STM JOURNALS

International Journal of
Computer Science Languages

ISSN: 3048-944X
Volume 2, Issue 2, 2024
July–December
DOI (Journal): 10.37591/IJCSL

https://journals.stmjournals.com/ijcsl

Review IJCSL

A Design and Development of Online Ticket Reservation System

Niranjan R. Chougala[1,*], Malatesh S.H.[2], Manjula R. Chougala[3], Shruthi S.[4], Shridhar V.[5], Surendra Babu M.S.[5]

Abstract

The purpose of this case study is to develop and demonstrate various facilities and features available in Full Stack Development using Django by considering a simple example of an Online Ticket Reservation System (OTRS). The concepts of Django Models, Views, Templates, Admin Interfaces, and Forms are used to ensure this case study utilizes all features of the framework to make it the complete miniature product. In addition, a well-documented report and simple explanation related to the product are incorporated to complete the software development product cycle. For a better understanding of the product and its use, the simple business logic has been developed and documented so that the user can easily improve and scale up accordingly. Various interactions and interfaces among the code have been shown to make naive users or developers understand better as well as encourage contributing additional business logic easily. Results from executing the code have been shown with a brief explanation. This work targets and encourages naïve developers to understand the full stack development framework better by considering a simple case study.

Keywords: Full stack development, Django Models, views, templates, admin interfaces, forms

INTRODUCTION

Preamble on Full Stack Web Development

Websites are developed by any organization or institution to provide information about the products or services offered by them. Websites must support quick searches and easy delivery of information [1]. A website needs to be

- Well designed
- Responsive
- Dynamic
- Informative

Developers need skills to use technologies and frameworks that help them develop good websites [2]. Full stack development requires skills in the front-end, back-end, and business logic of an application. Several technologies can be used for a full stack. Django, which is based on Python [3], is a very popular, secure, and responsive stack designed by experts with many years of experience. Learning and using Django will be useful for web developers.

Purpose/Objective

- Investigate the applications of learning full stack web development.

***Author for Correspondence**
Niranjan R. Chougala
E-mail: niranjan.chougala@rrit.ac.in

[1]Professor, Department of Computer Science and Engineering, R.R. Institute of Technology, Bangalore, Karnataka, India
[2]Professor and Head, Department of Computer Science and Engineering, M S Engineering College, Bangalore, Karnataka, India
[3]Assistant Professor, Department of Computer Science and Engineering, M S Ramaiah Institute of Technology, Bangalore, Karnataka, India
[4]Assistant Professor, Department of Artificial Intelligence and Machine Learning, BMS Institute of Technology and Management College, Bangalore, Karnataka, India
[5]Assistant Professor, Department of Computer Science and Engineering, R.R. Institute of Technology, Bangalore, Karnataka, India

Received Date: October 07, 2024
Accepted Date: October 11, 2024
Published Date: November 05, 2024

Citation: Niranjan R. Chougala, Malatesh S.H., Manjula R. Chougala, Shruthi S., Shridhar V., Surendra Babu M.S. A Design and Development of Online Ticket Reservation System. International Journal of Computer Science Languages. 2024; 2(2): 54–63p.

- Utilize rapid application development to design responsive web pages.
- Demonstrate Models, Views, and Templates in Django and their interconnectivity for full stack web development.
- Illustrate how to automate admin interfaces in Django.
- Develop and implement Django applications featuring dynamic pages integrated with databases.

Problem Statement

- The OTRS is a bus reservation system that allows us to book tickets and check seat availability with respect to the given date.
- To keep things simple, we have only one bus per day.
- Admin users can add/delete/modify the bus along with the seating capacity for the day.
- The user can book or check the seats on a chosen day with the required seat.
- If the requested seats are less than or equal to the available seats, booking is accepted, and seats are updated accordingly.

Architecture

- Django uses the Model-View-Template(MVT) Pattern
- The overall project is called *case_study*.
- It has an application called *bus*.
- M = model is given by the *models.py* in the bus application.
- V = view is given by *views.py* in the bus application.
- T = template is given by two HTML files under bus*templates**bus*.

Models

- We have a model for *OTRS* that includes the date (of journey), total seats, available seats, and booked seats.
- We have a model called *Booking Request* which captures the date and time of booking, date (of journey), requested seats, and whether the booking was accepted or rejected.
- A default Sqlite3 [4] database holds the records.

Views

- There is a view to show the static information of OTRS records.
- There is also a more important view of booking. This view uses a model form based on the booking request model. This view processes the form and updates the corresponding OTRS records for available and booked seats. This view renders the template in form.
- The form has validations such as whether there should be a (single) bus on the date of the journey and if the requested number of seats is between 1 and the available seats.

Templates

- As discussed, there are two templates: one for the view showing static OTRS content and the other for the booking form.

URL Conf.

- Corresponding to the two views, there are two URLs in the bus application.
- These URLS get included in the case_study project.

Admin Interface

- Admin is enabled for the OTRS model.
- We created a superuser.
- A group called ‘bus booker’ has been created with privileges on OTRS.
- A user called ‘Ravi’ has been created and added to the bus booker.
- Ravi can now operate on OTRS records.

Booking Procedure

- Enter or choose a date.
- Enter the number of seats requested.
- Click the submit button.
- If validation succeeds, booking is performed, and the numbers are updated in OTRS, as can be seen in the refreshed static page or the admin interface.
- In case of validation errors, an error message is shown.

Full Stack Developer

A full stack developer is a multifaceted programmer proficient in both front-end and back-end technologies, capable of developing and maintaining complete web applications, as shown in Figure 1.

Django Architecture

Django follows the MVT architecture, where 'M' represents Model, 'V' stands for View, and 'T' denotes Template as shown in Figure 2.

- *M:* Models represent data from applications. Each model corresponds to a table in a relational database, and Django supports various databases including SQLite, MySQL, SQL Server, Oracle, and PostgreSQL. Every model is given as a Python class, with the data members as the columns in the table. Models can represent various types of relationships, including one-to-one, one-to-many, and many-to-many relationships.
- *V:* View is Python that takes on a web request, typically, HttpRequest, and returns a response, typically HttpResponse with HTML. The response can be constructed either as an HTML string or by rendering a template [5].
- *T:* Template is usually an HTML file with placeholders or variables provided by view-rendering them. Templates also supported three constructs other than the variables: tags, filters, and comments. Tags represent logic, such as conditions or for and, and loops. The filters act on the data. Comments are useful for documenting the template but are ignored otherwise.

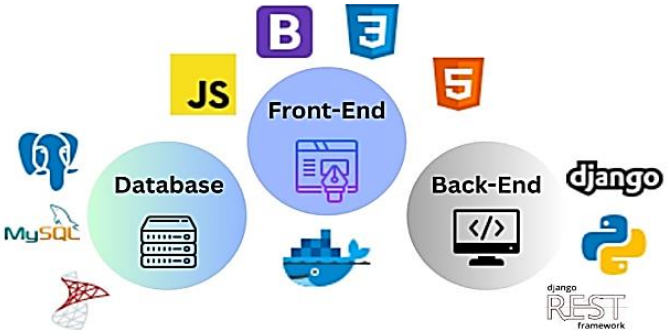

Figure 1. Full stack developer.

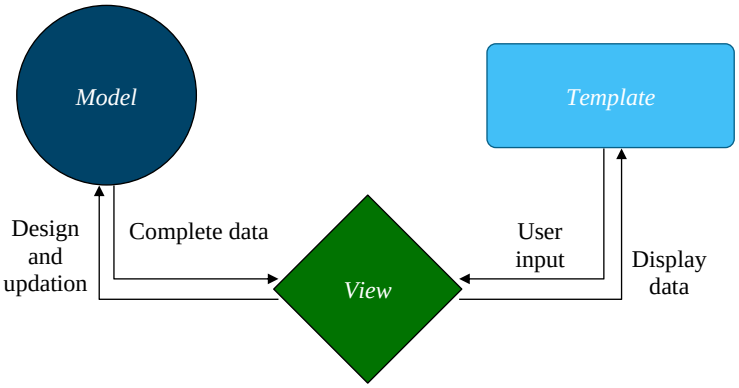

Figure 2. Django MVT architecture.

MVT Interaction

MVT Interaction involves a dynamic connection between models, views, and templates in web applications, facilitating smooth data flow and user interface presentation, as shown in Figure 3.

THE CODE

models.py

In Django, the models are tabulated in a relational database. Django supports several databases such as MySQL, SQL*Server, Oracle, and ProgreSQL [6]. However, the default database is SQLite unless configured otherwise. The default database was used. Each model is given here as a Python [3] class extending the django.db.models.Model. Each data member in the class corresponds to a field in the table. The relationships between models can be one-to-one, one-to-many, or many-to-many. In OTRS, we have the following models:

- *OTRS:* Represents a bus on a particular date with the date of journey, total seats, available seats, and booked seats.
- *Booking request:* This represents a request with the date of the journey and the seats requested. It also records the date and time of the booking and serves as the foundation for the booking form model.

models.py

```
from django.db import models

# Create your models here.

class OTRS(models.Model):
    date = models.DateField()
    total_seats = models.IntegerField()
    available_seats = models.IntegerField()
    booked_seats = models.IntegerField()

class BookingRequest(models.Model):
    booking_date = models.DateTimeField()
    date = models.DateField()
    requested_seats = models.IntegerField()
    granted = models.BooleanField()
```

views.py

A Django View is a function that processes a web request and provides a web response. Typically, the request is an HttpRequest. The most common response was HttpResponse, which contained HTML. The response can be constructed either using django.http.HttpResponse or by rendering an HTML template. In OTRS, we obtain the following views:

- *BusListView:* This is a view of OTRS records that uses the concept of a generic ListView based on the OTRS model.

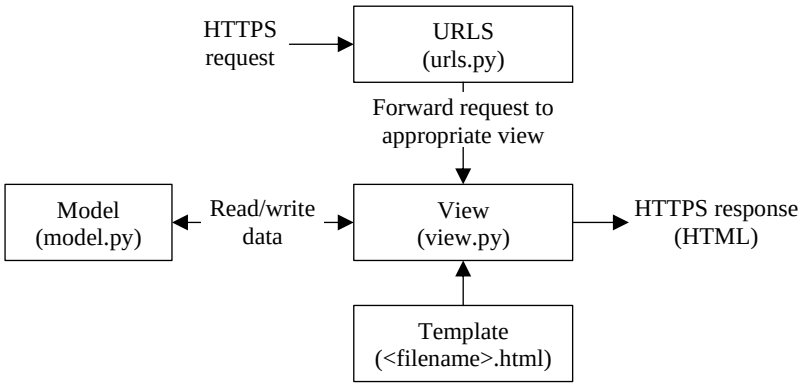

Figure 3. MVT interaction.

- *Book:* This is a view that renders a template using a BookingRequestForm for booking seats. The BookingRequestForm is based on the BookingRequest model [7].

views.py

```
from django.shortcuts import render
from django.views.generic import ListView
from.models import OTRS, BookingRequest
from.forms import BookingRequestForm
import datetime

# Create your views here.

class BusListView(ListView):
    model = OTRS

def book(request):
    booked = False
    if request.POST:
        form = BookingRequestForm(request.POST)
        if form.is_valid():
            booked = False
            date = form.cleaned_data[‘date’]
            requested_seats = form.cleaned_data[‘requested_seats’]
            otrs = OTRS.objects.get(date=date)
            if otrs.available_seats >= requested_seats:
                otrs.available_seats -= requested_seats
                otrs.booked_seats += requested_seats
                booked = True
                otrs.save()
            booking_request = BookingRequest.objects.create(booking_date=datetime.datetime.now(), date=date,
requested_seats=requested_seats, granted=booked)
            booking_request.save()
    else:
        form = BookingRequestForm()
    ctx = { ‘form’: form, ‘booked’: booked}
    return render(request, ‘bus/booking_request.html’, ctx)
```

admin.py

We create a superuser for using the admin interface by running the command py manage.py createsuperuser in the main case study project directory. We registered the models OTRS and BookingRequest in admin.py. The way the OTRS records are to be displayed is specified in the class OTRSAdmin. True power lies not in control but in the responsibility to use it wisely. We created another user called “Ravi” with permissions on OTRS records so that he can create the initial OTRS records.

admin.py

```
from django.contrib import admin
from.models import OTRS

# Register your models here.

class OTRSAdmin(admin.ModelAdmin):
    list_display = [‘date’, ‘total_seats’, ‘available_seats’, ‘booked_seats’]
    search_fields = [‘date’]
    list_filter = [‘date’, ‘available_seats’]
    ordering = [‘date’]

admin.site.register(OTRS, OTRSAdmin)
```


forms.py

There are three ways to create form in Django. One is the use of HTML [8]. The other is to create a class that extends the django.forms.Form. The third approach, which we used here, is to base the form on a model. In the OTRS application, we created a BookingRequestForm class extending django.forms.ModelForm based on the BookingRequest model.

Validations

Forms can have associated validations [9]. In this case, all validations regarding booking are included in the clean-up method. The validation error increases if there is no bus on the date of booking, the number of seats requested is not positive, or the number of seats requested exceeds the number of seats available.

forms.py

```
from django import forms
from django.forms import ModelForm
from django.forms import ValidationError
from.models import OTRS, BookingRequest

class DateInput(forms.DateInput):
   input_type = 'date'

class BookingRequestForm(ModelForm):
   class Meta:
      model = BookingRequest
      fields = ['date', 'requested_seats']
      widgets = {
         'date': DateInput(),
      }

   def clean(self):
      date = self.cleaned_data['date']
      requested_seats = self.cleaned_data['requested_seats']
      try:
         otrs = OTRS.objects.get(date=date)
      except Exception as ex:
         raise ValidationError('No bus on that date')
      if requested_seats <= 0:
         raise ValidationError('Seats should be > 0')
      if otrs.available_seats < requested_seats:
         raise ValidationError('Only %s seats available' %otrs.available_seats)
      return self.cleaned_data
```

urls.py

We have 2 URLs corresponding to the two views:

1. *Bookings:* Gives the complete list view of all OTRS records
2. *Book:* URL for the view of the booking form.

These entries are made as paths in the urlpattern list of urls.py for the bus application. These URLs are included in the urls.py of the main case study project [10].

urls.py

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

urlpatterns = [
   path('bookings', views.BusListView.as_view()),
   path('book', views.book),
]
```


Templates: Static View

Template: OTRS_list.html

```
<h1>Bookings</h1>
<ol>
{% for o in otrs_list %}
<li>{{ o.date }} {{ o.available_seats }} {{ o.booked_seats }} {{ o.total_seats }}</li>
{% endfor %}
</ol>
```

Template for Form

Template: booking_request.html

```
{% if booked %}
<h1 style='background-color:green;'>Booked Successfully!</h1>
{% endif %}
<h1>Booking Request Form</h1>
<form method='POST' action='/bus/book'>
   {% csrf_token %}
   <table>
      {{ form.as_table }}
   </table>
   <input type='submit'/>
</form>
```

RUNNING AND RESULTS

How To Run

- On the command line, the outer case_study directory gives the command py manage.py runserver. The server starts. Point browser at http://127.0.0.1:8000/bus/bookings.
- You will see static information like this in Figure 4.
- Go to the admin page http://127.0.0.1:8000/admin. Login as ravi, and interacts with the OTRS system to create entries (Figure 5).
- For booking buses, we try url http://127.0.0.1:8000/bus/book. Try valid and invalid values for form and submit (Figure 6).
- *No bus error:* A No bus error occurs when a program tries to access misaligned or nonexistent memory, resulting in a crash or execution failure (Figure 7).
- *Insufficient seat error:* An insufficient seat error arises when there are not enough seats available to accommodate a booking request (Figure 8).
- *Successful booking:* Your booking has been successfully completed! (Figure 9).

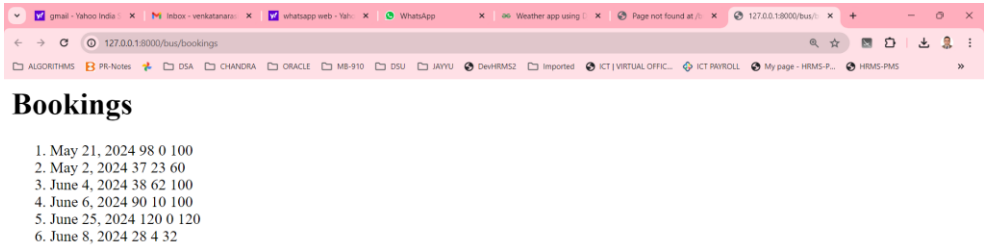

Figure 4. Bookings list page.

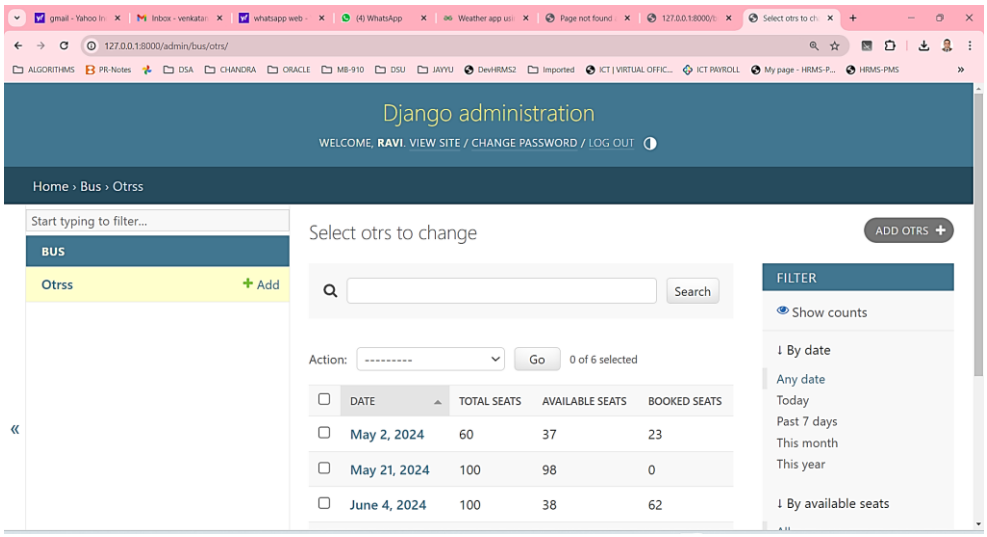

Figure 5. OTRS Admin Interface.

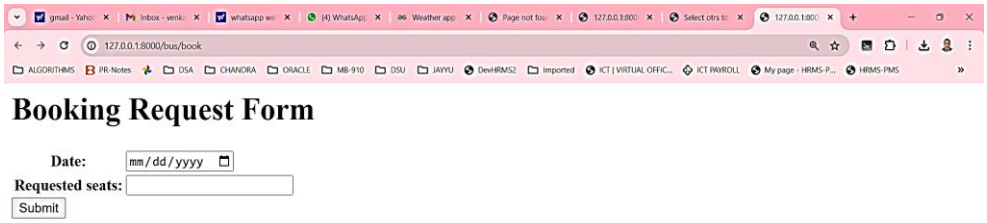

Figure 6. Booking request form.

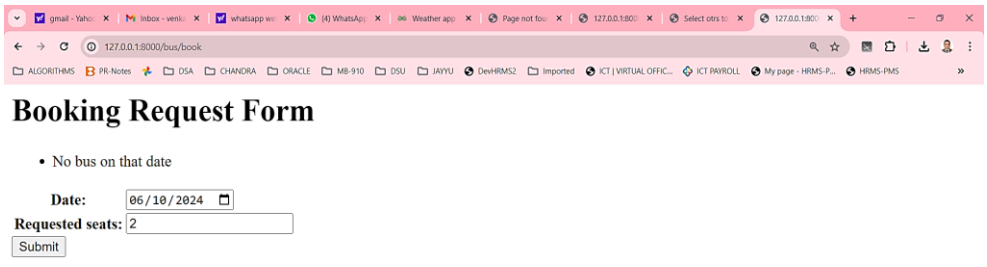

Figure 7. No bus on that date error.

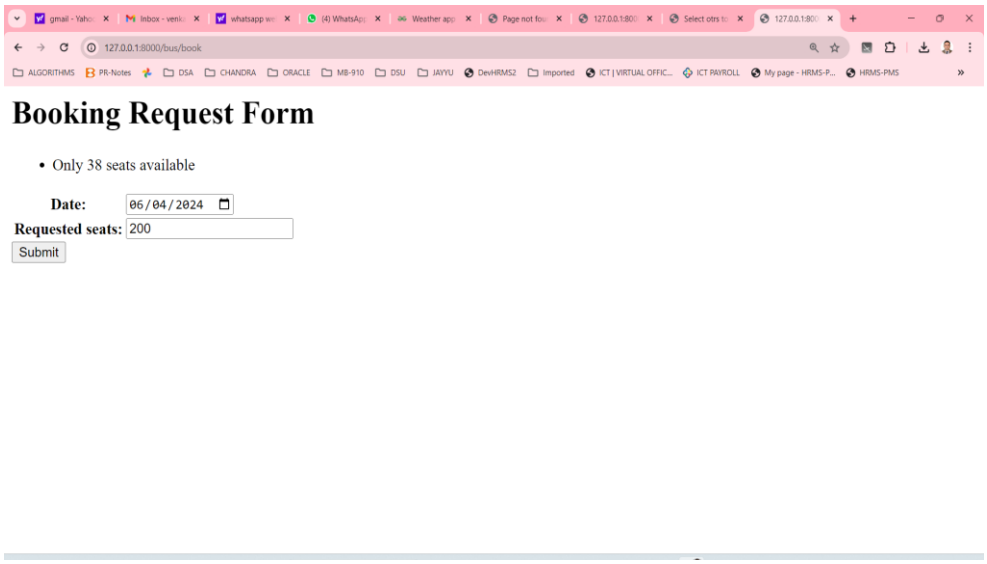

Figure 8. Insufficient seat error.

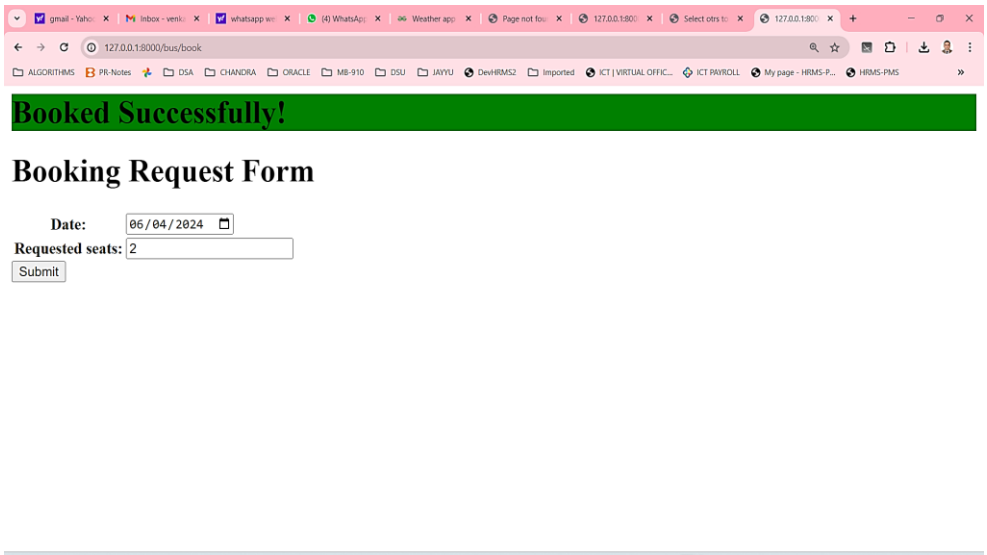

Figure 9. Successful booking.

CONCLUSION

We have demonstrated a Django project using Full stack development with Django Models, Views, Templates, Admin Interfaces, and Forms features to operate and manage a simple OTRS using very basic business logic along with localhost execution output. The bus application has suitable models for the OTRS and booking requests. It has a form based on the booking request model. The view for processing posted requests uses the form, and if the form is valid, accepts the request and makes the necessary changes to the OTRS records. The admin interface is used for OTRS with permissions for staff users based on a group.

REFERENCES

1. Northwood C. The Full Stack Developer: Your Essential Guide to the Everyday Skills Expected of a Modern Full Stack Web Developer. New York, NY: Apress; 2018. DOI: 10.1007/978-1-4842-4152-3.
2. Sommerville I. Software Engineering. 10th ed. Harlow, England: Pearson; 2016.
3. Manjunath R, Shivakumar Swamy N, Niranjan R Chougala, Shruthi S. Machine Learning Using Python. Bengaluru, India; 2024.

4. Vaswani V. MySQL Database Usage & Administration. New York, NY: McGraw-Hill; 2010.
5. Duckett J. JavaScript and jQuery: Interactive Front-End Web Development. Indianapolis, IN: Wiley Publishing; 2014.
6. Bendoraitis A, Kronika J. Django 3 Web Development Cookbook: Actionable Solutions to Common Problems in Python Web Development. Birmingham, UK: Packt Publishing Ltd; 2020.
7. Holovaty A, Kaplan-Moss J. The Definitive Guide to Django: Web Development Done Right. New York, NY: Apress; 2009.
8. Goldstein A, Lazaris L, Weyl E. HTML5 & CSS3 for the Real World: Powerful HTML5 and CSS3 Techniques You Can Use Today! Melbourne, Australia: SitePoint Pty Ltd.; 2015.
9. W3Schools. (2024). Django Tutorial. [Online]. Available from: https://www.w3schools.com/django/index.php
10. GeeksforGeeks. (2020). Django Tutorial. Learn Django Framework. [Online]. Available from: https://www.geeksforgeeks.org/django-tutorial/

