PROJECT REPORT:

AAPNI PARKING – CONVENIENT PARKING SYSTEM FOR YOUR VEHICLE

By

NEHAL GUPTA

ID: MST01-0019

Abstract

The AapniParking project aims to provide a comprehensive vehicle parking management website developed using Django framework, HTML, CSS. This report outlines the project's objectives, key features, technologies used, implementation details, challenges faced, and future enhancements.

Table of Contents

1. Introduction

The AapniParking project is a fully functional parking management website developed using Django web framework. It contains lots of functionalities like add new vehicle, book parking slot, view and manage vehicles, dashboard for users and similarly for admin all these functionalities is there with some more features like generate invoice, add parking space with a good UI to give an overall good experience to users.

2. Project Overview

The project integrates Django framework along with Bootstrap, HTML, CSS for frontend styling and JavaScript for responsiveness. The website have a functionality where on successful booking an email is send regarding booking with booking id and parking slot no to the user . The website key functionalities are responsive dashboard which provide information related to available parking slots, total parking etc, detail page which provide all the information related to it, CRUD operations on the parking booking, parking space, vehicle. A comprehensive Admin dashboard where admin can manage all the functionalities and perform CRUD operation.

3. Technologies Used

Django 5.0

Bootstrap 5

SQLite3 (default database engine)

H.T.M.L

C.S.S

JavaScript

jQuery

Django 5.0: Django is a high-level Python web framework that enables rapid development of secure and maintainable websites. It works on MVT architecture and provides built-in features for everything including Django Admin Interface, default database – SQLlite3.

Bootstrap 5: Bootstrap is a popular front-end framework that helps you create responsive and accessible web designs. It is a Powerful, extensible, and feature-packed frontend toolkit. It provides pre-designed templates and components using HTML, CSS, and JavaScript for creating good and responsive user interfaces.

SQLite: SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. SQLite implements a small, fast, self-contained, high reliability, full-featured, SQL database engine. It is the default database engine used in Django web framework.

HTML: HTML stands for HyperText Markup Language. It is the standard markup language used to create web pages. It is a combination of Hypertext and Markup language. It creates a standard structure and uses tags to display images, links, text and other multimedia content on the web.

CSS: CSS (Cascading Style Sheets) is simply designed language intended to simplify the process of making web pages presentable. It allows us to apply styles to HTML documents and make the pages look more presentable and styled. It describes how a webpage should look. It prescribes colors, fonts, spacing, etc. to enhance user interface look.

JavaScript: JavaScript is a lightweight interpreted (or just-in-time compiled) programming language which is use to add interactivity, dynamic behaviour, and functionality to web pages. It allows developers to manipulate the HTML and CSS of a webpage, handle user interactions or overall client side webpage behaviour.

jQuery: jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, and Ajax.

4. System Architecture

The system architecture follows a Django MVT (Model-View-Template) architecture. It is a software design pattern and is a collection of three important components Model View and Template. The model defines the database and data structure and maintain the data, views is responsible for handling user requests and returning response as per the request and templates is responsible for rendering HTML pages.

5. Implementation Details

Home Page/Dashboard: Dynamically fetches data from the database and display it according to the user logged in if the user is customer then it will show data according to that or if the user is admin then it will fetch and show data according to admin. It provides available parking slot numbers dynamically. Provides a user-friendly interface, show table and provide search functionality in the table to look for data in table and have a small section of contact us where the admin no is shown and admin email id is shown

Add Parking Space Page: It renders a form which allows admin to add a new parking space. It user interface is created using html, bootstrap, CSS.
Bookparking page: It also renders a form to both admin and user who is logged in and provide the functionality to book a parking space for a vehicle.

AddVehicle page: It too renders a form for adding vehicle to user and admin using bootstrap, html, CSS. This functionality keeps user free from the hassle of adding the details of their vehicle again and again wherever they park their vehicle as it stores the vehicle detail and displays all the vehicle registered in the bookparking form.

Detail Page: It displays all the information related to parking space or parking booking or vehicle .It is a detailed view and have functionalities of editing or modifying the details.

Admin functionality in which there are many features given to admin through which he can manage the vehicles , parking and parking space. Admin have the option to perform CRUD operations on the data.

Login page: The login page provides the feature of login through Django pre built authentication system and login the user and let it use different features.

Registration page: A person can create a user account and registered oneself. The registration page uses the Django prebuilt user model to register the user and save it in database. The registration page form is created using bootstrap framework, Html, CSS and makes it user-friendly and responsive.

User Detail page: The user detail page dynamically fetch data and displays the logged in user details. It uses Html and CSS for making the user interface part attractive and presentable.

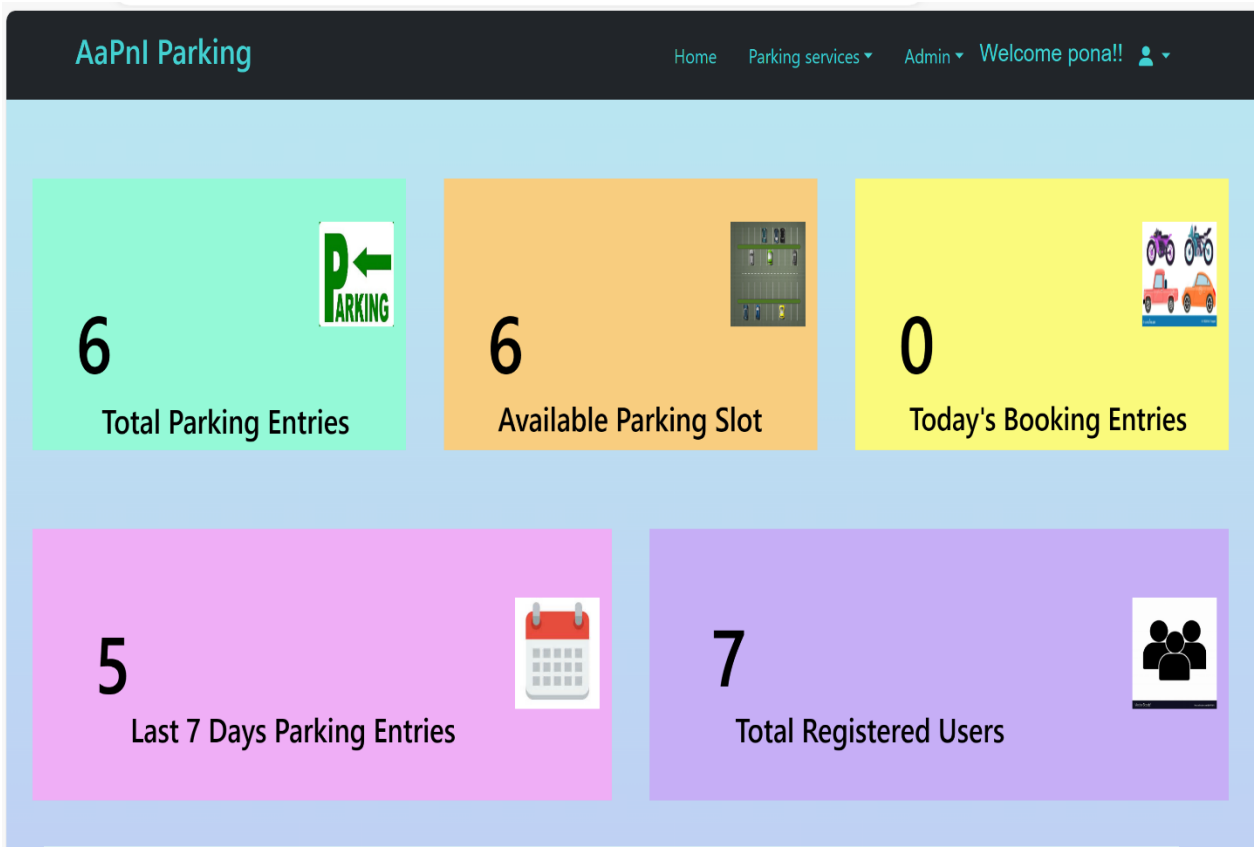


Fig 1(a): Home Page (admin)

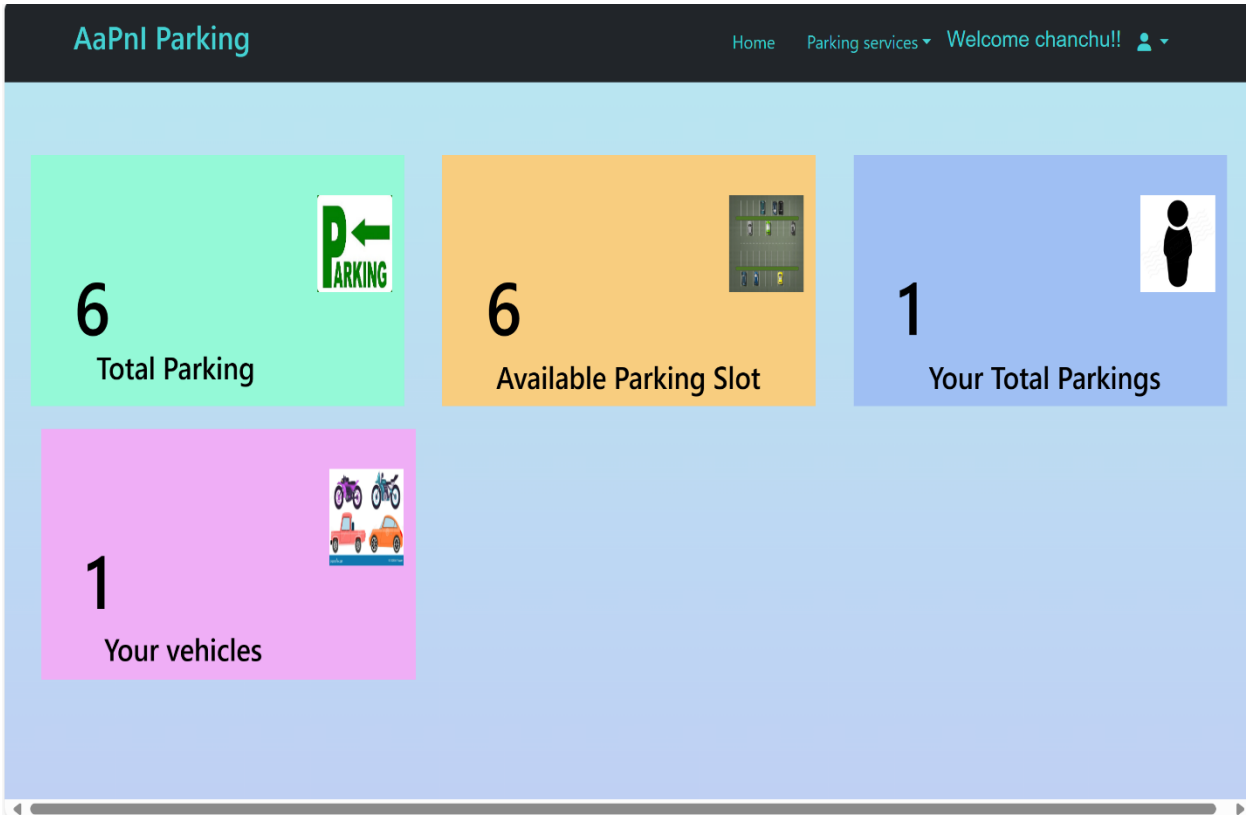


Fig 1(b): Home Page (users)

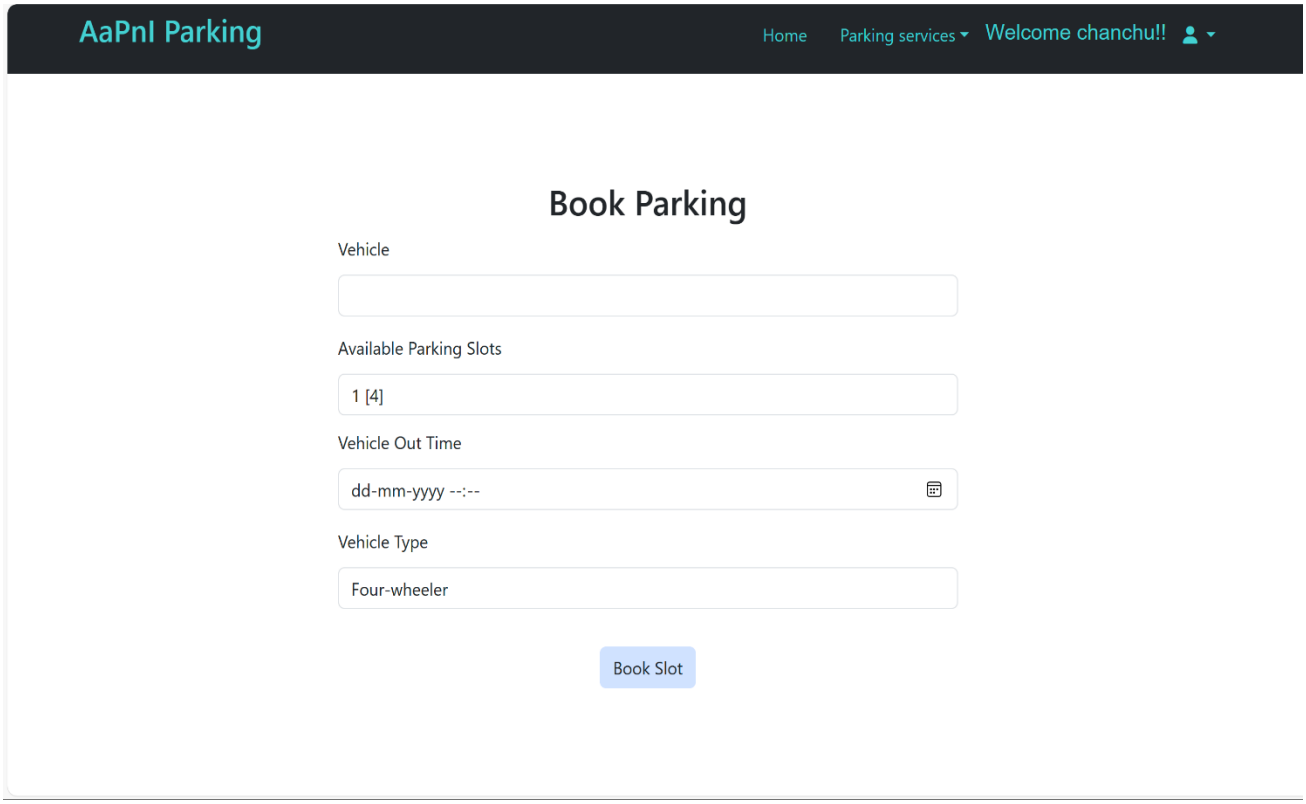


Fig 2: Book Parking Slot Page

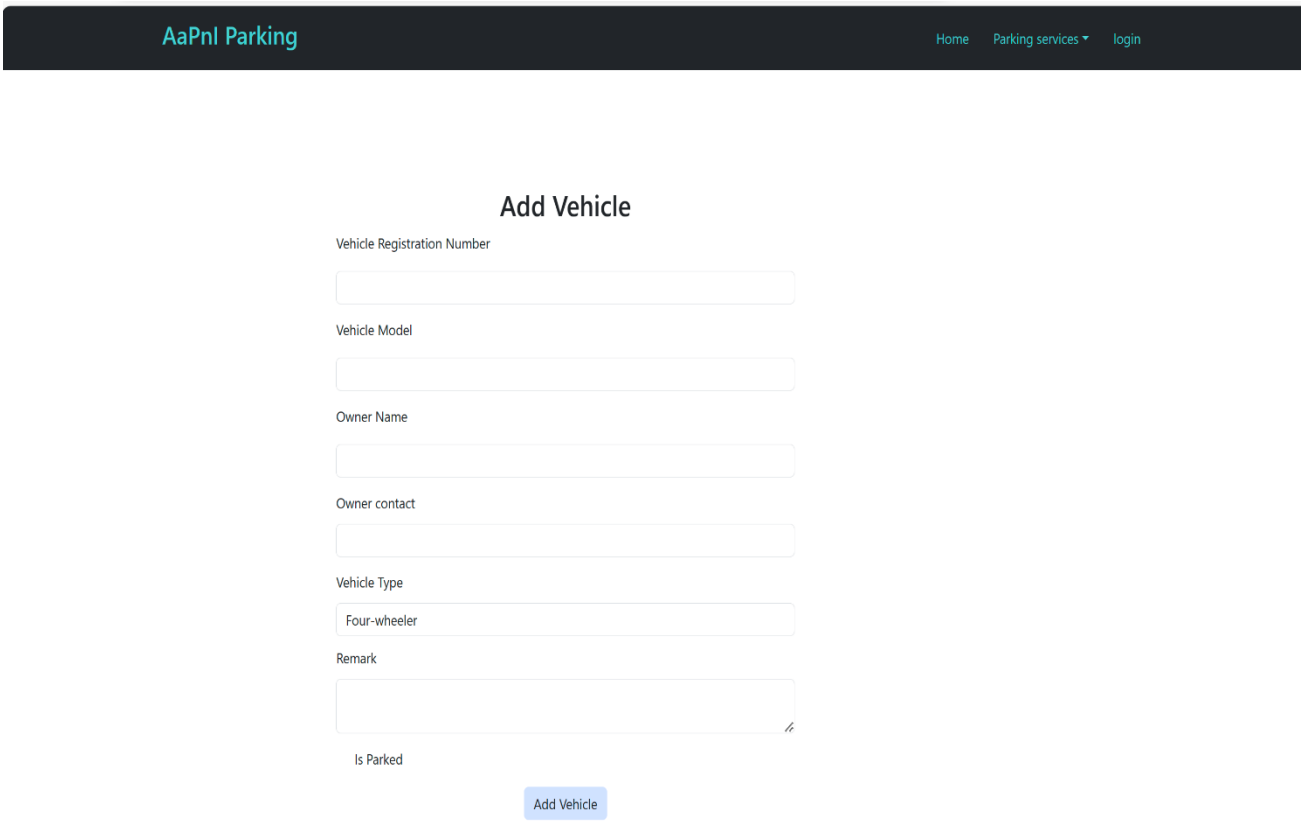


Fig 3: Add Vehicle Page

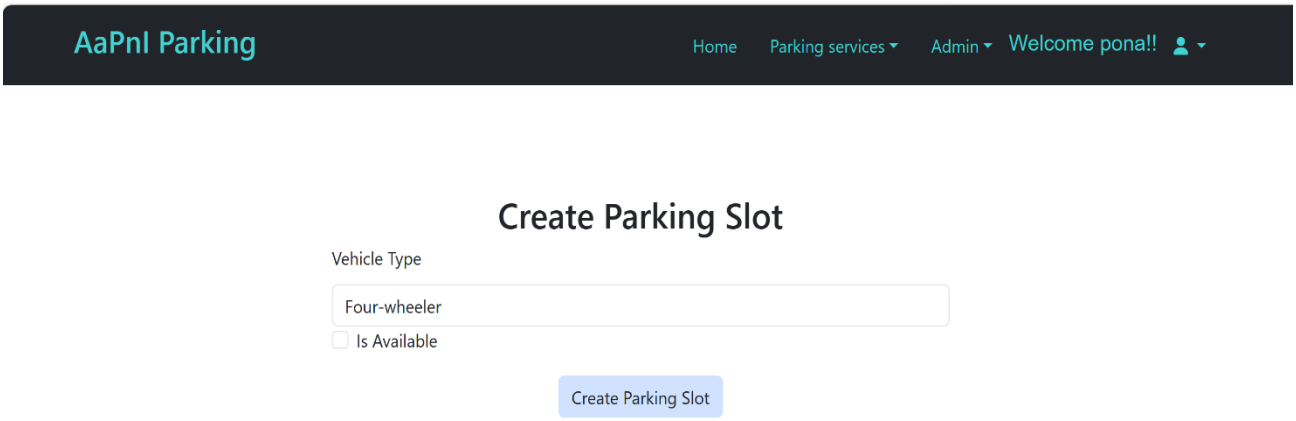


Fig 4: Add Parking Space

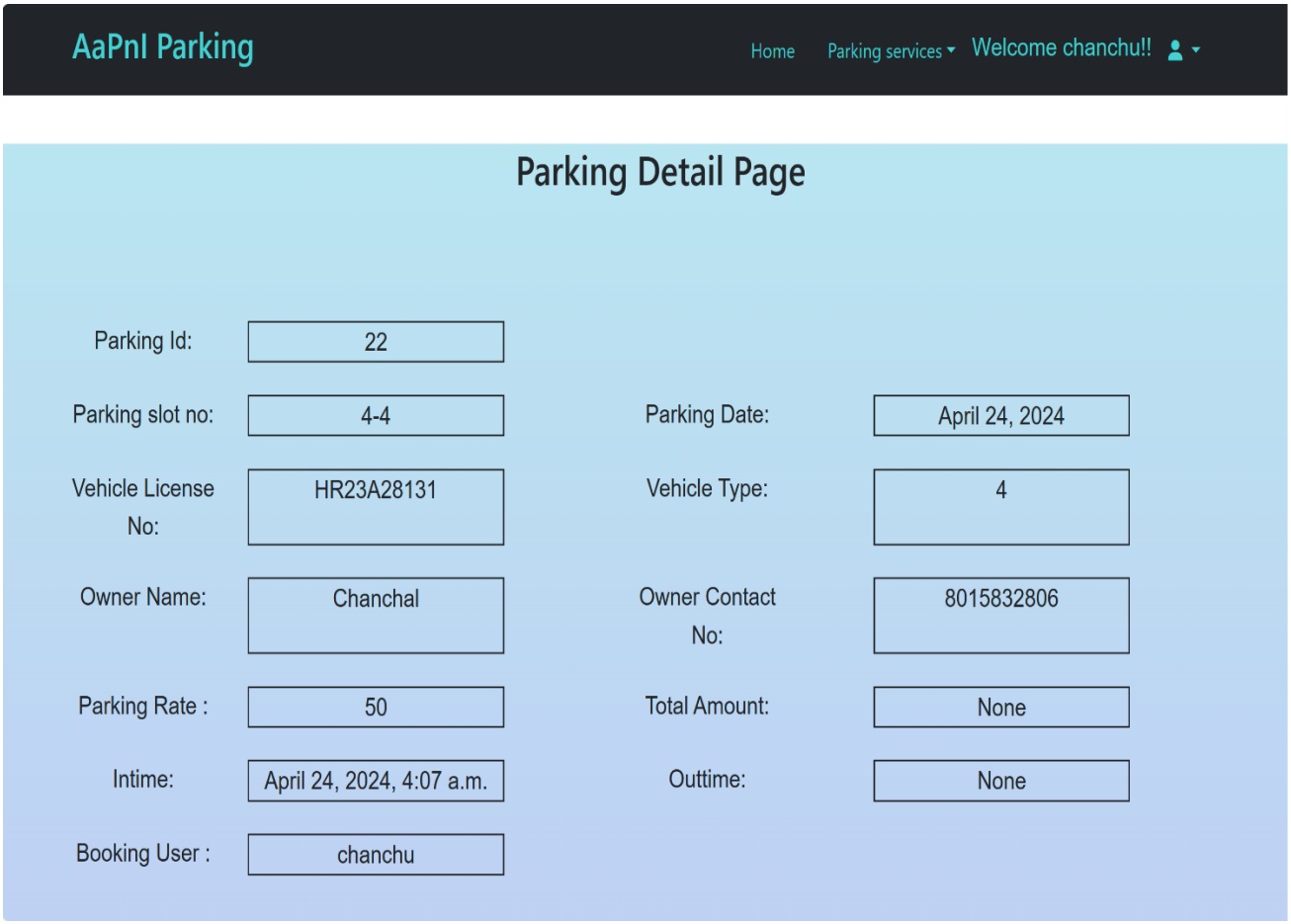


Fig 5: Detail parking page

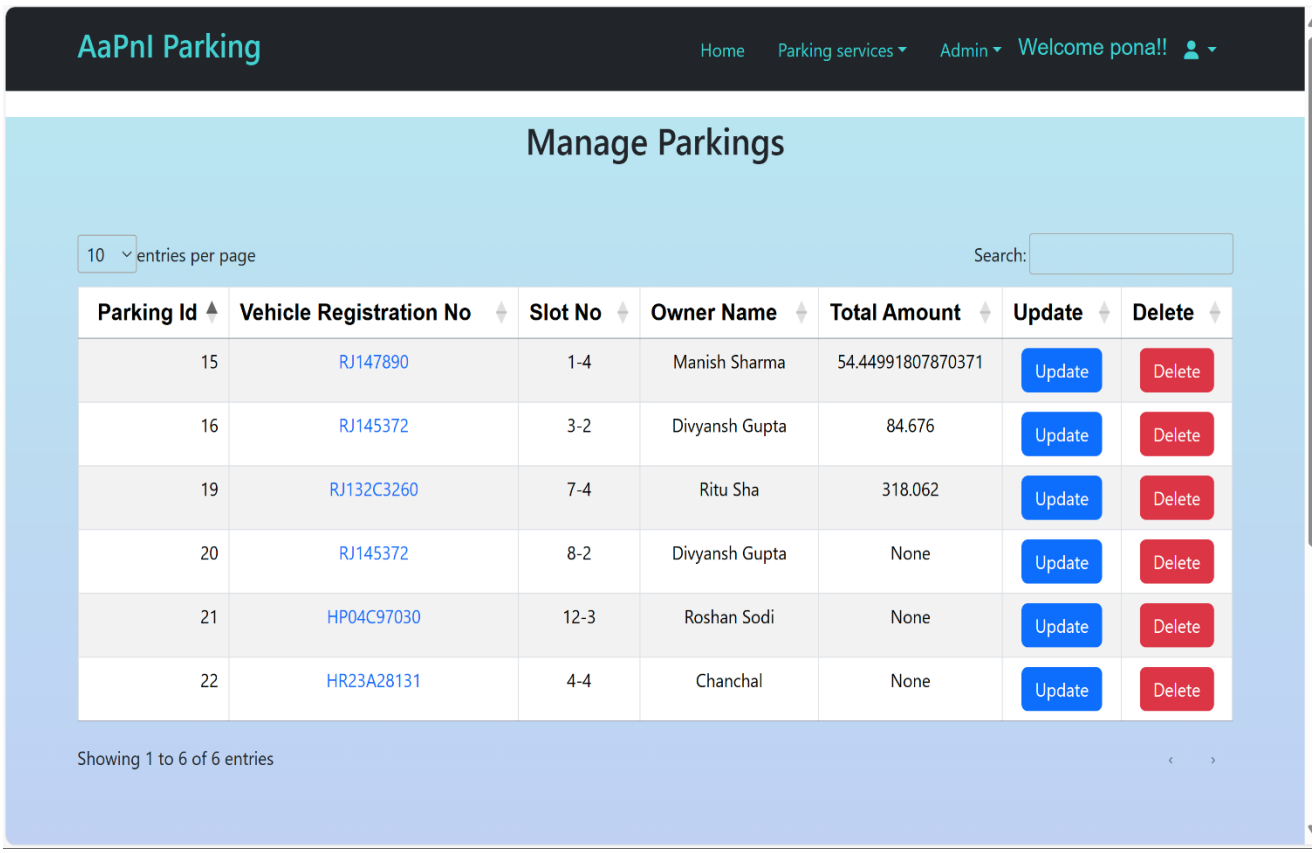


Fig 6: Manage Parking Page

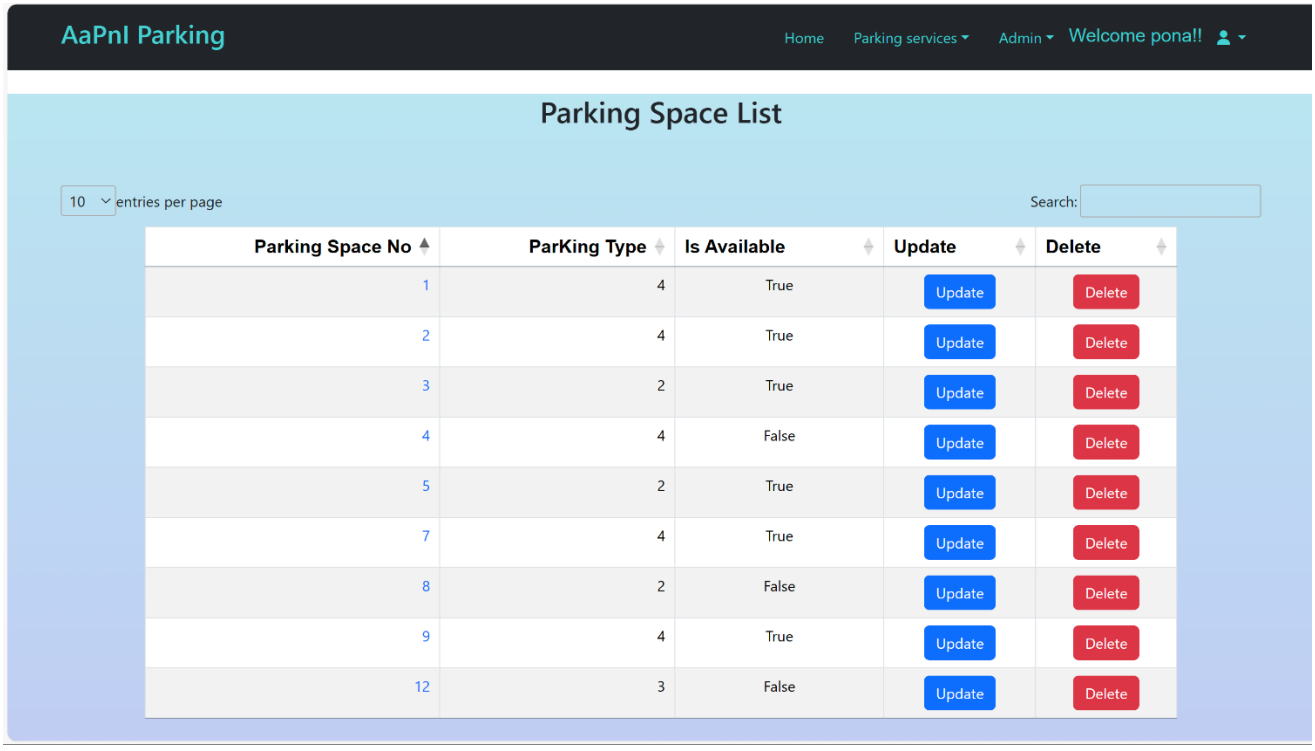


Fig 7: View Parking Space page

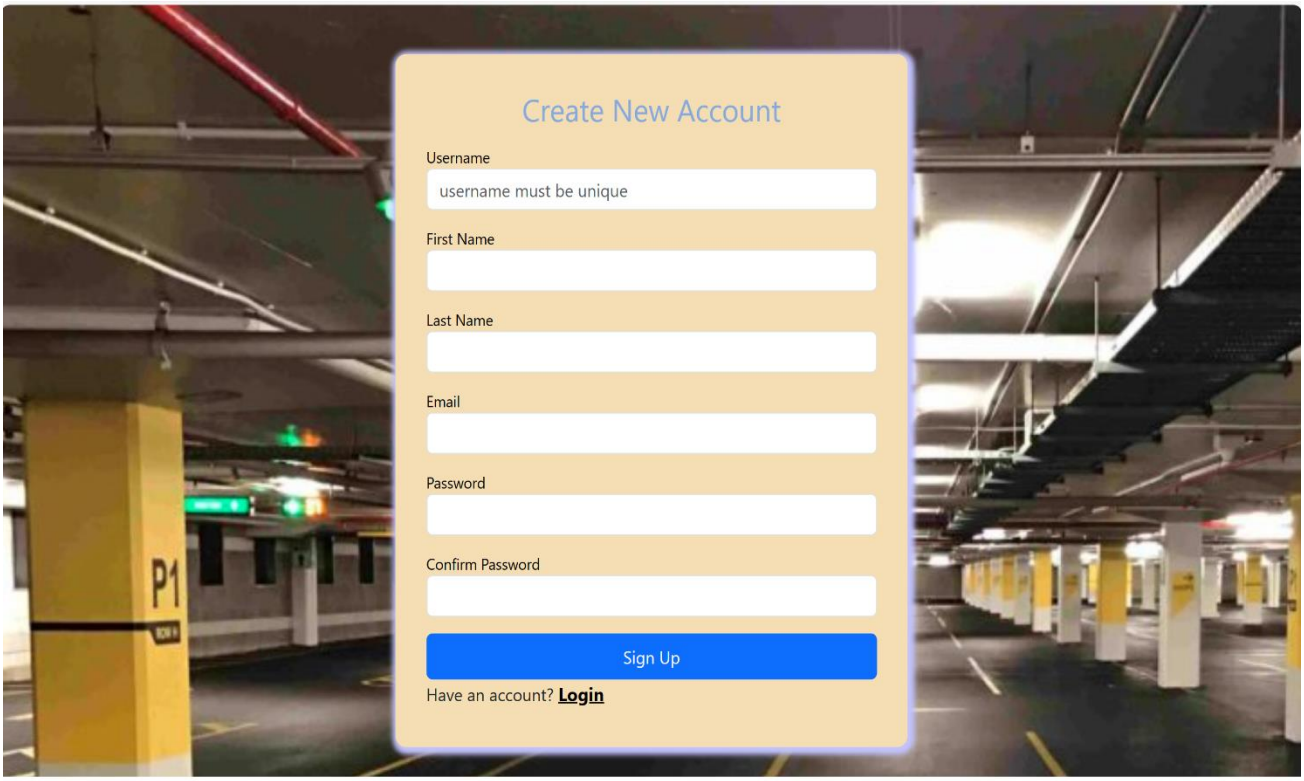


Fig 8: User Registration Page

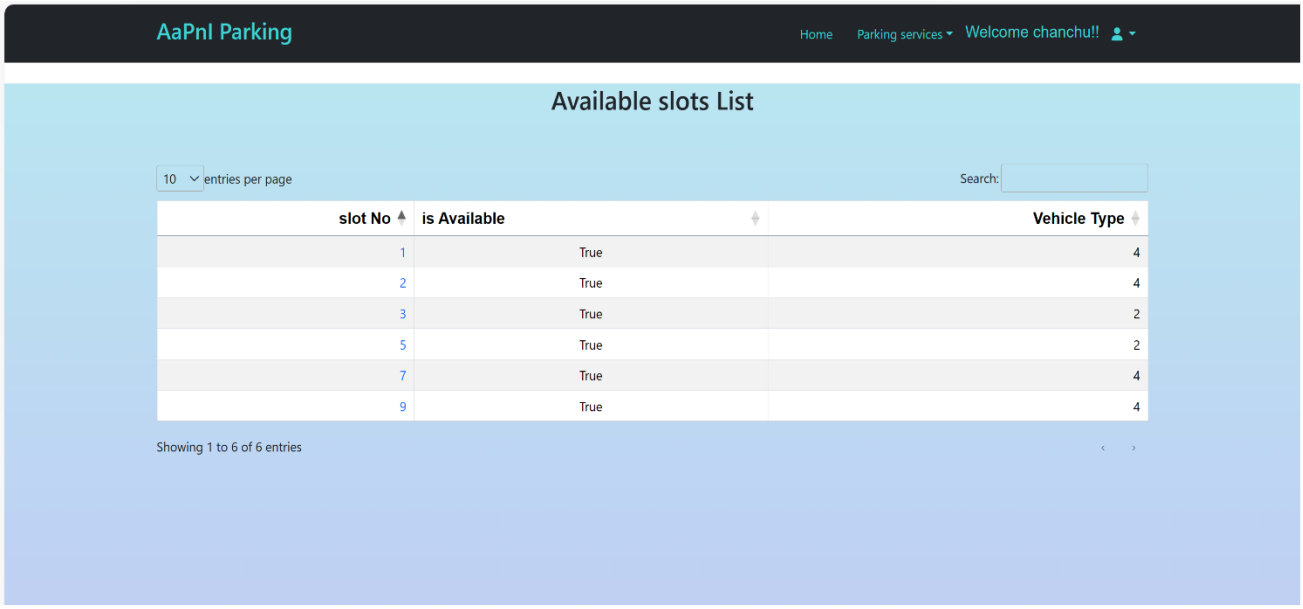


Fig 9: Available Parking Slots Page

Home Page/Dashboard Page:

Functionality:

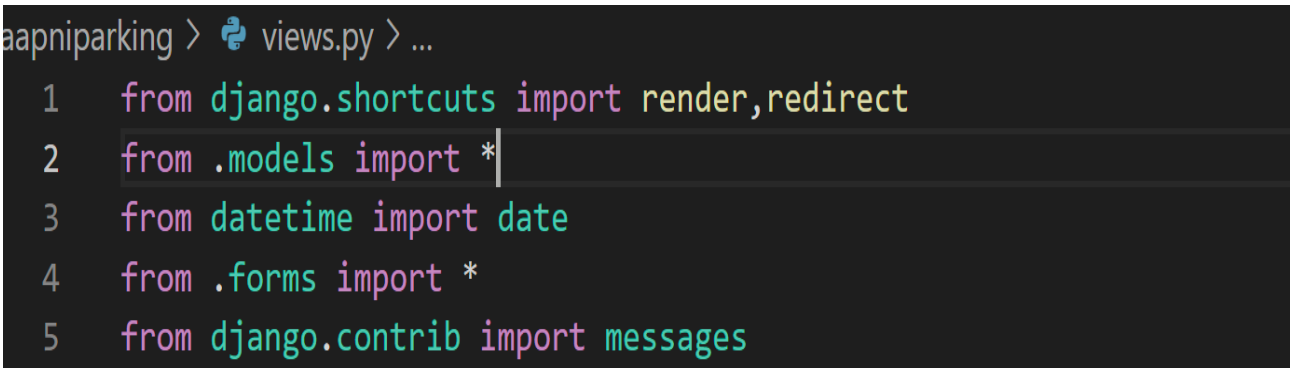
Dynamically displays values according to the user logged is customer or whether admin with a good interactive user interface which is built using Django modules.

Technologies Used:

Frontend: HTML, CSS, JavaScript , Bootstrap

Backend: Django

Views.py



```
rking > views.py > bookingslot
def home(request):
    space=ParkingSpace.objects.filter(is_available=True)
    count_space=space.count()
    t_space=ParkingBookSlot.objects.all().count()
    parking=ParkingBookSlot.objects.filter(parking_date=date.today())
    if request.user.is_authenticated:
        if request.user.is_staff:
            total_users=User.objects.all().count()
            c_date=date.today()
            pdate=str(c_date)
            if c_date.month==1:
                p_y=c_date.year-1
                p_m=12
            else:
                p_y=c_date.year
            if c_date.day<=7 and c_date.month in [1,3,5,7,8,10,12]:
                p_m=c_date.month-1
                p_d=31+(c_date.day-7)
            elif c_date.day<=7 and c_date.month==2:
                p_m=c_date.month-1
                p_d=28+(c_date.day-7)
            elif c_date.day<=7 and c_date.month in [4,6,9,11]:
                p_m=c_date.month-1
                p_d=30+(c_date.day-7)
            else:
                p_d=c_date.day
                p_m=c_date.month
            p_d=c_date.day-7
            p_date1=date(p_y,p_m,p_d)
            p_date1=str(p_date1)
            app=ParkingBookSlot.objects.filter(parking_date__range=[p_date1, pdate])
```

```
            else:
                p_d=c_date.day
                p_m=c_date.month
            p_d=c_date.day-7
            p_date1=date(p_y,p_m,p_d)
            p_date1=str(p_date1)
            app=ParkingBookSlot.objects.filter(parking_date__range=[p_date1, pdate])
            days=app.count()
            parking=parking.count()
            context={'total':total_users,'available':count_space,'total_slots':t_space,'day':days,'
        else:
            user_parking=ParkingBookSlot.objects.filter(booking_user=request.user.id)
            park_count=user_parking.count()
            context={'available':count_space,'total_slots':t_space,'my_parking':park_count}
    else:
        context={'available':count_space,'total_slots':t_space}
    return render(request,'dashboard.html',context)
```

Book Parking Slot Page:

Functionality:

Renders a html form which takes input with responsive layout with bootstrap styling.

Technologies Used:

Frontend: HTML, CSS, Bootstrap

Backend: Django, Django Forms, Django Models

Model Fields:

vehicle_uniqueid=Foreign key

veh_model= CharField

slot_no=Foreign Key

owner_name= CharField

veh_registration= CharField

owner_contactno= CharField

parking_rate=IntegerField

intime=DateTimeField

outtime= DateTimeField

fee= FloatField

vehicle_type= CharField

booking_user=ForeignKey

parking_date= DateField

views.py

```
from django.shortcuts import render,redirect
from .models import *
from datetime import date
from .forms import *
from django.contrib import messages
from django.db.models import Q
from django.contrib.auth.decorators import login_required
```

```
@login_required
def bookingslot(request,sid=0):
    space=ParkingSpace.objects.filter(is_available=True)
    if not request.user.is_staff:
        vehicles=Vehicle.objects.filter(Q(created_by=request.user) & Q(parked=False))
        print(vehicles)
    else:
        vehicles=Vehicle.objects.filter(parked=False)
    space1=""
    if sid==0:
        id=False
    else:
        id=True
        space1=ParkingSpace.objects.get(no=sid)
    if request.method=="POST":
        form = addParkingForm(request.POST)
        if form.is_valid():
            instance=form.save(commit=False)
            if sid==0:
                slotins_no=request.POST['slot_no']
                slot_ins=ParkingSpace.objects.get(no=slotins_no)
                instance.slot_no=slot_ins
            else:
                instance.slot_no=space1
            instance.owner_name=instance.vehicle_uniqueid.owner
            instance.veh_model=instance.vehicle_uniqueid.veh_model
```

```
        instance.veh_model=instance.vehicle_uniqueid.veh_model
        instance.veh_registration=instance.vehicle_uniqueid.veh_registration_no
        instance.owner_contactno=instance.vehicle_uniqueid.contact
        userid=request.user
        instance.booking_user=userid
        instance.save()
        if not request.user.is_staff:
            subject = 'Parking Confirmed'
            message = f'Hi {userid.username}!!, Your booking slot is confirmed .Your parking slot_no is {instance.slot_no
            email_from = settings.EMAIL_HOST_USER
            recipient_list = [userid.email, ]
            send_mail( subject, message, email_from, recipient_list )
            messages.success(request, f"An email has been sent of booking confirmation on your registered email id")
        messages.success(request, f"Your booking slot is confirmed .Your parking slot_no is {instance.slot_no.no} and you
        return redirect('dashboard')
    else:
        print(form.errors)
        messages.error(request, "There was some issue while trying to book slot.Please try again later")
else:
    form= addParkingForm()
context={'slots':space,'reg_no':vehicles,'id':id,'update':False,'slot':space1}
return render(request,'bookparking.html',context)
```

Models.py

```
class ParkingBookSlot(models.Model):
    vehicle_uniqueid=models.ForeignKey(Vehicle,related_name='vehicle_uniqueid',on_delete=models.CASCADE)
    veh_model=models.CharField(max_length=255)
    slot_no=models.ForeignKey(ParkingSpace,on_delete=models.SET_NULL,null=True)
    owner_name=models.CharField(max_length=255)
    veh_registration=models.CharField(max_length=10)
    owner_contactno=models.CharField(max_length=10,null=True,blank=True)
    parking_rate=models.PositiveIntegerField()
    intime=models.DateTimeField(auto_now_add=True)
    outtime=models.DateTimeField(null=True,blank=True)
    fee=models.FloatField(null=True,blank=True)
    vehicle_type=models.CharField(max_length=50,choices=type,default="2")
    booking_user=models.ForeignKey(User,on_delete=models.CASCADE)
    parking_date=models.DateField(auto_now_add=True)
    def __str__(self):
        return f"{self.owner_name}"
    def save(self,*args,**kwargs):
        if self.vehicle_type=='4':
            self.parking_rate=50
        elif self.vehicle_type=='3':
            self.parking_rate=30
        elif self.vehicle_type=='2':
            self.parking_rate=20
        if self.outtime!=None:
            timediff=(self.outtime-self.intime).total_seconds()/60
            if timediff<=360:
                self.fee=self.parking_rate
            elif timediff>360:
                rate=self.parking_rate/(6*60)
                self.fee=round(rate*(timediff-360)+self.parking_rate,2)
```

```
            rate=self.parking_rate/(6*60)
            self.fee=round(rate*(timediff-360)+self.parking_rate,3)
        if self.outtime==None:
            ins=Vehicle.objects.get(id=self.vehicle_uniqueid.id)
            ins.parked=True
            space_ins=ParkingSpace.objects.get(no=self.slot_no.no)
            ins.save()
            if space_ins:
                space_ins.is_available=False
                space_ins.save()
        elif self.outtime:
            ins=Vehicle.objects.get(id=self.vehicle_uniqueid.id)
            space_ins=ParkingSpace.objects.get(no=self.slot_no.no)
            space_ins.is_available=True
            space_ins.save()
            ins.parked=False
            ins.save()
        super().save(*args,**kwargs)
```

forms.py

```
from .models import *
from django import forms
```

```
class addParkingForm(forms.ModelForm):
    class Meta:
        model=ParkingBookSlot
        fields=['vehicle_uniqueid','outtime','vehicle_type']
```

Add Parking Space Page:

Functionality:

Renders a html form which takes input with responsive layout and bootstrap styling for both user and admin

Technologies Used:

Frontend: HTML, CSS, Bootstrap

Backend: Django, Django Forms, Django Models

Model Fields:

no = BigAutoField

veh_type = CharField

is_available = BooleanField

views.py

```python
from .forms import *
from django.contrib import messages
from django.db.models import Q
from django.contrib.auth.decorators import login_required
```

```python
@login_required
def addParkingSpace(request):
    if request.method=="POST":
        form=createParkingSlot(request.POST)
        if form.is_valid():
            form.save()
            messages.success(request, "parking slot added successfully")
            return redirect('dashboard')
    else:
        form=createParkingSlot()
    context={'update':False}
    return render(request,'addparkingspace.html',context)
```

Models.py

```
type=[("4","4-wheeler"),("3","3-wheeler"),("2","2-wheeler")]

class ParkingSpace(models.Model):
    no = models.BigAutoField(unique=True,primary_key=True)
    veh_type = models.CharField(max_length=255,choices=type)
    is_available = models.BooleanField(default=True)

    def __str__(self):
        return f"{self.no}-{self.veh_type}"
```

forms.py

```
from .models import *
from django import forms

class createParkingSlot(forms.ModelForm):
    class Meta:
        model=ParkingSpace
        fields=['veh_type','is_available']
```

Add Vehicle Page:

Functionality:

Renders a html form which takes input with responsive layout and bootstrap styling for both user and admin.

Technologies Used:

Frontend: HTML, CSS, Bootstrap

Backend: Django, Django Forms, Django Models

Model Fields:

veh_registration_no= CharField

veh_model= CharField

owner= CharField

contact= CharField

vehicle_category= CharField

remark= TextField

created_by= ForeignKey

views.py

```
from django.shortcuts import render,redirect
from .models import *
from datetime import date
from .forms import *
from django.contrib import messages
from django.db.models import Q
from django.contrib.auth.decorators import login_required
```

```
@login_required
def addVehicle(request):
    if not request.user.is_authenticated:
        messages.info(request, f'You are not logged in .Please login in to add vehicle ')
        return redirect('login')
    users=User.objects.all()
    if request.method=="POST":
        form=addVehicleForm(request.POST)
        if form.is_valid():
            ins=form.save(commit=False)
            ins.created_by=request.user
            ins.save()
            if not request.user.is_staff:
                subject = 'Parking Confirmed'
                message = f'Hi {request.user.username}!!, Your vehicle is added successfully.Thanks for using our parki
                email_from = settings.EMAIL_HOST_USER
                recipient_list = [request.user.email, ]
                send_mail( subject, message, email_from, recipient_list )
            messages.success(request, "vehicle added successfully")
            return redirect('dashboard')
```

```
        else:
            messages.error(request, "There was some issue while trying to add vehcile details.Please try again later")
    else:
        form=addVehicleForm
    context={'user':users,'form':form,'update':False}
    return render(request,'addvehicle.html',context)
```

Models.py

```
from django.db import models
from django.contrib.auth.models import User
from django.core.validators import RegexValidator
```

```
class Vehicle(models.Model):
    veh_registration_no=models.CharField(max_length=10)
    veh_model=models.CharField(max_length=255)
    owner=models.CharField(max_length=200,default='dev')
    contact=models.CharField(max_length=10,validators=[RegexValidator(regex=r'^[6-9]\d{9}$',message="Phone number must b
    vehicle_category=models.CharField(max_length=50,choices=type,default="2")
    remark=models.TextField(blank=True)
    created_by=models.ForeignKey(User,on_delete=models.SET_NULL,null=True)
    parked=models.BooleanField(default=False)
    def __str__(self):
        return self.veh_registration_no
```

forms.py

```
from .models import *
from django import forms

class addVehicleForm(forms.ModelForm):
    class Meta:
        model=Vehicle
        fields=['veh_registration_no','veh_model','owner','contact','vehicle_category','remark','parked']
```

Detail Parking Page:

Functionality:

A detail page which displays dynamic information related to parking with an attractive User Interface which works smoothly and for admin have a feature of modifying the data and generate booking receipt.

Technologies Used:

Frontend: HTML, CSS, Bootstrap

Backend: Django

views.py

```
def parkingdetail(request,id):
    if not request.user.is_authenticated:
        messages.info(request, f'You are not authorised staff to view this page')
        return redirect('login')
    parkings=ParkingBookSlot.objects.get(id=id)
    context={'type':'parking','parking':parkings}
    return render(request,'detail.html',context)
```

```
def generate_pdf(request,pid):
    bookslot= ParkingBookSlot.objects.get(id=pid)
    response = FileResponse(generate_pdf_file(pid),
                            as_attachment=True,
                            filename=f'{bookslot.owner_name}.pdf')
    return response

def generate_pdf_file(pid):
    from io import BytesIO
    buffer = BytesIO()
    f = canvas.Canvas(buffer)
    ticket = ParkingBookSlot.objects.get(id=pid)
    f.drawString(250, 750, "Parking Book Reciept")

    y = 700
    f.drawString(450, 750, f" Parking Date: {ticket.parking_date}")
    f.drawString(400, y, f"Vehicle Type: {ticket.vehicle_type}")
    f.drawString(50, y, f"Vehicle Registration No: {ticket.veh_registration}")
    f.drawString(50, y - 30, f"Vehicle Intime: {ticket.intime}")
    f.drawString(370, y - 30, f"Vehicle Intime: {ticket.outtime}")
    f.drawString(50, y - 60, f"Parking Slot No: {ticket.slot_no}")
    f.drawString(50, y - 90, f"Parking Rate: {ticket.parking_rate}")
    f.drawString(50, y - 120, f"Parking Total Amount: {ticket.fee}")
    f.drawString(400, y - 120, f"Amount Due: {ticket.fee}")
```

```
f.drawString(50, y - 150, f"Rate is calculated: flat amount ={ticket.parking_rate} till 6hr addidtional hours= rs 8.33
y -= 180

f.showPage()
f.save()

buffer.seek(0)
return buffer
```

Detail Vehicle Page:

Functionality:

A detail page which displays dynamic information related to vehicle and give feature of editing the information with an attractive User Interface with smooth working.

Technologies Used:

Frontend: HTML, CSS, Bootstrap

Backend: Django

views.py

```
def vehicledetail(request,id):
    if not request.user.is_authenticated:
        messages.info(request, f'You are not authorised staff to view this page')
        return redirect('login')
    vehicles=Vehicle.objects.get(id=id)
    context={'type':'vehicle','vehicle':vehicles}
    return render(request,'detail.html',context)
```

Detail Parking Space Page:

Functionality:

A detail page which displays dynamic information related to parking spcae with an user friendly User Interface which works smoothly.

Technologies Used:

Frontend: HTML, CSS, Bootstrap

Backend: Django

views.py

```
def spacedetail(request,id):
    if not request.user.is_staff:
        messages.info(request, f'You are not authorised staff to view this page')
        return redirect('login')
    spaces=ParkingSpace.objects.get(no=id)
    if spaces.is_available==False:
        bookslot=ParkingBookSlot.objects.get(slot_no=id)
    else:
        bookslot=""
    context={'type':'space','space':spaces,'booking':bookslot}
    return render(request,'detail.html',context)
```

User Detail Page:

Functionality:

User detail page displays user information with an user friendly User Interface.

Technologies Used:

Frontend: HTML, CSS, Bootstrap

Backend: Django

views.py

```
def userdetail(request,uid):
    if request.user.is_authenticated:
        user=User.objects.get(id=uid)
        context={'user':user}
        return render(request,'userdetail.html',context)
    else:
        messages.info(request,"you are not authorised to view this page")
        return render(request,'userdetail.html')
```

View Vehicles Page:

Functionality:

Renders dynamic data in the form of table with options to delete and modify data with an smooth UI part created using html, CSS.

Technologies Used:

Frontend: HTML, CSS, Bootstrap

Backend: Django

views.py

```
def view_vehicles(request):
    if request.user.is_staff:
        vehicle_list=Vehicle.objects.all()
    elif request.user.is_authenticated and not request.user.is_staff:
        vehicle_list=Vehicle.objects.filter(created_by=request.user)
    else:
        messages.error(request,"Please sign in to view")
        return redirect('login')
    context={'vehicles':vehicle_list}
    return render(request,'viewvehicles.html',context)
```

View Parkings / Manage Parkings Page:

Functionality:

Renders dynamic data according to the user logged in in the form of table with options to delete and modify the data with a good UI part created using html, CSS.

Technologies Used:

Frontend: HTML, CSS, Bootstrap

Backend: Django

views.py

```
def view_parkings(request):
    if request.user.is_staff:
        parking_list=ParkingBookSlot.objects.all()
    elif request.user.is_authenticated and not request.user.is_staff:
        fullname=request.user.first_name+request.user.last_name
        parking_list=ParkingBookSlot.objects.filter(Q(booking_user=request.user) | Q(owner_name=fullname) | Q(owner_name
    else:
        messages.error(request,"Please sign in to view")
        return redirect('login')
    context={'parkings':parking_list}
    return render(request,'viewparkings.html',context)
```

View ParkingSpace Page:

Functionality:

Renders parking space data in the form of table with options to delete and modify the data with a good UI part created using html, CSS.

Technologies Used:

Frontend: HTML, CSS, Bootstrap

Backend: Django

Views.py

```
def view_space(request):
    space_list=ParkingSpace.objects.all()
    context={'spaces':space_list}
    return render(request,'viewspace.html',context)
```

User Registration Page:

Functionality:

Renders an html form to take input with a smooth user interface working.

Technologies Used:

Frontend: HTML, CSS, Bootstrap

Backend: Django, Django built in User model

Views.py

```
from django.contrib.auth.models import User
from django.db import IntegrityError
from django.core.mail import send_mail
```

```python
def register(request):
    if request.method=="POST":
        user=request.POST['user']
        firstname=request.POST['fname']
        lastname=request.POST['lname']
        email=request.POST['email']
        password=request.POST['password']
        confirm_password=request.POST['confpass']
        if confirm_password != password:
            messages.error(request,"password doesn't match")
            messages.info(request,"Failed to create user account.Please try again")
            return redirect('register')
        if User.objects.filter(username=user).exists():
            messages.error(request, "Username already taken. Please choose another username.")
            return redirect('register')
        else:
```

```python
    else:
        try:
            user=User.objects.create_user(username=user,first_name=firstname,last_name=lastname,email=email,password=pa
            user.save()
            subject = 'Welcome to AapniParking'
            message = f'Hi {firstname}!!, your account is created successfully.Your username is {user}'
            email_from = settings.EMAIL_HOST_USER
            recipient_list = [email, ]
            send_mail( subject, message, email_from, recipient_list )
            messages.success(request, "User registered successfully. You can now login.")
            return redirect('login')
        except IntegrityError as e:
            messages.error(request, "Username already exists.Try login with the same username")
return render(request,"register.html")
```

Login Page:

Functionality:

Renders an html form with an enhanced user interface.

Technologies Used:

Frontend: HTML, CSS, Bootstrap

Backend: Django, Django built in User model

Views.py

```python
def log_in(request):
    if request.method=='POST':
        username=request.POST['user']
        password=request.POST['password']
        user=authenticate(request,username=username,password=password)
        if user is not None:
            login(request,user)
            return redirect('dashboard')
        else:
            messages.info(request,"User doesn't exist. Please register yourself")
    return render(request,'login.html')
```

6. Challenges Faced

1.Providing the functionality of generating booking receipt by integrating it with the Reportlab module to have the feature of pdf generation.

2.Providing with some many good features for both admin and user and to make them easily accessible in user interface while ensuring smoot functioning of website.

7. Future Enhancements

1. Using data visualization to show different aspects of vehicle parking like total bookings in a day, month, types of vehicle generally parked more, revenue comparison over the month.
2. Integration with social media platforms for wider reach, mass publicity.

8. Conclusion

The AapniParking project successfully delivers a user-friendly, interactive with lots of feature website using Django framework. Its attractive design, integration of various technologies, and focus on providing smooth and good user experience makes it a valuable website to use for parking booking and management.

9. References

1. Django Documentation: [https://docs.djangoproject.com/en/5.0/]
2. BootstrapDocumentation:[https://getbootstrap.com/docs/5.0/getting-started/introduction/]
3. MDB Docs: [https://mdbootstrap.com/docs/standard/]
4. ReportLab: [https://docs.reportlab.com/reportlab/userguide/]