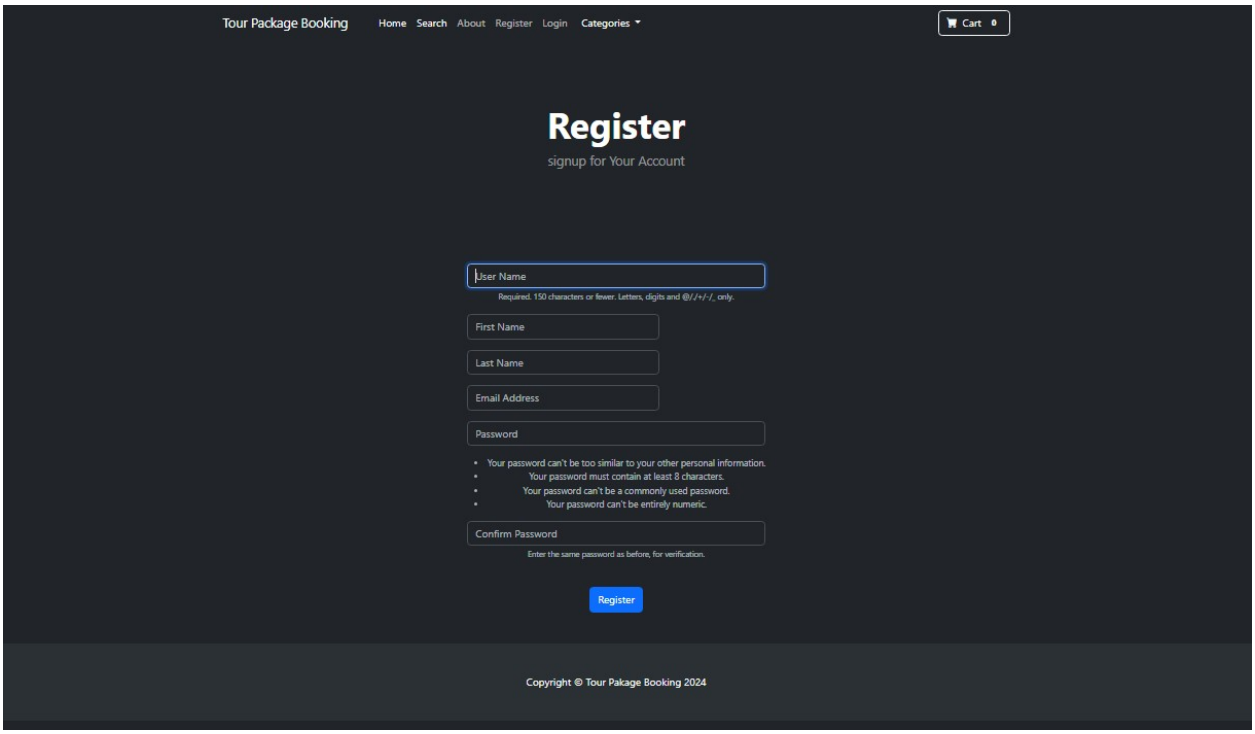
Home Page :-



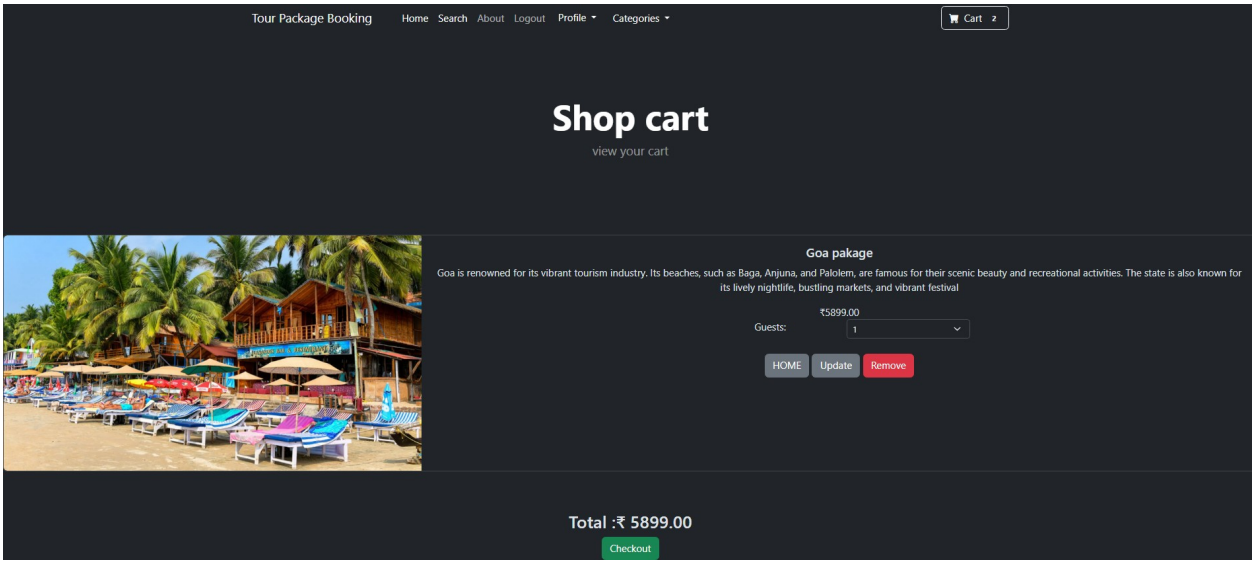
Login Page :-



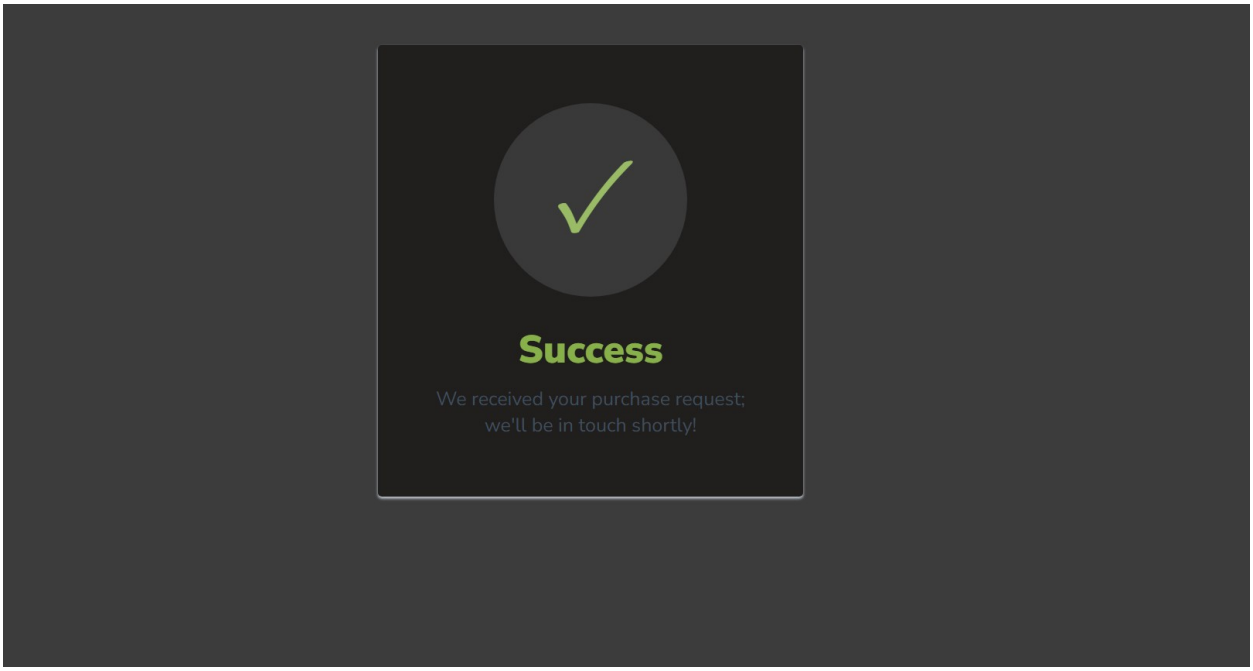
Registration Page :-



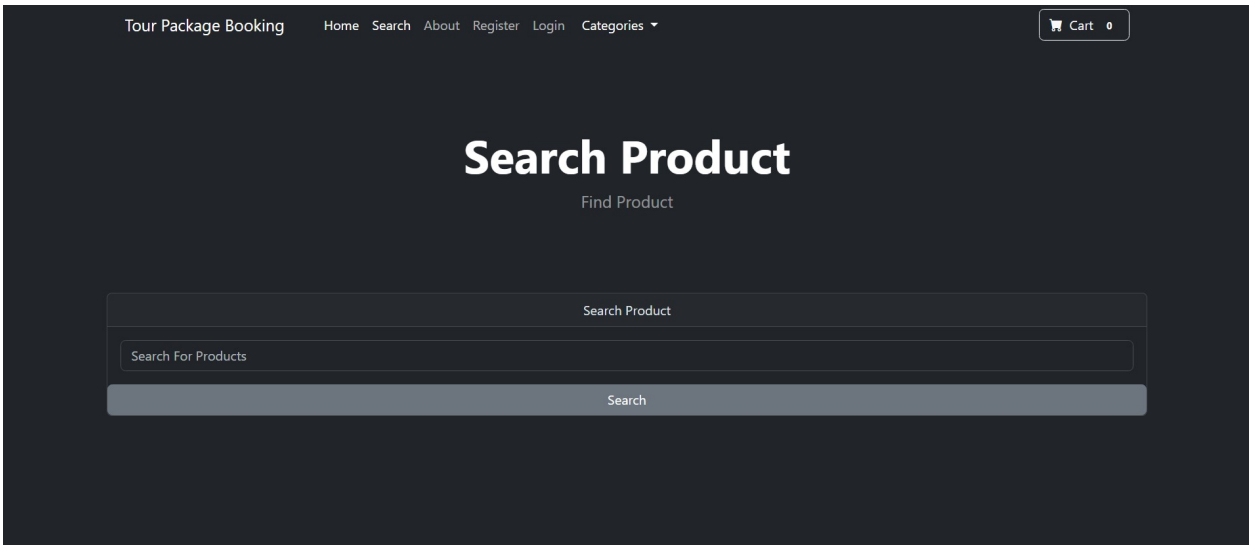
Cart & Checkout Page :-



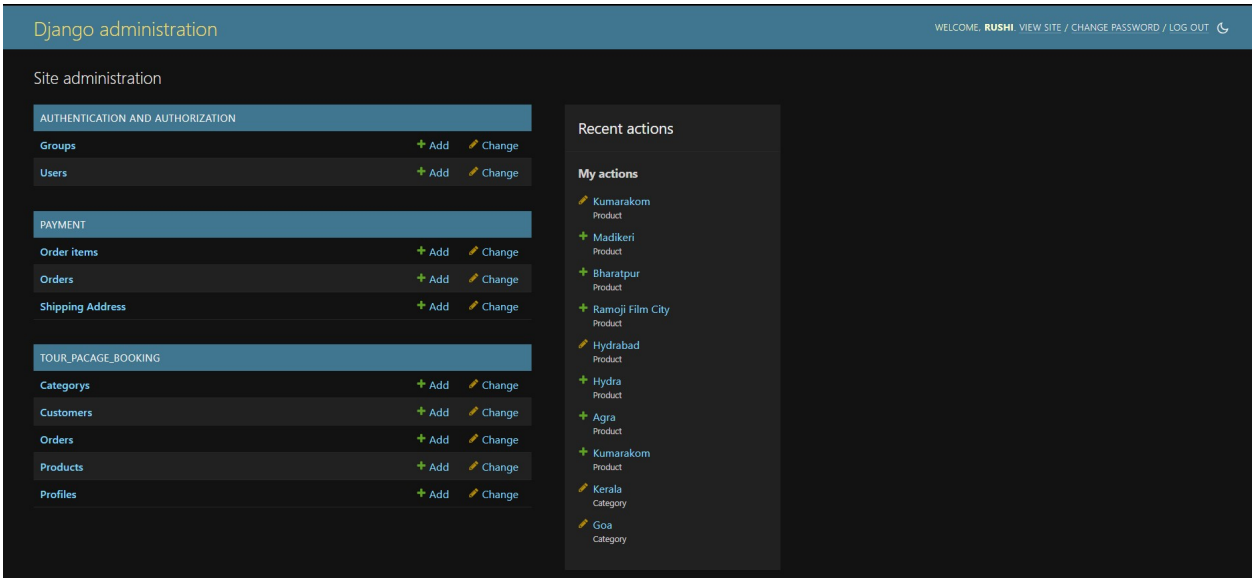
Payment Page :-



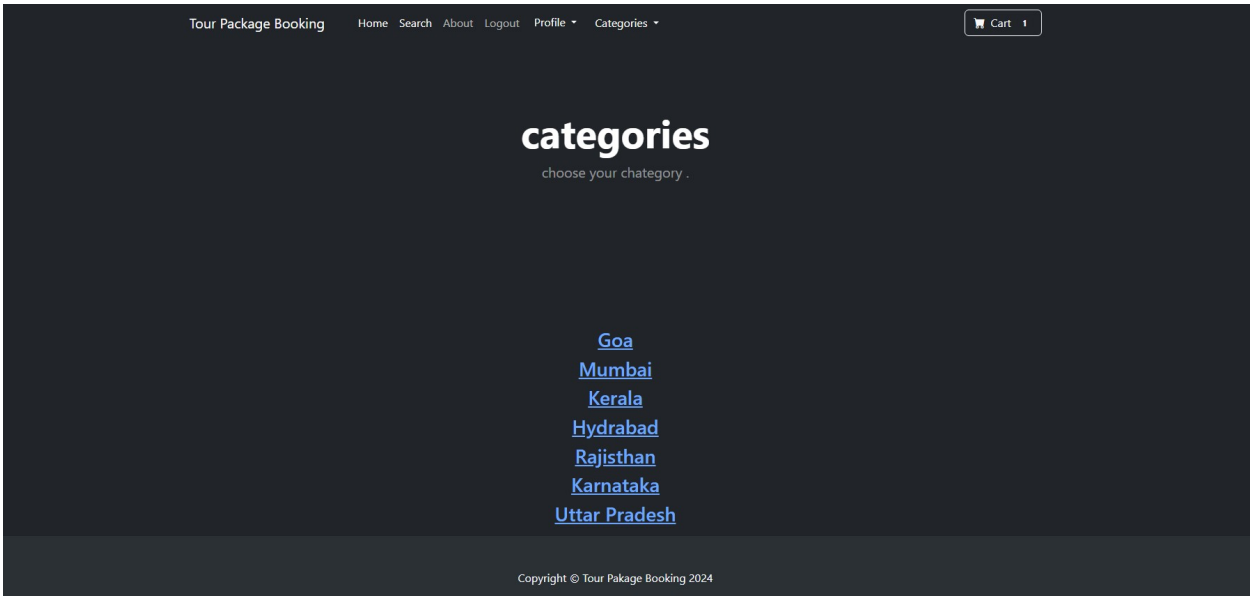
Search Page :-



Django Admin Page :-



Product-Category Page :-



5.1. Navigation

- **Intuitive Menu**: A structured navigation bar integrating existing Django templates with quick access to Home, Tour Packages, Bookings, Payments, Contact Us, and User Profile.
- **Search & Filters**: Leverage Django's filtering system to enable users to find tour packages based on location, price, and duration.
- **Breadcrumb Navigation**: Implement breadcrumb support from Django template context to enhance user tracking.

5.2. Color Scheme

- **Consistent Theme**: Use the existing color scheme from the Django E-comm repository, ensuring travel-related aesthetics with blues, oranges, and greens.
- **Contrast for Readability**: Maintain high readability with optimized color contrast.

5.3. Forms & Buttons

- **User-Friendly Booking Forms**: Utilize Django forms and crispy-forms for optimized input handling with dropdowns, calendars, and autofill options.
- **Call-to-Action Buttons**: Implement Bootstrap buttons from the repository for actions like "Book Now," "View Details," and "Contact Support."

5.4. Responsiveness

- **Mobile-First Design**: Ensure Bootstrap’s responsive grid layout is optimized for mobile, tablet, and desktop views.
- **Adaptive Layouts**: Use existing styles from the repository while ensuring flexibility in different screen sizes.

5.5. User Experience (UX)

- **Minimalist Yet Functional Design**: Maintain a clean UI with a focus on Django template inheritance for easy modifications.
- **Fast Load Times**: Optimize assets (images, CSS, and JavaScript) to improve website performance, leveraging Django's static file management.
- **Interactive Elements**: Implement smooth transitions, hover effects, and feedback animations using JavaScript and Bootstrap components already in the repository.

6.Coding:

Architecture

- The application follows the **MVC (Model-View-Controller) Architecture** for better scalability and maintainability.

6.1 Backend (Django - Models & Views)

The backend is developed using Django, handling business logic and database interactions.

Views

Handles Error Handling and User Feedback, User Registration , User Authentication, About Page , Home Page, Product Detail View , Product Management, Category Management, User Account Management , Password Management User Profile Management ,Handles Product Search .

views.py

```
from django.shortcuts import get_object_or_404,render,redirect

from .models import Category, Product,Profile

from django.contrib.auth import authenticate,login,logout

from django.contrib import messages

from django.contrib.auth.models import User

from django.contrib.auth.forms import UserCreationForm

from payment.forms import ShippingForm

from payment.models import ShippingAddress

from .forms import SignUpForm,UpdateUserForm,ChangePasswordForm,UserInfoForm

from django import forms

from django.db.models import Q

import json

from cart.cart import Cart


def search(request):

    if request.method == "POST":
```

```
        searched = request.POST['searched']

        #quary for product in model product


        searched     =   Product.objects.filter(Q(name__icontains   =
searched) | Q(discription__icontains = searched) )



        if not searched :

            messages.success(request , 'The Product dose not exist..
please try again..')

            return render(request,'search.html',{})

        else:

            return
render(request,'search.html',{'searched':searched})




     else :

        return render(request,'search.html',{})




def update_info(request):

    if request.user.is_authenticated:

        # Get the current user's profile

        current_user          =          get_object_or_404(Profile,
user__id=request.user.id)



        # Try to get the current user's shipping info; create a new
instance if it doesn't exist

        shipping_user                                              =
ShippingAddress.objects.filter(user__id=request.user.id).first()

        if not shipping_user:

            shipping_user  =  ShippingAddress(user=request.user)    #
Create an empty instance linked to the user



        # Get forms for user info and shipping info
```

```
        form = UserInfoForm(request.POST or None, instance=current_user)
        shipping_form = ShippingForm(request.POST or None, instance=shipping_user)

        if request.method == "POST" and form.is_valid() and shipping_form.is_valid():
            # Save the updated user info and shipping info
            form.save()
            shipping_form.save()

            messages.success(request, 'Your info has been updated!')
            return redirect('home')

        return render(request, 'update_info.html', {'form': form, 'shipping_form': shipping_form})
    else:
        messages.error(request, 'You must be logged in.')
        return redirect('home')



def update_password(request):
     if request.user.is_authenticated:
          current_user = request.user
          # Did they fill out the form
          if request.method  == 'POST':
               form = ChangePasswordForm(current_user, request.POST)
               # Is the form valid
               if form.is_valid():
                    form.save()
                    messages.success(request, "Your Password Has Been Updated...")
```

```
                    login(request, current_user)
                    return redirect('update_user')
               else:
                    for error in list(form.errors.values()):
                         messages.error(request, error)
                         return redirect('update_password')
          else:
               form = ChangePasswordForm(current_user)
               return     render(request,     "update_password.html",
{'form':form})
     else:
          messages.success(request, "You Must Be Logged In To View
That Page...")
          return redirect('home')



def update_user(request):
    if request.user.is_authenticated:
        current_user = User.objects.get(id=request.user.id)
        user_form     =     UpdateUserForm(request.POST     or     None
,instance=current_user)
        if user_form.is_valid():
            user_form.save()
            login(request,current_user)
            messages.success(request,'user has been updated !!')
            return redirect('home')
        return
render(request,'update_user.html',{'user_form':user_form})
    else:
        messages.error(request,'you must be loggend in ')
        return redirect('home')
```

```
def category_summary(request):
    categories = Category.objects.all()
    return
render(request,'category_summary.html',{'categories':categories})



def category(request,foo):
    #replace '-' hyphens with spaces
    foo = foo.replace('-',' ')
    #grab category from url
    try:
        category = Category.objects.get(name=foo)
        products = Product.objects.filter(category=category)
        return  render(request,'category.html',{'products':products  ,
'category':category})

    except:
        messages.success(request,('that category dosenot exist'))
        return redirect('home')



def product(request,pk):
    product = Product.objects.get(id=pk)
    return render(request, 'product.html', {'product': product})



# Create your views here.
def home(request):
```

```
    products = Product.objects.all
    return render(request, 'home.html', {'products': products})


def about(request):
    return render(request, 'about.html')



def login_user(request):
    if request.method == 'POST':
            username = request.POST.get('username')
            password = request.POST.get('password')
            user     =     authenticate(request,     username=username,
password=password)
            if user is not None:
                login(request, user)
                # cart stuff
                current_user                                          =
Profile.objects.get(user__id=request.user.id)
                  #get saved cart
                saved_cart = current_user.old_cart
                #cnvert db string to python  dicsinory
                if saved_cart:
                     #convert to json
                     converted_cart = json.loads(saved_cart)
                      #add loaded cart dictionary to session
                     cart = Cart(request)
                     for key,value in converted_cart.items():
                          cart.db_add(product=key,quantity=value)
```

```
            messages.success(request,("You Have Been Logged In!"))
            return redirect('home')
        else :
            messages.success(request,("There was an error, please try again..."))
        return redirect('login')

    else:
        return render(request, 'login.html',{})

def logout_user(request):
    logout(request)
    messages.success(request,("your have been logged out"))
    return redirect('home')

def register_user(request):
    form = SignUpForm()
    if request.method == 'POST':
        form = SignUpForm(request.POST)
        if form.is_valid():
            form.save()
            username = form.cleaned_data['username']
            password = form.cleaned_data['password1']

            # Login the user after successful registration
            user = authenticate(username=username, password=password)
            if user is not None:
                login(request, user)
```

```
                messages.success(request, "Username Created - Please
Fill Out Your User Info Below...")

                return redirect('update_info')

            else:

                messages.error(request,    "Authentication    failed.
Please try again.")

                return redirect('register')

        else:

            print(form.errors)  # Debugging: Print the form errors

            messages.error(request,  "Whoops!  There  was  a  problem
registering, please try again...")

            return redirect('register')

    else:

        return render(request, 'register.html', {'form': form})
```

URLs

Defines API and frontend routes.

urls.py

```
from django.urls import path

from . import views


urlpatterns = [

   path('',views.home,name ='home'),

   path('about/',views.about,name ='about'),

   path('login/',views.login_user,name ='login'),

   path('logout/',views.logout_user,name ='logout'),

   path('register/',views.register_user,name ='register'),

   path('update_password/',views.update_password,name
='update_password'),

   path('update_user/',views.update_user,name ='update_user'),

   path('update_info/',views.update_info,name ='update_info'),

   path('product/<int:pk>',views.product,name ='product'),
```

```
   path('category/<str:foo>',views.category,name ='category'),

   path('category_summary/',views.category_summary,name
='category_summary'),

   path('search/',views.search,name ='search'),
```

6.2.Frontend (HTML, CSS, JavaScript, Django Templates)

- Uses Django Templates to dynamically render web pages.
- CSS and JavaScript enhance UI/UX with a responsive design.
- Includes base.html for layout consistency.

Base.html

```
{% load static %}



<!DOCTYPE html>

<html lang="en" data-bs-theme="dark">

    <head>

        <meta charset="utf-8" />

        <meta name="viewport" content="width=device-width, initial-
scale=1, shrink-to-fit=no" />

        <meta name="description" content="" />

        <meta name="author" content="" />

        <title>Tour Package Booking</title>

        <!-- Favicon-->

        <link rel="icon" type="image/x-icon" href="{% static
'assets/favicon.ico'%}" />

        <!-- Bootstrap icons-->

        <link href="https://cdn.jsdelivr.net/npm/bootstrap-
icons@1.5.0/font/bootstrap-icons.css" rel="stylesheet" />

        <!-- Core theme CSS (includes Bootstrap)-->

        <link href="{% static 'css/styles.css' %}" rel="stylesheet"
/>




        <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstra
```

```
p.min.css"            rel="stylesheet"            integrity="sha384-
QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH"
crossorigin="anonymous">


        <script     src="https://code.jquery.com/jquery-3.7.1.min.js"
integrity="sha256-/JqT3SQfawRcv/BIHPThkBvs0OEvtFFmqPF/lYI/Cxo="
crossorigin="anonymous"></script>

    </head>

    <body>


        {% include 'navbar.html' %}

            {% if messages %}

                {% for message in messages  %}


                <div  class="alert  alert-warning  alert-dismissible
fade show" role="alert">

                    {{ message }}

                    <button type="button" class="btn-close" data-bs-
dismiss="alert" aria-label="Close"></button>

                  </div>

                {% endfor%}

            {% endif%}


        {% block content%}


        {% endblock%}




        <!-- Footer-->

        <footer class="py-5 bg-body-tertiary">

            <div  class="container"><p  class="m-0  text-center  text-
light">Copyright &copy; Tour Pakage Booking 2024</p></div>
```

```
        </footer>
        <!-- Bootstrap core JS-->
        <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.
bundle.min.js"></script>
        <!-- Core theme JS-->
        <script src="{% static 'js/script.js' %}"></script>
    </body>
</html>
```

Home.html

```
{% extends 'base.html' %}
{%load static %}
{% block content%}

                <!-- Header-->
                <style>
                    /* Custom styles for carousel size */
                    .carousel {
                        max-width: 2000px; /* Set a max width for
the carousel */
                        margin: auto; /* Center the carousel */
                    }
                    .carousel-item img {
                        width: 100%; /* Make images responsive */
                        height: 500px; /* Set a fixed height for
images */
                        object-fit: cover; /* Cover the area while
maintaining aspect ratio */
                    }
                </style>

        <header class="bg-dark py-2">
            <div class="container px-4 px-lg-5 my-5">
```

```
                <div id="carouselExampleAutoplaying" class="carousel
slide" data-bs-ride="carousel">

                    <div class="carousel-inner">

                      <div class="carousel-item active">

                      <img   src="{%static   '/slider/goa.jpg'   %}"
class="d-block w-100" alt="...">

                      </div>

                      <div class="carousel-item">

                        <img  src="{%static  '/slider/mumbai.jpg'  %}"
class="d-block w-100" alt="...">

                      </div>

                      <div class="carousel-item">

                        <img  src="{%static  '/slider/kerla.jpg'  %}"
class="d-block w-100" alt="...">

                      </div>

                    </div>

                    <button             class="carousel-control-prev"
type="button"  data-bs-target="#carouselExampleAutoplaying"  data-bs-
slide="prev">

                      <span  class="carousel-control-prev-icon"  aria-
hidden="true"></span>

                      <span class="visually-hidden">Previous</span>

                    </button>

                    <button             class="carousel-control-next"
type="button"  data-bs-target="#carouselExampleAutoplaying"  data-bs-
slide="next">

                      <span  class="carousel-control-next-icon"  aria-
hidden="true"></span>

                      <span class="visually-hidden">Next</span>

                    </button>

                  </div>

                <!-- <div class="text-center text-white">

                    <h1    class="display-4    fw-bolder">Shop    in
style</h1>

                    <p   class="lead   fw-normal   text-white-50   mb-
0">With this shop hompeage template</p>

                </div> -->
```

```
            </div>
        </header>
        <!-- to do
        https://getbootstrap.com/docs/4.0/components/carousel/

        -->
        <!-- Section-->
        <section class="py-2">
            <div class="container px-4 px-lg-5 mt-5">
                <div class="row gx-4 gx-lg-5 row-cols-2 row-cols-md-
3 row-cols-xl-4 justify-content-center">

                    {% for product in products %}
                      {% if product.is_sale %}
                       <!-- box1-->
                       <div class="col mb-5">
                          <div class="card h-100">

                                <!--sale badge-->
                                <div  class="badge  bg-danger  text-light
position-absolute" style ="top:0.5rem; right:0.5rem ">
                                    sale
                                </div>

                                <!-- Product image-->
                                <img                 class="card-img-top"
src="{{product.image.url}}" alt="..." />
                                <!-- Product details-->
                                <div class="card-body p-4">
                                    <div class="text-center">
                                        <!-- Product name-->
```

```
                                    <h5                    class="fw-
bolder">{{product.name}}</h5>
                                    <!-- Product price-->

                                    <!--sign   for   rupees   code   '
&#8337; ' -->
                                    &#8377;<!--   quary   for   product
price-->
                                    <del     class     ="text-danger"
>{{product.price}} </del>  
                                    <!--sale_price-->
                                    {{product.sale_price}}
                                    <br/>
                                    <!--quary   for   description   -->
{{product.discription}}
                                </div>
                            </div>
                            <!-- Product actions-->
                            <div class="card-footer p-4 pt-0 border-
top-0 bg-transparent">
                                <div          class="text-center"><a
class="btn  btn-outline-light  mt-auto"  href="{%  url  'product'
product.id %}">View Product</a></div>
                            </div>
                        </div>
                    </div>
                    {% else %}
                     <!-- box1-->
                     <div class="col mb-5">
                        <div class="card h-100">
                            <!-- Product image-->
                            <img                class="card-img-top"
src="{{product.image.url}}" alt="..." />
                            <!-- Product details-->
                            <div class="card-body p-4">
```

```
                                    <div class="text-center">
                                        <!-- Product name-->
                                        <h5                         class="fw-
bolder">{{product.name}}</h5>
                                        <!-- Product price-->
                                       <!--sign   for   rupees   code   '
&#8337; ' --> &#8377;{{product.price}} <!-- quary for product price-
->
                                       <br/>
                                       <!--quary   for   description   -->
{{product.discription}}
                                    </div>
                                </div>
                                <!-- Product actions-->
                                <div class="card-footer p-4 pt-0 border-
top-0 bg-transparent">
                                    <div          class="text-center"><a
class="btn   btn-outline-light   mt-auto"   href="{%   url   'product'
product.id %}">View Product</a></div>
                                </div>
                            </div>
                        </div>


                        {% endif%}
                        {% endfor%}




                    </div>
                </div>
            </section>


{% endblock %}
```

6.3.Middleware

- **Global error handling** ensures smooth application performance.
- **Logging mechanisms** track and debug issues efficiently.

6.4.Security

- **User Authentication & Authorization** using Django's built-in authentication system.

- **CSRF Protection & Input Validation** prevent security vulnerabilities.

6.5.Performance Optimizations

- **Caching** techniques (Django cache framework) improve response times.
- **Pagination** optimizes data retrieval and enhances page load speed.

6.6 Deployment & CI/CD

- Automated Builds: The pipeline is triggered on code pushes and pull requests to the main branch, automatically initiating the build process.
- Environment Setup: The pipeline sets up a Python environment with the specified version (Python 3.10) and installs all necessary dependencies from the requirements.txt file.
- Database Migrations: It automatically runs database migrations to ensure that the database schema is up to date with the latest changes in the application.
- Automated Testing: The pipeline includes a step to run automated tests, verifying that the application functions correctly before deployment. This helps catch any issues early in the development process.
- Deployment to PythonAnywhere: Upon successful completion of the build and tests, the application is deployed to PythonAnywhere. The deployment process includes pulling the latest code, installing dependencies, running migrations, and collecting static files.
- Secure Credentials: Sensitive information, such as PythonAnywhere credentials, is managed securely using GitHub Secrets, ensuring that sensitive data is not exposed in the codebase.

```
name: CICD_Django_Ecomm

on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest
```

```
        steps:
        - name: Checkout code
          uses: actions/checkout@v2

        - name: Set up Python
          uses: actions/setup-python@v2
          with:
            python-version: '3.10'

        - name: Install dependencies
          run: |
            python -m pip install --upgrade pip
            pip install -r requirements.txt

        - name: Run migrations
          run: |
            python manage.py migrate

        - name: Run Tests
          run: |
            python manage.py test

        - name: Deploy to PythonAnywhere
          env:
            PA_USERNAME: ${{ secrets.PA_USERNAME }}
            PA_PASSWORD: ${{ secrets.PA_PASSWORD }}
          run: |
            echo "Deploying to PythonAnywhere..."
            sshpass -p "$PA_PASSWORD" ssh -o
StrictHostKeyChecking=no "$PA_USERNAME@ssh.pythonanywhere.com" <<
'EOF'
              cd /home/rushiMithagare/Django-E-comm
              git pull origin main
              ...
            EOF
```

7.Future Scope

- **Payment Gateway Integration**
  - Implement **Razorpay** to allow secure online transactions.
  - Enable multiple payment methods, including **credit/debit cards, UPI, and net banking**.
- **PostgreSQL Database Migration**
  - Move from **SQLite to PostgreSQL** for better scalability, performance, and reliability.
- **AI-Powered Tour Recommendations**
  - Use **machine learning** to suggest personalized tour packages based on user preferences and past bookings.
- **User Reviews & Ratings System**
  - Allow users to **rate and review** tours to enhance credibility and user trust.
  - Implement a **moderation system** for filtering inappropriate content.
- **Multilingual Support**
  - Add **multiple language options** to cater to a global audience.
- **Advanced Search & Filters**
  - Implement **filters for location, price range, duration, and ratings** to improve user experience.
- **Mobile App Development**
  - Develop a **cross-platform mobile app** using Flutter or React Native for better accessibility.
- **Automated Booking & Confirmation System**
  - Integrate **automated email/SMS notifications** for booking confirmations and reminders.
- **Google Maps & Location-Based Services**
  - Add **interactive maps** for tour locations and real-time distance calculations.
- **Admin Dashboard for Analytics**

- Provide an **admin panel** with insights on **user behavior, popular destinations, and revenue trends**.

8.Conclusion :

The **Tour Package Booking System** effectively addresses the limitations of traditional travel booking methods by providing a **comprehensive digital solution** for booking and managing tour packages. By leveraging **modern web technologies like Django**, the system ensures a **secure, scalable, and user-friendly** experience that simplifies the tour selection and booking process for users while optimizing operations for administrators.

This project highlights the **transformative power of digital platforms** in the travel industry, making it easier for users to explore and book various tour options at their convenience. Additionally, **automation improves business efficiency** by minimizing manual data handling, **enhancing customer response times, and generating valuable insights** through reporting and analytics features.

Future Enhancements

Looking ahead, several improvements can be incorporated to enhance the platform's functionality:

- **Mobile Application Development** for seamless access on smartphones and tablets.
- **Integration of Dynamic Pricing Models** to offer real-time price adjustments based on demand and availability.
- **Real-time Notifications** via email/SMS for instant booking updates and reminders.
- **Personalized Package Recommendations** using AI-based user behavior analysis.

These enhancements will further strengthen the system, ensuring it meets the evolving needs of both **travelers and tour operators** while maintaining a **modern, seamless booking experience**.

9.Bibliography:

Books:

Elder, John. *Introduction to Django with Python for Web Development*. 1st ed., Codemy, 2020.

Duckett, Jon. *HTML and CSS: Design and Build Websites.* 1st ed., Wiley, 2011.

Duckett, Jon. *JavaScript and JQuery: Interactive Front-End Web Development.* 1st ed., Wiley, 2014.

Flanagan, David. *JavaScript: The Definitive Guide.* 7th ed., O'Reilly Media, 2020.

Websites:

1. **Django Official Documentation** : https://www.djangoproject.com/
2. **SQLite Official Website**: https://www.sqlite.org/
3. **Codemy – Django Tutorials** : https://www.codemy.com/
4. **Stackoverflow** : https://stackoverflow.com/questions/tagged/django-forms