PROJECT REPORT

BUSIFY:
A COMPREHENSIVE BUS SCHEDULE MANAGEMENT & BOOKING SYSTEM

By

Linkan Kumar Rout

ID - MST01-0055

ABSTRACT

This project report presents the development and implementation of "Busify," a comprehensive bus schedule management and booking system built using the Django framework. This project aims to streamline the management and booking of bus schedules through a user-friendly web application. The primary purpose is to provide an efficient platform for users to view, search, and book bus trips while enabling administrators to manage schedules easily. Technologies used include the Django framework for the backend and SQLite for the database, complemented by HTML, CSS, Bootstrap, and JavaScript for the frontend. The project incorporates secure user authentication and authorization, an admin interface for CRUD operations on bus schedules, and search and filter functionalities for users to find specific schedules. Additionally, it features efficient pagination and dynamic sorting of schedules to enhance user experience. An integral part of the system is the integration of email functionality to send booking confirmations automatically, ensuring users are notified of their bookings promptly. PayPal payment integration has been made in this project to facilitate seamless and secure online transactions for booking bus trips. The application is designed to be responsive and compatible with various devices, providing a seamless user experience. The project successfully delivers a functional web application that allows users to search for and book bus trips easily while enabling efficient schedule management through an admin panel. Automated status updates for schedules based on date and time are implemented, along with successful user authentication and email booking confirmations. This project demonstrates the effectiveness of Django in building scalable web applications with robust backend and responsive frontend capabilities, serving as a foundation for future enhancements and customization.

Table of Contents

1. Introduction

The rapid advancements in technology have revolutionized various aspects of daily life, including transportation. Efficient transportation management is essential for the smooth functioning of urban and rural areas alike. With the increasing reliance on public transportation, there is a growing need for systems that can effectively manage bus schedules and bookings. This project, "Busify: A Comprehensive Bus Schedule Management and Booking System," aims to address this need by providing a robust, user-friendly platform for managing bus operations.

Busify is designed to streamline the process of bus schedule management, allowing administrators to create, update, and delete schedules with ease. The system also provides users with the ability to search for bus schedules, book tickets, and receive email confirmations for their bookings. Built using the Django framework, Busify leverages modern web technologies to ensure a responsive and intuitive user experience.

The project utilizes SQLite as its database, ensuring a lightweight yet powerful data storage solution. Front-end technologies such as HTML, CSS, Bootstrap, and JavaScript are employed to create an appealing and functional interface. The integration of these technologies results in a comprehensive system that caters to the needs of both administrators and users.

This report delves into the various aspects of the project, including its purpose, system architecture, implementation details, and the challenges faced during development. Additionally, it outlines potential future enhancements that could further improve the system's functionality and user experience. Through this project, we aim to contribute to the efficient management of public transportation, enhancing the convenience and reliability of bus services for users.

2. Project Overview

The project "Busify: A Comprehensive Bus Schedule Management and Booking System" represents a modern solution tailored for efficient management and booking of bus schedules. It addresses the critical need for a user-friendly platform that benefits both administrators and passengers alike. With a focus on usability and accessibility, Busify allows administrators to seamlessly create, update, and delete bus schedules through an intuitive web interface. This ensures schedules are accurately maintained and readily accessible to users. Built using contemporary technologies such as Django, HTML, CSS, Bootstrap, and JavaScript, Busify offers a responsive and visually appealing interface. Users can effortlessly search for bus schedules based on their preferences and book tickets with minimal effort.

One of Busify's standout features is its streamlined booking and confirmation process. Users can securely book bus tickets directly through the platform and receive email confirmations, enhancing reliability and convenience. The project leverages SQLite as its database, ensuring efficient data management and system performance even under high load conditions. Looking forward, Busify is designed with scalability in mind, paving the way for future enhancements such as integration with additional payment gateways, real-time bus tracking functionalities, and advanced analytics for route optimization. This project report provides a comprehensive overview of the development journey, system architecture, implementation strategies, challenges faced, and opportunities for further enhancement.

3. Technologies Used

1. Django – 4.2.13
2. Python – 3.11.9
3. HTML 5
4. CSS 3
5. JavaScript
6. Bootstrap 4
7. J Query
8. G-mail (for smtp config)
9. SQLite3 (default database engine)
10. PayPal (Payment Gateway Integration)

4. System Architecture

Busify's system architecture is designed to provide a robust and scalable platform for managing bus schedules and bookings effectively. The architecture is structured into frontend, backend, database management, and integration with essential services.

The **frontend** of Busify is developed using HTML, CSS, JavaScript, and Bootstrap, ensuring a responsive and user-friendly interface. These technologies enable dynamic rendering of content, seamless navigation, and intuitive user interactions. The frontend communicates with the backend via HTTP requests to retrieve data, update schedules, and manage user bookings.

On the **backend**, Busify leverages Django, a powerful Python-based web framework. Django's MVC (Model-View-Controller) architecture and ORM (Object-Relational Mapping) simplify the development of complex business logic and ensure efficient data handling. The backend manages user authentication, business rules for scheduling, and data persistence using SQLite, a lightweight relational database. SQLite is chosen for its simplicity, compatibility with Django, and ability to handle moderate data volumes effectively.

Busify integrates SMTP for email notifications, keeping users informed about their booking statuses in real-time. The architecture is designed to support potential integrations with third-party APIs, such as payment gateways or location services, to enhance functionality and provide additional value to users.

In summary, Busify's system architecture is tailored to meet the demands of managing bus schedules and bookings efficiently. It ensures reliability, scalability, and maintainability, providing a seamless experience for both users booking bus tickets and administrators managing schedules.

5. Implementation Details

Home Page: The Home Page (/ or /home) serves as the gateway to the Busify platform, offering a welcoming introduction and overview of its services. It prominently features a carousel or banner section highlighting key features, promotional offers, or current bus routes. This page aims to engage users immediately upon arrival, encouraging them to explore further.

Find Trip Page: The Find Trip Page provides a user-friendly interface for users to search for specific bus trips based on various parameters. Users can input departure and destination locations, preferred date and time to refine their search. The page displays relevant schedules that meet the search criteria, facilitating efficient trip planning.

All Schedules Page: The All Schedules Page provides a comprehensive listing of all available bus schedules within the Busify system. It displays schedules in a structured table format, detailing essential information such as bus number, departure point, destination, schedule time, fare per person, and the current status (active or cancelled). Users can paginate through the schedules and utilize sorting options to find specific routes or times that suit their travel needs.

Search Results Page: Upon initiating a search query through the site's search functionality (typically accessed via a search bar on the navigation or schedules page), users are directed to the Search Results Page. This page dynamically generates results based on the user's search terms, displaying schedules that match the criteria. It enhances user convenience by quickly narrowing down available schedules based on specific preferences or destinations.

Authentication Pages: Authentication Pages are crucial for user security and access management. The Login Page allows existing users to securely sign in to their accounts, while the Registration Page enables new users to create profiles within the Busify system. These pages ensure that user interactions with the platform are authenticated and protected, enhancing overall security and trust.

Bus Categories Page: The Bus Categories Page showcases different categories of buses available for travel through Busify. It provides descriptions and features of each category, such as luxury, double decker, sleeper buses, mini bus etc. helping users choose the most suitable option for their journey preferences.

Locations Page: The Locations Page presents a comprehensive list of all available travel destinations serviced by Busify.

Booking Confirmation Page: The Booking Confirmation Page confirms the details of a successfully booked schedule. It displays essential booking information, including bus number, departure/destination locations, scheduled time, fare, and booking status. This page reassures users that their booking is confirmed but payment has to be done in user booking page and provides a reference for future travel plans.

Contact Us Page: The Contact Us Page offers users a direct means to communicate with Busify's support team. It includes a contact form where users can submit inquiries, feedback, or issues they encounter while using the platform. Additionally, contact details such as email addresses, phone numbers, and physical addresses may be provided for users preferring alternative communication methods.

Payment Page: The Payment Page facilitates secure online transactions for booking bus trips. Users can complete their payments through integrated payment gateways like PayPal. This page ensures a smooth and secure payment process, enabling users to finalize their bookings effortlessly.

User Booking Page: The User Booking Page displays all of a user's bookings, categorized by pending and completed bookings. For pending bookings, users have the option to make payments. For completed bookings, users can download their tickets and have the option to delete them if needed. This page provides users with a centralized location to manage their bus trip bookings effectively.

1. models.py

The models.py file is a crucial part of the Django project that defines the data models for the application. These models represent the structure of the database and include fields that describe the attributes of each model. In the Busify project, the models typically include:

- **Bus**: This model defines the attributes of a bus, such as the bus number, category, bus image and capacity.
- **Schedule**: This model contains information about bus schedules, including the bus, departure location, destination, time, fare, and status.
- **Category**: This model categorizes buses, such as luxury, inter-city, sleeper etc.
- **Location**: This model defines the travel destinations serviced by Busify.
- **Booking:** This model defines the booking made by user.

```python
from django.db import models
from django.utils import timezone
from django.db.models import Sum
import random
import string

def generate_unique_code():
    timestamp_str = str(int(timezone.now().timestamp()))  # Get current timestamp as string
    random_chars = ''.join(random.choices(string.ascii_uppercase + string.digits, k=6))  # Generate random 6-character string
    return timestamp_str + '-' + random_chars


class Category(models.Model):
    name = models.CharField(max_length=250)
    description = models.TextField()
    status = models.CharField(max_length=2, choices=(('1','Active'),('2','Inactive')), default=1)
    date_created = models.DateTimeField(default=timezone.now)
    date_updated = models.DateTimeField(auto_now=True)

    def __str__(self):
        return self.name
```

```python
class Location(models.Model):
    location = models.CharField(max_length=250)
    status = models.CharField(max_length=2, choices=(('1','Active'),('2','Inactive')), default=1)
    date_created = models.DateTimeField(default=timezone.now)
    date_updated = models.DateTimeField(auto_now=True)

    def __str__(self):
        return self.location
```

```python
class Bus(models.Model):
    category = models.ForeignKey(Category,on_delete=models.CASCADE, blank= True, null = True)
    bus_number = models.CharField(max_length=250)
    seats = models.IntegerField()
    status = models.CharField(max_length=2, choices=(('1','Active'),('2','Inactive')), default=1)
    date_created = models.DateTimeField(default=timezone.now)
    date_updated = models.DateTimeField(auto_now=True)
    image = models.ImageField(upload_to='bus_images/', blank=True, null=True)

    def __str__(self):
        return self.bus_number

class Schedule(models.Model):
    code = models.CharField(max_length=20, default=generate_unique_code, unique=True)
    bus = models.ForeignKey(Bus, on_delete=models.CASCADE)
    depart = models.ForeignKey(Location, on_delete=models.CASCADE, related_name='depart_location')
    destination = models.ForeignKey(Location, on_delete=models.CASCADE, related_name='destination')
    schedule = models.DateTimeField()
    fare = models.FloatField()
    status = models.CharField(max_length=2, choices=(('1', 'Active'), ('2', 'Cancelled')), default='1')
    date_created = models.DateTimeField(default=timezone.now)
    date_updated = models.DateTimeField(auto_now=True)

    def __str__(self):
        return str(self.code + ' - ' + self.bus.bus_number)

    def count_available(self):
        booked = Booking.objects.filter(schedule=self).aggregate(Sum('seats'))['seats__sum'] or 0
        return self.bus.seats - booked

    def get_available_seats(self):
        booked_seats = Booking.objects.filter(schedule=self).values_list('selected_seats', flat=True)
        booked_seats = [seat for sublist in booked_seats for seat in sublist]  # Flatten the list of lists
        return [seat for seat in range(1, self.bus.seats + 1) if seat not in booked_seats]

class Booking(models.Model):
    STATUS_CHOICES = (
        ('1', 'Pending'),
        ('2', 'Paid'),
        ('3', 'Cancelled'),  # Added Cancelled status
    )

    code = models.CharField(max_length=100)
    name = models.CharField(max_length=250)
    schedule = models.ForeignKey('Schedule', on_delete=models.CASCADE)
    seats = models.PositiveIntegerField(default=0)
    selected_seats = models.JSONField(default=list)  # Store selected seats
    status = models.CharField(max_length=2, choices=STATUS_CHOICES, default='1')
    date_created = models.DateTimeField(default=timezone.now)
    date_updated = models.DateTimeField(auto_now=True)

    def __str__(self):
        return f"{self.code} - {self.name}"

    def total_payable(self):
        return self.seats * self.schedule.fare

class Contact(models.Model):
    name = models.CharField(max_length=250)
    email = models.EmailField()
    subject = models.CharField(max_length=250)
    message = models.TextField()
    date_submitted = models.DateTimeField(default=timezone.now)

    def __str__(self):
        return self.subject
```

2. views.py

The views.py file is responsible for handling the logic of the application. It processes user requests, interacts with the models to retrieve or update data, and returns the appropriate responses, usually in the form of rendered HTML templates. Key views in the Busify project include:

- **Home Page View**: Renders the home page with introductory information and popular routes.

- **Schedule List View**: Displays all available bus schedules, allowing sorting and pagination.
- **Find Trip View**: Allows users to search for trips based on departure and destination locations and other criteria.
- **Bus Categories View**: Shows different categories of buses available.
- **Locations View**: Lists all available travel destinations.
- **Contact Us View**: Provides a form for users to submit inquiries or feedback.
- **Authentication Views**: Handles user login and registration processes.

```python
import re
from django.shortcuts import render, redirect, get_object_or_404
from django.contrib.auth.models import User
from django.contrib.auth import authenticate, login, logout
from django.contrib.auth.decorators import login_required
from django.contrib import messages
from django.core.exceptions import ValidationError
from django.utils import timezone
from django.http import JsonResponse
from django.views.decorators.http import require_POST
from django.core.paginator import Paginator
from django.db.models import Q
from .models import *
from .utils.email_utils import send_booking_confirmation

# Home Page View
def homePage(request):
    return render(request, "reserve/index.html")

# Function for validating password
def validate_password(password1, password2):
    if password1 != password2:
        raise ValidationError('Passwords do not match')

    if len(password1) < 8:
        raise ValidationError('Password must be at least 8 characters long')

    if not re.search(r'[A-Z]', password1):
        raise ValidationError('Password must contain at least one uppercase letter')

    if not re.search(r'[a-z]', password1):
        raise ValidationError('Password must contain at least one lowercase letter')

    if not re.search(r'\d', password1):
        raise ValidationError('Password must contain at least one digit')

    if not re.search(r'[!@#$%^&*(),.?":{}|<>]', password1):
        raise ValidationError('Password must contain at least one special character')
```

```python
# SignUp View
def signup_view(request):
    if request.method == 'POST':
        uname = request.POST.get('username')
        email = request.POST.get('email')
        pass1 = request.POST.get('password1')
        pass2 = request.POST.get('password2')
        first_name = request.POST.get('firstname')
        last_name = request.POST.get('lastname')
        password = re.match('^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$', pass1)

        if password is None:
            messages.error(request, 'Your password must be 8 characters and combination of one uppercase, one lowercase, one digit, and one special character atleast.')
            return redirect('signup')
        else:
            if pass1 != pass2:
                messages.error(request, 'Password and Confirm password are not matchin g')
                return redirect('signup')
            else:
                my_user = User.objects.create_user(uname, email, pass1)
                my_user.is_active = True
                my_user.first_name = first_name
                my_user.last_name = last_name
                my_user.save()
                messages.success(request, 'You have signed up successfully!')
                return redirect('signup')
    return render(request, 'registration/signup.html')

# LogIn View
def login_view(request):
    if request.user.is_authenticated:
        return redirect('home')

    if request.method == 'POST':
        username = request.POST.get('username')
        password = request.POST.get('password')

        if not username or not password:
            messages.error(request, 'Please fill out both fields.')
            return render(request, 'registration/login.html')

        user = authenticate(request, username=username, password=password)
        if user is not None:
            login(request, user)
            messages.success(request, 'Logged in successfully!')
            if user.is_staff:
                return redirect('/admin/')  # Redirect admins to admin interface
            return redirect('home')
        else:
            messages.error(request, 'Invalid username or password.')

    return render(request, 'registration/login.html')

# Logout View
def logout_view(request):
    logout(request)
    return redirect('home')

```

```python
# Finding Trips
def find_trips(request):
    trips = None
    if request.method == 'POST':
        departure = request.POST.get('departure')
        destination = request.POST.get('destination')
        date = request.POST.get('date')

        if departure and destination and date:
            trips = Schedule.objects.filter(
                depart__location__icontains=departure,
                destination__location__icontains=destination,
                schedule__date=date,
                status='1'
            )
            return render(request, 'reserve/all_trips.html', {'trips': trips})

    return render(request, 'reserve/find_trips.html')

# Book a Trip with Seat selection
@login_required
def book_trip(request, schedule_id):
    schedule = Schedule.objects.get(id=schedule_id)
    booked_seats = Booking.objects.filter(schedule=schedule).values_list('selected_seats', flat=True)
    booked_seats = [seat for sublist in booked_seats for seat in sublist]  # Flatten the list of lists
    seat_range = range(1, schedule.bus.seats + 1)

    if request.method == 'POST':
        selected_seats = request.POST.getlist('seats')
        selected_seats = [int(seat) for seat in selected_seats]  # Convert to integers

        if all(seat not in booked_seats for seat in selected_seats):
            booking = Booking(
                code='BKG' + str(timezone.now().timestamp()).replace('.', ''),
                name=request.user.first_name,
                schedule=schedule,
                seats=len(selected_seats),
                selected_seats=selected_seats,
                status='2'  # Paid status
            )
            booking.save()

            return redirect('booking-confirmation', booking_id=booking.id)
        else:
            error_message = "Some of the selected seats are not available."
            return render(request, 'reserve/book_trip.html', {'schedule': schedule, 'booked_seats': booked_seats, 'seat_range': seat_range, 'error_message': error_message})

    return render(request, 'reserve/book_trip.html', {'schedule': schedule, 'booked_seats': booked_seats, 'seat_range': seat_range})

# Booking Confirmation View & and Sending Mail to respective user
@login_required
def booking_confirmation(request, booking_id):
    booking = get_object_or_404(Booking, id=booking_id)

    # Prepare booking details
    from_location = booking.schedule.depart
    to_location = booking.schedule.destination
    booking_details = f"""
    Booking Code: {booking.code}
    Name: {booking.name}
    From: {from_location}
    To: {to_location}
    Schedule: {booking.schedule.schedule.strftime('%Y-%m-%d %H:%M')}
    Seats: {booking.seats}
    Total Payable: {booking.total_payable()}
    Status: {'Pending' if booking.status == '1' else 'Paid'}
    Date Created: {booking.date_created.strftime('%Y-%m-%d %H:%M')}
    """

    # Send email confirmation
    user_email = request.user.email
    send_booking_confirmation(user_email, booking_details)

    return render(request, 'reserve/booking_confirmation.html', {'booking': booking})
```

```python
# Contact Us View
def contact_us(request):
    if request.method == 'POST':
        name = request.POST.get('name')
        email = request.POST.get('email')
        subject = request.POST.get('subject')
        message = request.POST.get('message')

        if name and email and subject and message:
            contact = Contact(name=name, email=email, subject=subject, message=message)
            contact.save()
            messages.success(request, 'Thank you for contacting us. We will get back to you s
oon.')
            return redirect('contact-us')
        else:
            messages.error(request, 'Please fill in all fields.')

    return render(request, 'reserve/contact_us.html')

# Update User Profile
@login_required
@require_POST
def update_profile(request):
    user = request.user
    first_name = request.POST.get('first_name')
    last_name = request.POST.get('last_name')
    email = request.POST.get('email')
    password = request.POST.get('password')
    confirm_password = request.POST.get('confirm_password')

    # Update user details
    user.first_name = first_name
    user.last_name = last_name
    user.email = email

    if password and password == confirm_password:
        user.set_password(password)

    user.save()

    return JsonResponse({'status': 'success'})

# View All Schedules with Sorting and Pagination
def schedule_list(request):
    # Update status of schedules behind today's date and time
    now = timezone.now()
    Schedule.objects.filter(schedule__lt=now).update(status='2')

    # Sorting logic based on query parameter 'sort'
    sort_by = request.GET.get('sort')
    if sort_by:
        if sort_by.endswith('_asc'):
            field = sort_by[:-4]  # Remove '_asc' suffix
            schedules_list = Schedule.objects.order_by(field)
        elif sort_by.endswith('_desc'):
            field = sort_by[:-5]  # Remove '_desc' suffix
            schedules_list = Schedule.objects.order_by(f'-{field}')
        else:
            schedules_list = Schedule.objects.all()
    else:
        schedules_list = Schedule.objects.all()

    # Pagination
    paginator = Paginator(schedules_list, 10)
    page_number = request.GET.get('page')
    schedules = paginator.get_page(page_number)

    context = {
        'schedules': schedules,
        'sort_by': sort_by
    }
    return render(request, 'reserve/schedule_list.html', context)

```

```python
# View All User Booking with Pagination
@login_required
def user_bookings(request):
    bookings_list = Booking.objects.filter(name=request.user.first_name)
    paginator = Paginator(bookings_list, 5)

    page_number = request.GET.get('page')
    bookings = paginator.get_page(page_number)
    return render(request, 'reserve/user_bookings.html', {'bookings': bookings})

# View all Categories
@login_required
def all_categories(request):
    categories_list = Category.objects.all()
    paginator = Paginator(categories_list, 5)

    page_number = request.GET.get('page')
    categories = paginator.get_page(page_number)
    return render(request, 'reserve/all_categories.html',{'categories':categories})

# View all Buses with sorting and pagination
@login_required
def all_buses(request):
    query = request.GET.get('q')
    if query:
        buses_list = Bus.objects.filter(
            Q(bus_number__icontains=query) | Q(category__name__icontains=query)
        )
    else:
        buses_list = Bus.objects.all()

    paginator = Paginator(buses_list, 5)

    page_number = request.GET.get('page')
    buses = paginator.get_page(page_number)
    return render(request, 'reserve/bus_list.html', {'buses': buses, 'query': query})

# View all Locations with sorting
@login_required
def all_locations(request):
    query = request.GET.get('q')
    sort = request.GET.get('sort', 'name_asc')

    if query:
        locations_list = Location.objects.filter(location__icontains=query)
    else:
        locations_list = Location.objects.all()

    if sort == 'name_asc':
        locations_list = locations_list.order_by('location')
    elif sort == 'name_desc':
        locations_list = locations_list.order_by('-location')
    elif sort == 'id_asc':
        locations_list = locations_list.order_by('id')
    elif sort == 'id_desc':
        locations_list = locations_list.order_by('-id')

    paginator = Paginator(locations_list, 10)
    page_number = request.GET.get('page')
    locations = paginator.get_page(page_number)

    return render(request, 'reserve/location_list.html', {'locations': locations, 'query': qu
ery})
```

```python
# View for canceling a booking
@login_required
def cancel_booking(request):
    booking_id = request.GET.get('id')
    booking = get_object_or_404(Booking, id=booking_id)

    if booking.status == '1':
        booking.status = '3'
        booking.save()

        from_location = booking.schedule.depart
        to_location = booking.schedule.destination
        booking_details = f"""
            Booking Code: {booking.code}
            Name: {booking.name}
            From: {from_location}
            To: {to_location}
            Schedule: {booking.schedule.schedule.strftime('%Y-%m-%d %H:%M')}
            Seats: {booking.seats}
            Total Payable: {booking.total_payable()}
            Status: Canceled
            Date Created: {booking.date_created.strftime('%Y-%m-%d %H:%M')}
            """
        # Send email
        user_email = request.user.email
        send_booking_confirmation(user_email, booking_details)
        messages.success(request, 'Booking has been successfully cancelled.')
    else:
        messages.error(request, 'Only pending bookings can be cancelled.')

    return redirect('user-bookings')

# View for delete a booking
@login_required
def delete_booking(request):
    booking_id = request.GET.get('id')
    booking = get_object_or_404(Booking, id=booking_id)

    if booking.status != '1':
        booking.delete()
        messages.success(request, 'Booking has been successfully deleted.')
    else:
        messages.error(request, 'Pending bookings cannot be deleted. Please cancel them instead.')

    return redirect('user-bookings')

```

3. utils/email_utils.py

The email_utils.py file contains utility functions that are used across different parts of the project. These functions help keep the codebase organized and avoid redundancy. In the Busify project, utility functions include:

- **Email Sending Function**: A function to send booking confirmation emails to users after they book a trip.

```python
from django.core.mail import send_mail
from django.conf import settings

def send_booking_confirmation(user_email, booking_details):
    subject = 'Busify | Booking Confirmation'
    message = f'Thank you for your booking.\n\nHere are the booking details:\n{booking_details}\n\nThank You.\nWish you a Happy Journey\nTeam Busify'
    email_from = settings.EMAIL_HOST_USER
    recipient_list = [user_email]
    send_mail(subject, message, email_from, recipient_list)
```

4. urls.py

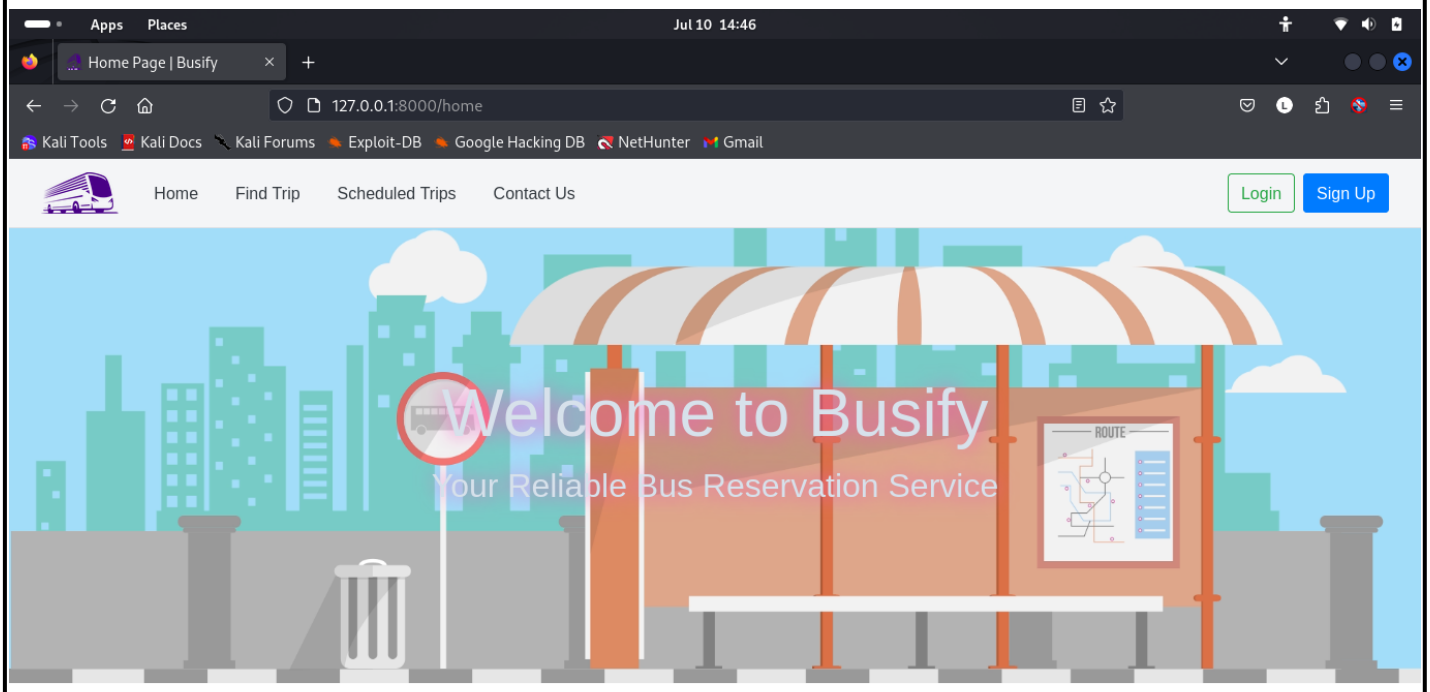
In the Busify project, the urls.py file plays a critical role in defining how users interact with the various pages and functionalities of the application.

```python
from django.urls import path
from . import views

urlpatterns = [
    path('', views.homePage),
    path('home', views.homePage, name='home'),
    path('signup', views.signup_view, name='signup'),
    path('login', views.login_view, name='login'),
    path('logout/', views.logout_view, name='logout'),
    path('find-trips/', views.find_trips, name='find-trips'),
    path('book-trip/<int:schedule_id>/', views.book_trip, name='book-trip'),
    path('booking-confirmation/<int:booking_id>/', views.booking_confirmation, name='booking-confirmation'),
    path('contact-us/', views.contact_us, name='contact-us'),
    path('update-profile/', views.update_profile, name='update-profile'),
    path('all_schedules', views.schedule_list, name="schedule-list"),
    path('user/bookings/', views.user_bookings, name='user-bookings'),
    path('bus/categories', views.all_categories, name='all-categories'),
    path('buses/list', views.all_buses, name='all-buses'),
    path('locations/list', views.all_locations, name='all-locations'),
]

```

```python
from django.contrib import admin
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include("reservationApp.urls")),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Some Figures from this project :

1. Home Page
Fig.1:-



Discover the Joy of Travel with Busify

Fig.2:-

At Busify, we believe in making travel easy and accessible for everyone. With our extensive network of routes and modern fleet, you can explore new destinations and create unforgettable memories. Our user-friendly platform ensures a seamless booking experience, so you can focus on what matters most - your journey.

Fig.3 :-

Why Choose Us?

Extensive Network of Routes

Explore our extensive network of routes covering major cities and remote destinations. From bustling urban centers to serene countryside, we connect you to where you need to be.

Flexible Booking Options

Choose from a range of flexible booking options tailored to suit your schedule. Whether you prefer to plan ahead or book last-minute, we have options to accommodate your needs.

Modern and Comfortable Fleet

Travel in comfort and style with our modern fleet of buses equipped with amenities designed to enhance your journey. Sit back, relax, and enjoy the ride.

How It Works

1. Browse Routes

Explore our route map and discover where we can take you.

2. Select Your Journey

Choose your departure and arrival locations, select travel dates, and

3. Book Your Ticket

Secure your seat by completing the easy booking process online.

4. Travel with Confidence

Arrive at your destination comfortably and on time, knowing

Fig.4 :-

What Our Customers Say

"Busify made booking my trip so easy! The buses were comfortable and the service was excellent. Highly recommended!"

- John Doe

"I travel frequently for work, and Busify has become my go-to choice for bus reservations. Convenient, reliable, and affordable!"

- Jane Smith

"Amazing service! The support team was very helpful when I needed to change my travel dates. Will definitely use Busify again."

- Michael Johnson

Start Your Journey Today

Ready to embark on your next adventure? Begin your hassle-free booking experience with Busify.

Book Now

Fig.5 :-

Latest News

New Routes Added

We've expanded our service to include new routes, making it easier to travel to your favorite destinations.

Special Offers

Check out our latest special offers and discounts on bus tickets. Don't miss out on great savings!

Featured Destinations



Fig.6 :-

Kashmir

Experience the picturesque valleys and serene lakes of Kashmir with Busify's convenient travel options.

Ladakh

Explore the rugged landscapes and monasteries of Ladakh with Busify's eco-friendly buses and great service.

Goa

Relax on the sandy beaches and enjoy the vibrant nightlife of Goa with Busify's affordable travel options.

Frequently Asked Questions

How can I book a ticket with Busify?

Booking a ticket with Busify is easy! Simply visit our website, select your desired route, choose your travel dates, and follow the prompts to complete your booking.

Fig.7 :-

Is there a loyalty program for frequent travelers?

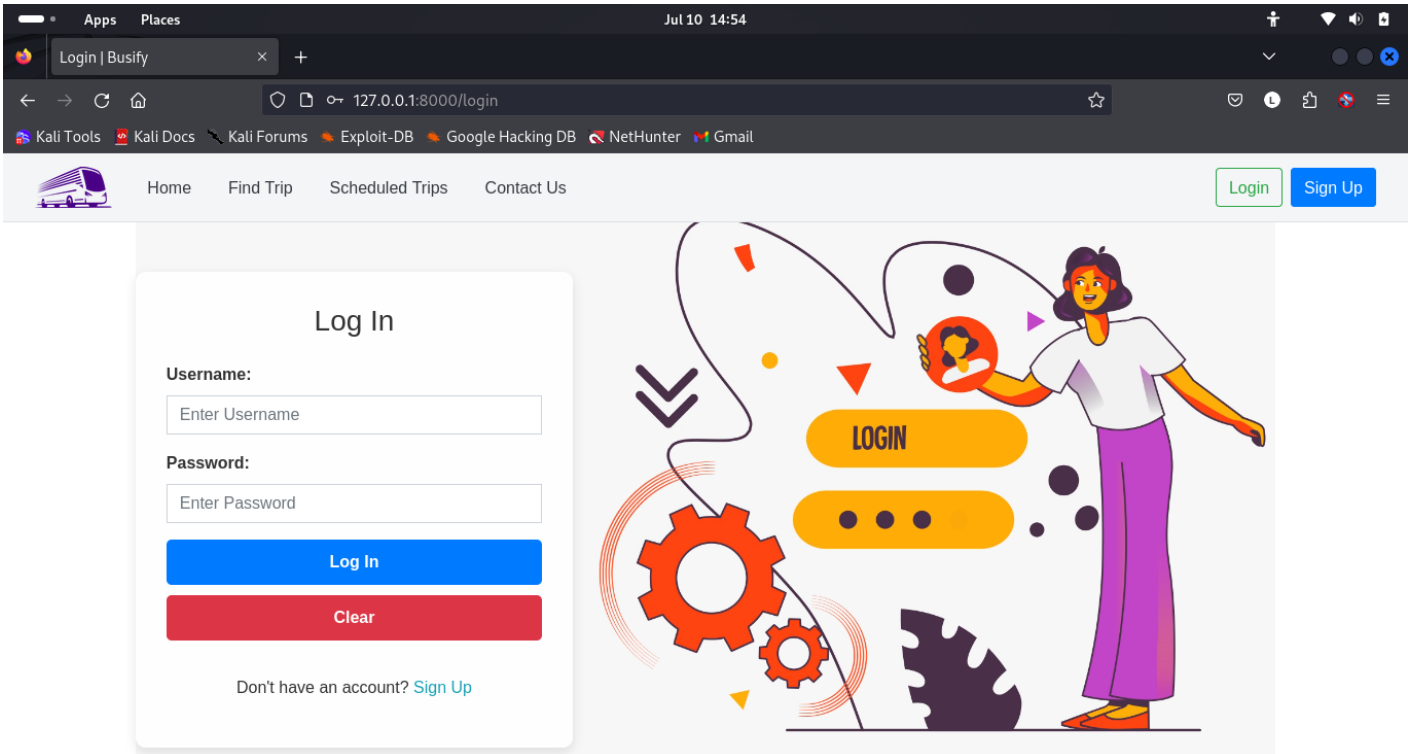
How can I contact customer support?

Stay Connected

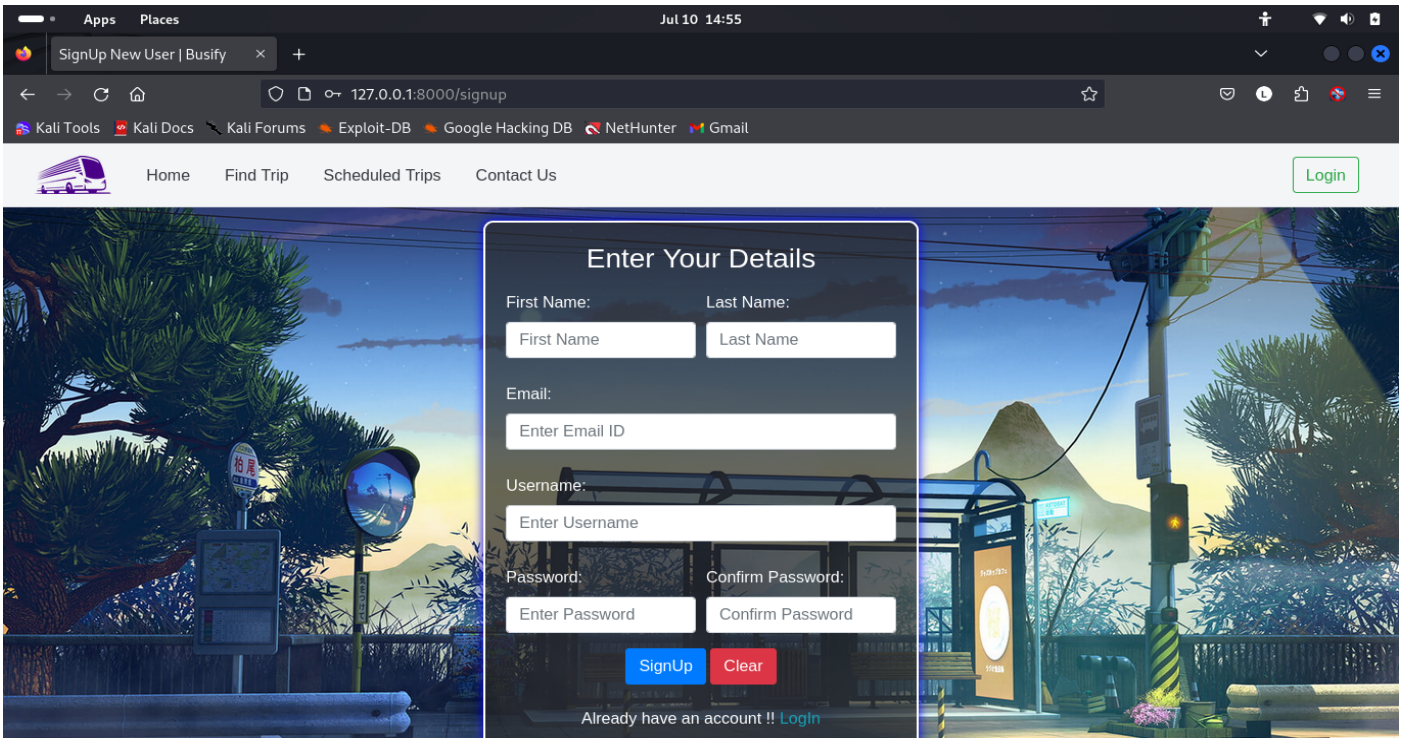
Follow us on social media and sign up for our newsletter to receive updates, special offers, and travel tips directly to your inbox.

Instagram Facebook GitHub

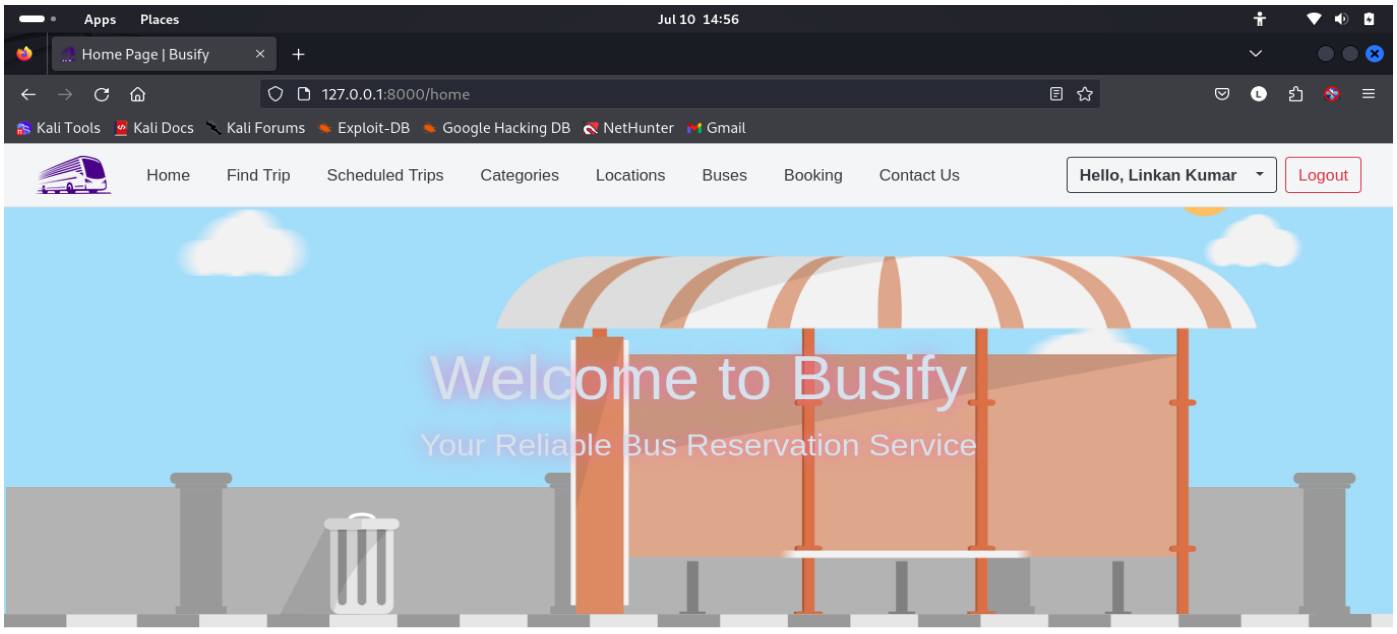
2. Log-In Page



3. Sign-Up Page

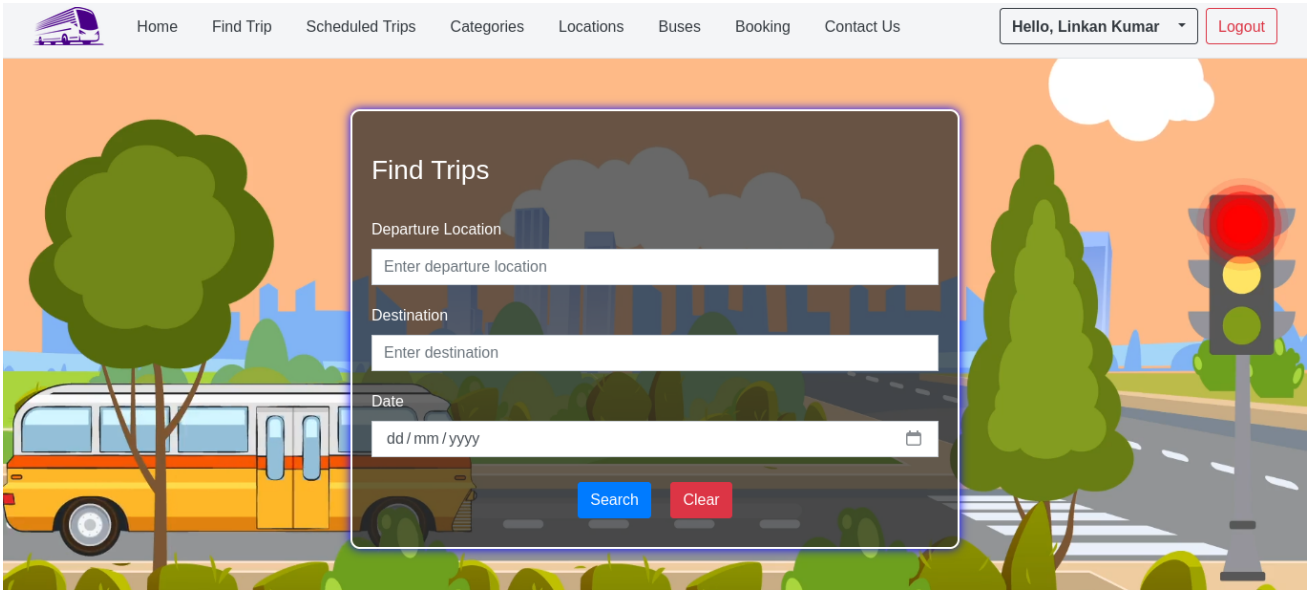


4. User Home Page

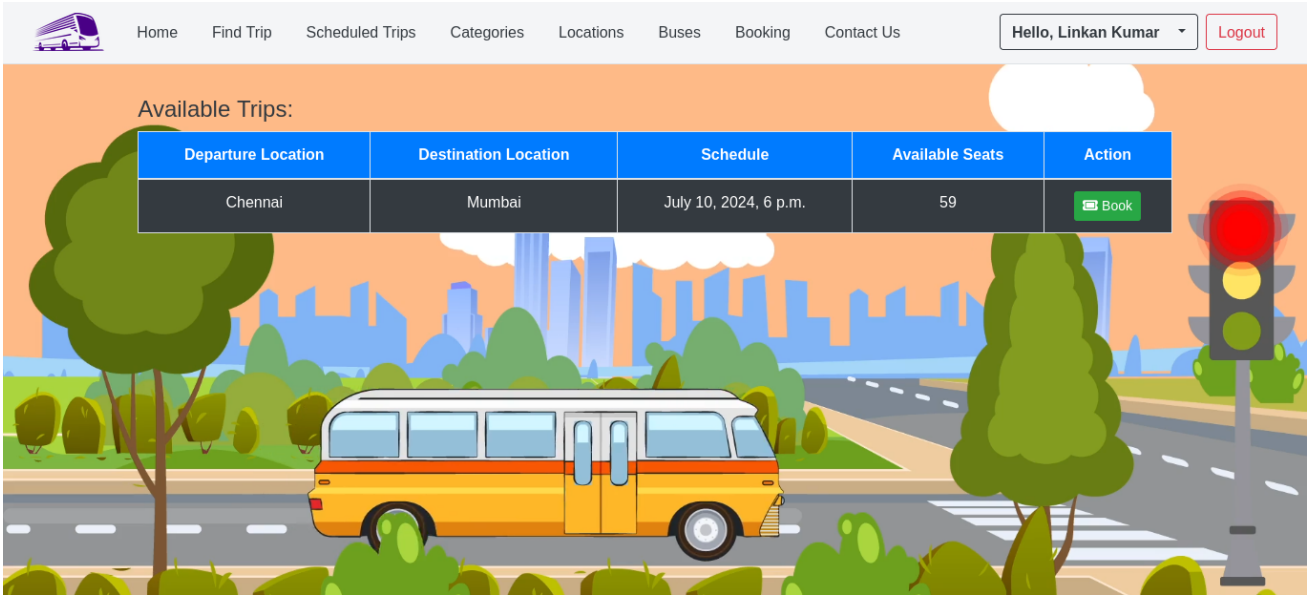


Discover the Joy of Travel with Busify

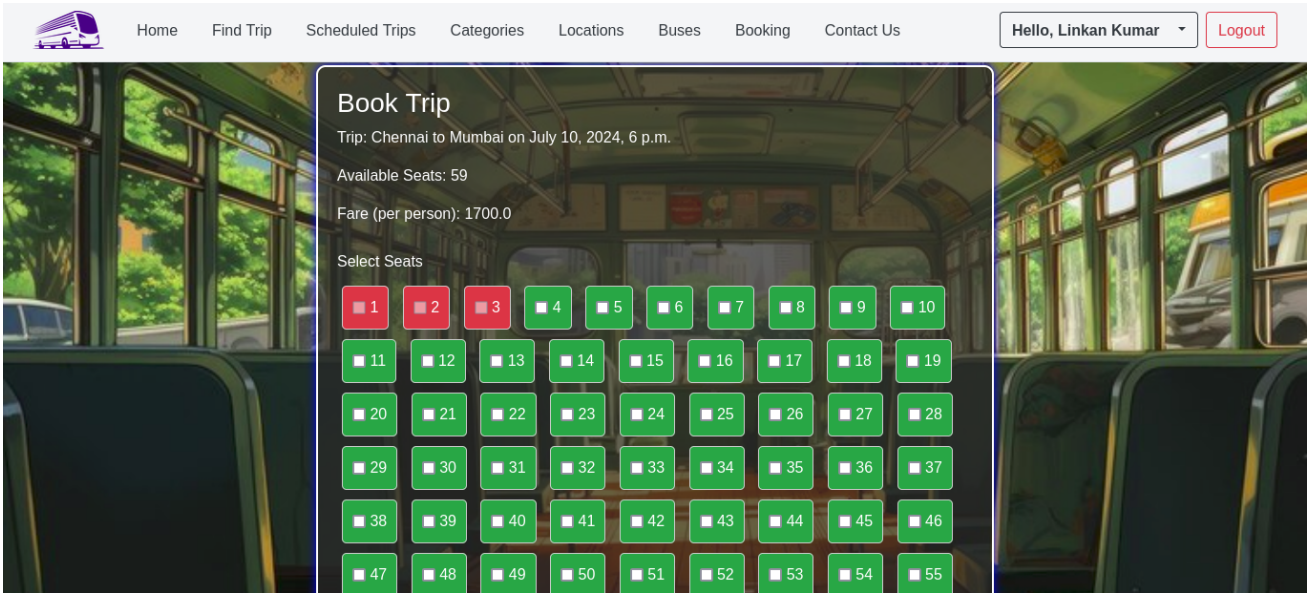
5. Find Trips Page



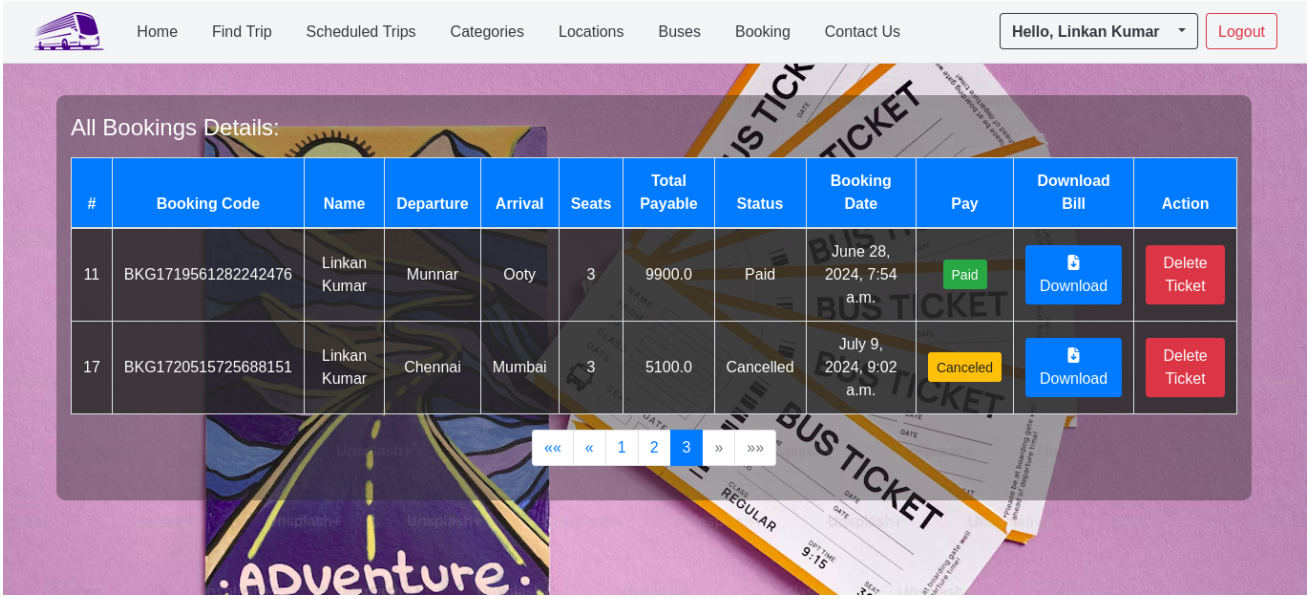
6. Available Trips



7. Book Trip Page



8. User Booking Page



6. Challenges Faced

1. Integrating PayPal's payment gateway presented several validation errors, especially concerning currency format and data consistency. Understanding and implementing the PayPal SDK correctly was initially challenging.

2. Implementing real-time notifications for booking confirmations was essential for user satisfaction but posed challenges in ensuring timely and reliable delivery.

3. Implementing an interactive seat selection system that accurately reflected real-time seat availability was complex, especially with multiple users accessing the system simultaneously.

7. Future Enhancements

1. Mobile Application Development:

- **Enhancement:** Develop a native mobile application for both iOS and Android platforms.
- **Benefit:** Enhances user accessibility and convenience, allowing users to book tickets and manage their reservations on the go.

2. Real-Time GPS Tracking:

- **Enhancement:** Integrate real-time GPS tracking for buses.
- **Benefit:** Provides users with live updates on bus locations, estimated arrival times, and delays, improving the overall travel experience.

3. Multi-Language Support:

- **Enhancement:** Add support for multiple languages.
- **Benefit:** Expands the user base by making the application accessible to non-English speaking users, enhancing usability and user satisfaction.

4. Loyalty Program:

- **Enhancement:** Implement a loyalty program to reward frequent travelers with discounts and special offers.
- **Benefit:** Encourages repeat bookings, builds customer loyalty, and increases user engagement.

5. Advanced Analytics and Reporting:

- **Enhancement:** Develop advanced analytics and reporting tools for administrators.
- **Benefit:** Provides insights into booking patterns, peak travel times, and revenue trends, aiding in better decision-making and strategic planning.

6. Integration with Other Travel Services:

- **Enhancement:** Partner with other travel services like airlines, hotels, and car rentals to offer bundled deals.
- **Benefit:** Provides users with comprehensive travel solutions, enhancing convenience and potentially increasing bookings.

7. Enhanced Security Measures:

- **Enhancement:** Implement advanced security measures such as two-factor authentication and enhanced data encryption.
- **Benefit:** Ensures user data security and builds trust, reducing the risk of data breaches and fraud.

8. AI-Powered Chatbot:

- **Enhancement:** Introduce an AI-powered chatbot to assist users with booking inquiries and support.
- **Benefit:** Provides instant customer service, improving user satisfaction and reducing the load on customer support teams.

8. Conclusion

In conclusion, the Busify project has successfully established a comprehensive bus reservation system that addresses the needs of modern travelers. Through a user-friendly interface and efficient backend architecture, the system streamlines the booking process, enhances user experience, and ensures secure transactions. Despite challenges faced during development, such as integrating payment gateways and maintaining data security, the project has emerged as a robust solution in the transportation sector. With a clear vision for future enhancements, including mobile application development and real-time GPS tracking, Busify is well-positioned to adapt to evolving user demands and technological advancements, paving the way for continued growth and innovation in the bus reservation landscape.

9. References

- **Django Documentation**: https://docs.djangoproject.com
- **Bootstrap Documentation**: https://getbootstrap.com/docs
- **PayPal Developer Documentation**: https://developer.paypal.com/docs
- **MDN Web Docs**: https://developer.mozilla.org
- **Stack Overflow**: https://stackoverflow.com