**Company Profile**

**Company and Mentor Details**

**Project Profile**

**Title: Blood Test System**

**Abstract: Our project, a comprehensive blood test management system developed using Django, offers a seamless experience for patients seeking blood tests. The system facilitates the booking and management of appointments, allowing patients to schedule appointments conveniently. Additionally, patients can provide feedback on their experience, contributing to continuous improvement. Moreover, the system provides a secure platform for patients to access and view their test reports online, ensuring efficient communication of critical health information. With user-friendly interfaces and robust functionalities, our project aims to enhance the efficiency and effectiveness of blood test management while prioritizing patient satisfaction and convenience.**

**Key-Words: Patient, Appointments, Laboratory, Feedbacks**

**Modules: Laboratory Admin, Patient, Home, Lab Worker**

**Technology:**

**Front-End - HTML, CSS, JavaScript, Bootstrap, ReactJS**

**Back End - Django web framework (Python)**

**Database – SQLite**

**Day 1**

**“Unraveling the Strengths of Python and Django: A Detailed Exploration”**

**Prompt: Craft a summary outlining Python and the Django framework, highlighting Python's capabilities while also presenting an accompanying diagram for clarity and brevity.**

Python, renowned for its simplicity and versatility, serves as a powerful programming language for a myriad of applications. With its clear syntax and extensive standard library, Python facilitates rapid development and readability, making it a favorite among developers worldwide. Its dynamic typing and strong support for object-oriented, functional, and procedural programming paradigms enable developers to tackle diverse tasks efficiently.

When coupled with the Django framework, Python becomes even more formidable for web development. Django, a high-level web framework, streamlines the creation of robust, scalable web applications. Built on the principles of DRY (Don't Repeat Yourself) and convention over configuration, Django promotes clean, maintainable code and accelerates development through its built-in features and utilities.

Django's architecture follows the Model-View-Controller (MVC) pattern, with slight variations, such as the Model-View-Template (MVT) pattern. Models define the data structure, views handle user requests and generate responses, and templates render HTML pages. Additionally, Django includes an Object-Relational Mapping (ORM) layer, facilitating database interactions without requiring direct SQL queries.

The framework provides built-in security features, such as protection against common web vulnerabilities like SQL injection and Cross-Site Scripting (XSS). Furthermore, Django offers authentication, session management, and authorization mechanisms out of the box, ensuring robust security for web applications.

In summary, Python's versatility and simplicity, coupled with Django's powerful features and conventions, make them a formidable combination for web development, enabling developers to create sophisticated and scalable web applications efficiently.

**[Python Functionality]**

- Clear Syntax

- Extensive Standard Library

- Dynamic Typing

- Support for Multiple Programming Paradigms

- Object-Oriented

- Functional

- Procedural

**[Django Framework]**

- High-Level Web Framework

- Follows MVC/MVT Architecture

- Built-in Features:

- ORM Layer

- Security Mechanisms

- Authentication & Authorization

- Promotes Clean, Maintainable Code

- Accelerates Development

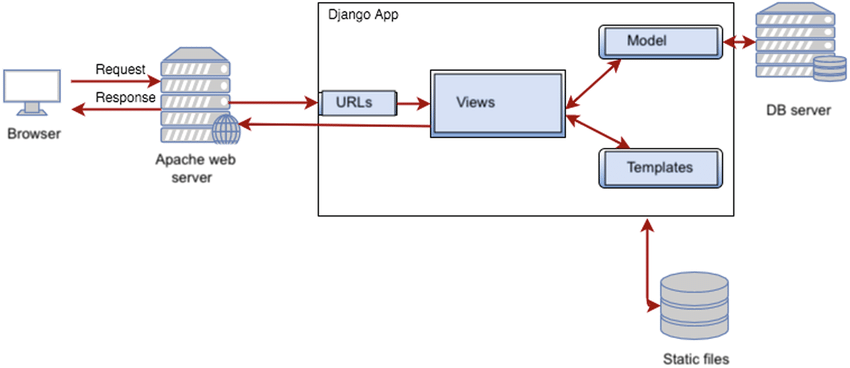
**[Web Applications]**

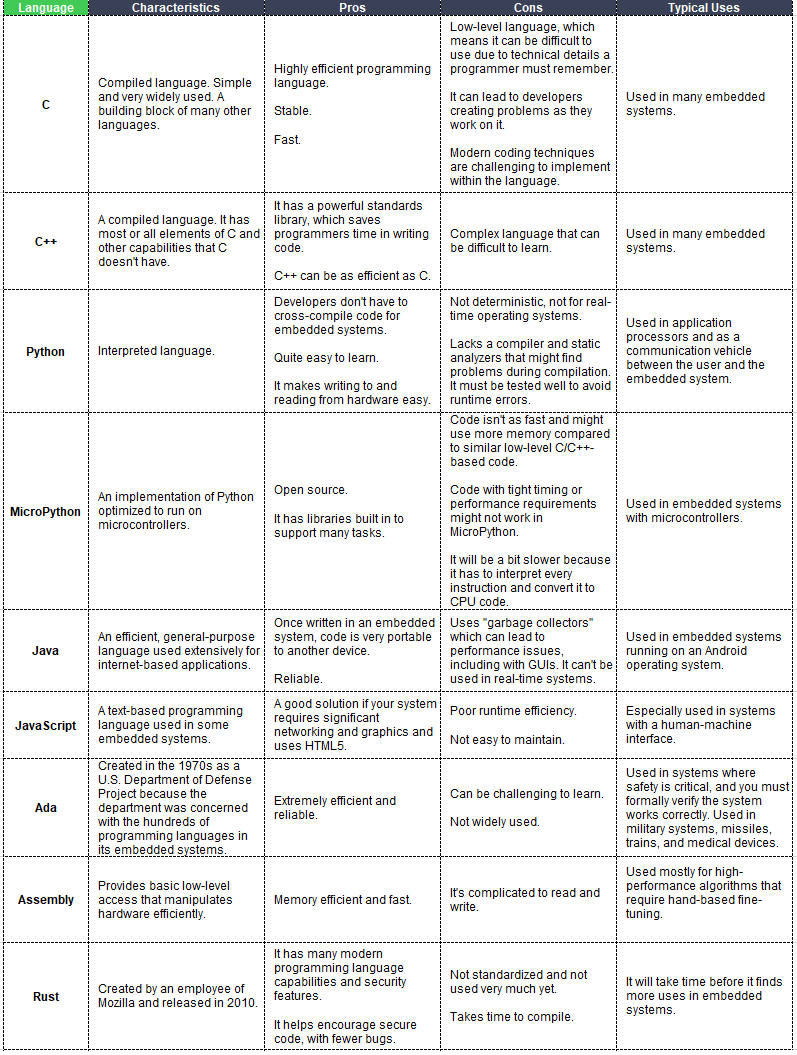
- Robust

- Scalable

- Secure

- Efficiently Developed



**Prompt: Compare python with other programming languages**

**Day 2**

**“Building a web application with Django”**

**Prompt: Django project development, featuring detailed code and visual aids.**

1. \*\*Setup Environment:\*\*

- Install Django: `pip install django`.

- Create a new Django project: `django-admin startproject myproject`.

- Create a new app: `python manage.py startapp myapp`.

2. \*\*Define Models:\*\*

```python

# myapp/models.py

from django.db import models

class MyModel(models.Model):

name = models.CharField(max\_length=100)

description = models.TextField()

created\_at = models.DateTimeField(auto\_now\_add=True)

def \_\_str\_\_(self):

return self.name

```

3. \*\*Migrate Database:\*\*

```

python manage.py makemigrations

python manage.py migrate

```

4. \*\*Create Views:\*\*

```python

# myapp/views.py

from django.shortcuts import render

from .models import MyModel

def my\_view(request):

objects = MyModel.objects.all()

return render(request, 'myapp/my\_template.html', {'objects': objects})

```

5. \*\*Create Templates:\*\*

- Create `my\_template.html` in `myapp/templates/myapp/` directory.

6. \*\*Define URLs:\*\*

```python

# myapp/urls.py

from django.urls import path

from . import views

urlpatterns = [

path('', views.my\_view, name='my\_view'),

]

```

7. \*\*Configure Project URLs:\*\*

```python

# myproject/urls.py

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

path('admin/', admin.site.urls),

path('', include('myapp.urls')),

]

```

8. \*\*Run Development Server:\*\*

```

python manage.py runserver

```

9. \*\*Access your project:\*\* Visit **`http://127.0.0.1:8000/`** in your browser.

This is a basic setup to get you started. From here, you can expand your project by adding more models, views, templates, and functionalities as needed.

**Day 3**

**“Login and Register”**

**Prompt: Creating Login and Register pages for website**

Step 1: Go to ‘views.py’ location and write code for login and signup.

|  |
| --- |
| def Lab\_signin(request):  if request.method=="POST":  print(request.POST['Lid'])  try:  m = LaboratoryRegister.objects.get(Lid=request.POST['Lid'])  if m.Laboratorypwd == request.POST['Laboratorypwd']:  request.session['Lid'] = m.Lid  request.session['name'] = m.Laboratoryfname  request.session['fname'] = m.Laboratorymname  request.session['mname'] = m.Laboratorylname  return redirect('laboratory:index')  else:  return render(request,'Lab\_signin.html',{'m':'Password incorrect'})  except:  return render(request,'Lab\_signin.html',{'m':'Password incorrect'})  return render(request,'Lab\_signin.html')  def signin(request):  if request.POST:  email = request.POST['uid']  pass1 = request.POST['userpwd']  try:  valid = UserRegister.objects.get(uid=email,userpwd=pass1)  if valid:  request.session['user'] = email  request.session['userId'] = valid.pk  return redirect('user:index')  else:  return render(request,'signin1.html',{'m':'Password incorrect'})  except:  return render(request,'signin1.html',{'m':'Password incorrect'})  # return redirect('/signin/')  return render(request,'signin1.html') |

|  |
| --- |
| # Registration  def signup(request):  obj=UserRegisterForm(request.POST)  if obj.is\_valid():  data=UserRegister.objects.all().filter(uid=request.POST['uid'])  if len(data)<=0:  obj.save()  return redirect('user:signin')  else:  return render(request,'signup.html',{'messagekey':"User Already Exists"})  return render(request,'signup.html')  def Lab\_signup(request):  obj=LaboratoryRegisterform(request.POST,request.FILES)  if obj.is\_valid():  data=LaboratoryRegister.objects.all().filter(Lid=request.POST['Lid'])  print(len(data))  if len(data)<=0:  obj.save()  return redirect('laboratory:lab\_signin')  else:  return render(request,'Lab\_signup.html',{'messagekey':"User Already Exists"})  return render(request,'Lab\_signup.html') |

Step 2: Create HTML templates to render the views.

|  |
| --- |
| <!— login.html -->  {% load static%}  <!DOCTYPE html>  <html lang="en" dir="ltr">  <head>  <title>َLogin</title>  <meta charset="utf-8">  <meta name="viewport" content="width=device-width, initial-scale=1">  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.1/dist/css/bootstrap.min.css">  <link rel="stylesheet" href="{% static 'signin1.css' %}">  <script src="https://cdn.jsdelivr.net/npm/jquery@3.6.0/dist/jquery.slim.min.js"></script>  <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js"></script>  <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.1/dist/js/bootstrap.bundle.min.js"></script>  <style>  .card{  background-color: transparent;  color: #017f92;  }  a{  color: #017f92;  }  a:hover{  color: #017f92;  }  </style>  </head>    <body>  <div>  <img src="{% static 'img/bg20.jpg' %}" style="height: 100vh; width:100%; position:absolute; z-index: -1; opacity: 1;">  </div>  <div style="position:relative; top:100px;">  <center>  <div class="col-md-4">  <div class="card bg-mute">  <form method="POST" autocomplete="off">  {% csrf\_token %}  <h1 class="h1 text-center mt-5 mb-5">Laboratory Login</h1>  <div class="m-1">  <input type="email" class="form-control" id="email" placeholder="Enter email" name="Lid" value="{{ Lid }}">  </div>  <div class="m-1">  <input type="password" class="form-control" placeholder="Enter password" name="Laboratorypwd" value="{{ Laboratorypwd }}">  </div>  <div class="mt-1 mb-1">  <a href="/med/show/"><button type="submit" class="btn btn-info">Sign Up</button></a>  </div>  <hr/>  <div class="foot-lnk text-center">  <a href="{% url 'laboratory:lab\_signup' %}">Create an account</a>  </div>  </form>  </div>  </div>  </center>  </div>  {% if m %}  <script>alert("{{m}}")</script>  {% endif %}  </body>  </html>  {% load static %}  <!DOCTYPE html>  <html lang="en" dir="ltr">  <head>  <title>َLogin</title>  <meta charset="utf-8">  <meta name="viewport" content="width=device-width, initial-scale=1">  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.1/dist/css/bootstrap.min.css">  <link rel="stylesheet" href="{% static 'signin1.css' %}">  <script src="https://cdn.jsdelivr.net/npm/jquery@3.6.0/dist/jquery.slim.min.js"></script>  <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js"></script>  <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.1/dist/js/bootstrap.bundle.min.js"></script>  <style>  .card{  background-color: transparent;  color: #17a2b8;  }  a{  color: #17a2b8;  }  a:hover{  color: #17a2b8;  }  </style>  </head>    <body>  <div>  <img src="{% static 'img/login.jpg' %}" style="height: 100vh; width:100%; position:absolute; opacity: 0.5;">  </div>  <div style="position:relative; top:100px;">  <center>  <div class="col-md-4">  <div class="card">  <form method="POST" >  {% csrf\_token %}  <h1 class="h1 text-center mt-5 mb-5">User Login</h1>  <div class="m-1">  <input type="text" class="form-control" id="email" placeholder="Enter email" name="uid" value="{{ uid }}">  </div>  <div class="m-1">  <input type="password" class="form-control" placeholder="Enter password" name="userpwd" value="{{ userpwd }}">  </div>  <div class="mt-4 mb-3">  <div class="">  <button type="submit" class="btn btn-info">Sign In</button>  </div>  </div>  <hr/>    <div class="foot-lnk text-center">  <a class="#Create" href="{% url 'user:signup' %}" >Create an account</a>  </div>      </form>  </div>  </div>  </center>  </div>  {% if m %}  <script>alert("{{m}}")</script>  {% endif %}  </body>  </html> |

|  |
| --- |
| <!—register.html -->  {% load static %}  <!DOCTYPE html>  <html lang="en" dir="ltr">  <head>  <title>Registration</title>  <meta charset="utf-8">  <meta name="viewport" content="width=device-width, initial-scale=1">  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.1/dist/css/bootstrap.min.css">  <link rel="stylesheet" href="{% static 'signin1.css' %}">  <script src="https://cdn.jsdelivr.net/npm/jquery@3.6.0/dist/jquery.slim.min.js"></script>  <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js"></script>  <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.1/dist/js/bootstrap.bundle.min.js"></script>  <style>  .card{  background-color: transparent;  color: #17a2b8;  }  a{  color: #17a2b8;  }  a:hover{  color: #17a2b8;  }  </style>  </head>  <body>  <div>  <img src="{% static 'img/login.jpg' %}" style="height: 100vh; width:100%; position:absolute; opacity: 0.5;">  </div>  <div style="position:relative;">  <center>  <div class="col-md-4">  <div class="card">  <form method="POST" >  {% csrf\_token %}    <h1 class="h1 text-center mt-2 mb-2">SIGN UP</h1>  <div class="m-1">  <input type="email" class="form-control" id="email" placeholder="Enter email"name="uid" value="{{ uid }}" >  </div>  <div class="m-1">  <input type="password" class="form-control" placeholder="Enter password" name="userpwd" value="{{ userpwd }}">  </div>  <div class="m-1">  <input type="text" class="form-control" id="fname" placeholder="First Name" name="userfname" value="{{ userfname }}">  </div>  <div class="m-1">  <input type="text" class="form-control" id="mname" placeholder="Middle Name" name="usermname" value="{{ usermname }}">  </div>  <div class="m-1">  <input type="text" class="form-control" id="lname" placeholder="Last Name" name="userlname" value="{{ userlname }}">  </div>  <div class="m-1">  <input type="number" class="form-control" id="user\_age" placeholder="Age" name="user\_age" value="{{ user\_age }}">  </div>  <label class="m-1" for="gender"> Select your gender</label>  <select name="usergender" class="m-1">  <option value="none" selected>Gender</option>  <option value="Male">Male</option>  <option value="Female">Female</option>    </select>  <div class="m-1">  <textarea row="3" class="form-control" id="Address" placeholder="Address" name="useraddress" value="{{ useraddress }}"></textarea>  </div>  <div class="m-1">  <input type="text" class="form-control" id="city" placeholder="City" name="usercity" value="{{ usercity }}">  </div>  <div class="m-1">  <input type="text" class="form-control" id="area" placeholder="Area" name="userarea" value="{{ userarea }}">  </div>  <div class="m-1">  <input type="text" class="form-control" id="Pincode" placeholder="Pincode" name="userpincode" value="{{ userpincode }}">  </div>  <div class="m-1">  <input type="number" class="form-control" id="content\_no" placeholder="Contact No" name="usercontactno" value="{{ usercontactno }}">  </div>  <div class="mt-1 mb-1">  <button type="submit" class="btn btn-info">Sign Up</button>  </div>  <hr/>  <div class="foot-lnk text-center">  <a href="{% url 'user:signin' %}">Already Member?</a>  </div>    </form>  </div>  </div>  </center>  {% if messagekey %}  <script>alert("{{ messagekey }}")</script>  {% endif %}  </div>  </body>  </html>    {% load static%}  <!DOCTYPE html>  <html lang="en" dir="ltr">  <head>  <title>َLogin</title>  <meta charset="utf-8">  <meta name="viewport" content="width=device-width, initial-scale=1">  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.1/dist/css/bootstrap.min.css">  <link rel="stylesheet" href="{% static 'signin1.css' %}">  <script src="https://cdn.jsdelivr.net/npm/jquery@3.6.0/dist/jquery.slim.min.js"></script>  <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js"></script>  <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.1/dist/js/bootstrap.bundle.min.js"></script>  <style>  .card{  background-color: transparent;  color: #017f92;  }  a{  color: #017f92;  }  a:hover{  color: #017f92;  }  </style>  </head>    <body>  <div>  <img src="{% static 'img/lab.jpg' %}" style="height: 100vh; width:100%; position:absolute; z-index: -1; opacity: 0.5;">  </div>  <div style="position:relative;">  <center>  <div class="col-md-4">  <div class="card">  <form method="POST" autocomplete="off" enctype="multipart/form-data">  {% csrf\_token %}  <h1 class="h1 text-center mt-2 mb-2">Laboratory Signup</h1>  <div class="">  <input type="email" class="form-control" id="email" placeholder="Enter email" name="Lid" value="{{ Lid }}">  </div>  <div class="">  <input type="password" class="form-control" placeholder="Enter password" name="Laboratorypwd" value="{{ Laboratorypwd }}">  </div>  <div class="">  <input type="text" class="form-control" id="fname" placeholder="Laboratory Name" name="Laboratoryfname" value="{{ Laboratoryfname }}">  </div>  <div class="">  <input type="text" class="form-control" id="mname" placeholder="First Name" name="Laboratorymname" value="{{ Laboratorymname }}">  </div>  <div class="">  <input type="text" class="form-control" id="lname" placeholder="Last Name" name="Laboratorylname" value="{{ Laboratorylname }}">  </div>  <div class="">  <textarea row="2" class="form-control" id="Address" placeholder="Address" name="Laboratoryaddress" value="{{ Laboratoryaddress }}"></textarea>  </div>  <div class="">  <input type="text" class="form-control" id="city" placeholder="City" name="Laboratorycity" value="{{ Laboratorycity }}">  </div>  <div class="">  <input type="text" class="form-control" id="area" placeholder="Area" name="Laboratoryarea" value="{{ Laboratoryarea }}">  </div>  <div class="">  <input type="text" class="form-control" id="Pincode" placeholder="Pincode" name="Laboratorypincode" value="{{ Laboratorypincode }}">  </div>  <div class="">  <input type="number" class="form-control" id="content\_no" placeholder="Contact No" name="Laboratorycontactno" value="{{ Laboratorycontactno }}">  </div>  <div class="">  <input id="id\_chemistphoto" type="file" name="Laboratoryphoto" class="form-control" value="{{ Laboratoryphoto }}">  </div>  <div class="mt-2">  <a href="/med/show/"><button type="submit" class="btn btn-info">Sign Up</button></a>    </div>  {% if messagekey %}  <script>alert("{{ messagekey }}")</script>  {% endif %}  <hr/>  <div class="foot-lnk text-center">  <a href="{% url 'laboratory:lab\_signin' %}">Already Member?</a>  </div>    </form>      </div>  </div>  </center>    </div>    </body>  </html> |

Step 3: Go to ‘forms.py’ location and create form for registrations.

|  |
| --- |
| from django import forms  from laboratoryapp.models import LaboratoryRegister,Appointment,Test\_category  class LaboratoryRegisterform(forms.ModelForm):  class Meta:  model=LaboratoryRegister  fields='\_\_all\_\_'  class UserRegisterForm(forms.ModelForm):  class Meta:  model=UserRegister  fields='\_\_all\_\_' |

Step 4: Go to ‘models.py’ and create register model.

|  |
| --- |
| from django.db import models  # Create your models here.  # User Details  class UserRegister(models.Model):  uid=models.EmailField(max\_length=50,verbose\_name='Email')  userpwd=models.CharField(max\_length=20,verbose\_name='Password')  userfname=models.CharField(max\_length=20,default='',verbose\_name='First\_Name')  usermname=models.CharField(max\_length=20,default='',verbose\_name='Middle\_Name')  userlname=models.CharField(max\_length=20,default='',verbose\_name='Last\_Name')  n='slect gender'  g = [  (None, 'select gender'),  ('Male', 'Male'),  ('Female', 'Female'),  ]  usergender=models.CharField(max\_length=12,choices=g,default=None)  user\_age=models.IntegerField(max\_length=3,default=None)  useraddress=models.CharField(max\_length=20,default='',verbose\_name='Address')  usercity=models.CharField(max\_length=30,default='',verbose\_name='City')  userarea=models.CharField(max\_length=20,default='',verbose\_name='Area')  userpincode=models.IntegerField(default='',verbose\_name='Pincode')  usercontactno=models.IntegerField(default='',verbose\_name='Contact\_No')  def \_\_str\_\_(self):  return self.uid  class LaboratoryRegister(models.Model):  Lid=models.EmailField(max\_length=50,verbose\_name='Email')  Laboratorypwd=models.CharField(max\_length=20,verbose\_name='Password')  Laboratoryfname=models.CharField(max\_length=20,default='',verbose\_name='Laboratory\_Name')  Laboratorymname=models.CharField(max\_length=20,default='',verbose\_name='First\_Name')  Laboratorylname=models.CharField(max\_length=20,default='',verbose\_name='Last\_Name')  Laboratoryaddress=models.CharField(max\_length=20,default='',verbose\_name='Address')  Laboratorycity=models.CharField(max\_length=20,default='',verbose\_name='City')  Laboratoryarea=models.CharField(max\_length=20,default='',verbose\_name='Area')  Laboratorypincode=models.IntegerField(default='',verbose\_name='Pincode')  Laboratorycontactno=models.IntegerField(default='',verbose\_name='Contact\_No')  Laboratoryphoto = models.FileField(upload\_to = 'upload',verbose\_name='Laboratory certificate')    def \_\_str\_\_(self):  return self.Lid |

**Day 4**

**“Dashboard”**

**Prompt: Creating Dashboard for BTS**

Step 1: Go to ‘views.py’ location and write code for dashboard.

|  |
| --- |
| def lab\_home(request):  if 'Lid' in request.session:  a=LaboratoryRegister.objects.get(Lid=request.session['Lid'])  b=Test\_category.objects.all()  return render(request,'index.html',{'a':a,'b':b,'c':len(b)})  else:  return redirect('laboratory:lab\_signin')  def index(request):  if 'user' in request.session.keys():  b=UserRegister.objects.get(id=request.session['userId'])  x=Test\_category.objects.filter(email=request.session['user'])  return render(request,'index1.html',{'b':b,'x':x})  return redirect('user:signin') |

Step 2: Create ‘home.html' and ‘profile.html’ and ‘c.html’.

|  |
| --- |
| <!—index.html -->  {% extends 'admin\_base.html' %}  {% load static %}  {% block content %}  <h1 class="text-center mb-2" style="font-family: 'Adamina';font-size: 40px; color:rgba(255, 165, 0, 1); ">  Welcome {{ a.Laboratorymname}}  </h1>  <div class="dashboard">    <div class="row">  <div class="col-md-6">  <div class="card">  <div class="row m-5">  <div class="col-md-8">  {% comment %} <img src="{% static 'icon/test.png' %}" class="fafa" height="50px" width="50px"><br> {% endcomment %}  <span class="text-muted">Total Patient Test Report</span>  </div>  <div class="col-md-4">  <span class="ts" style="font-weight: 900; font-size: 25px;">{{ c }}</span>  </div>  </div>  </div>  </div>    </div>  <div class="card" style="margin-top:100px; padding:0px 10px 0px 10px ;">  <div class="baner" style="margin: auto; margin-top:-4%; margin-bottom:20px;">  <h3 class="text-center" style="font-weight: bold; padding: 10px 0px 10px 0px;"> Patient Records Report</h3>  </div>  <table class="table table-hover table-striped" id="myTable">  <thead>  <tr>  <th>No</th>  <th>Patient NAME</th>  <th>Patient Email</th>  <th>Test Name</th>  <th>Haemoglobin</th>  <th>Platelet\_Count</th>  <th>Date</th>  <th>Action</th>  </tr>  </thead>  <tbody>  {% for i in b %}  <tr>  <td>{{forloop.counter}}</td>  <td>{{ i.patientname }}</td>  <td>{{ i.email }}</td>  <td>{{ i.Test\_name }}</td>  <td>{{ i.Haemoglobin }}</td>  <td>{{ i.Platelet\_Count }}</td>  <td>{{ i.date\_of\_test }}</td>  <td>  <a href="{% url 'laboratory:view\_test' i.id %}">view</a>  <a href="{% url 'laboratory:delete\_test' i.id %}"><img src="{% static 'icon/delete.png' %}" height="30px"  width="30px"></a>  </td>  </tr>  {% endfor %}  </tbody>  </table>  </div>  </div>    {% endblock %}  <!—index1.html -->  {% extends 'user\_base.html' %}  {% load static %}  {% block content %}  <h1 class="text-center mb-2" style="font-family: 'Adamina';font-size: 40px; color:rgba(255, 165, 0, 1); ">  Welcome {{ b.userfname}}  </h1>  <div class="dashboard">      <div class="card" style="margin-top:100px; padding:0px 10px 0px 10px ;">  <div class="baner" style="margin: auto; margin-top:-4%; margin-bottom:20px;">  <h3 class="text-center" style="font-weight: bold; padding: 10px 0px 10px 0px;"> Patient Records Report</h3>  </div>  <table class="table table-hover table-striped" id="myTable">  <thead>  <tr>  <th>No</th>  <th>Patient NAME</th>  <th>Patient Email</th>  <th>Test Name</th>  <th>Haemoglobin</th>  <th>Platelet\_Count</th>  <th>Date</th>  <th>Action</th>  </tr>  </thead>  <tbody>  {% for i in x %}  <tr>  <td>{{forloop.counter}}</td>  <td>{{ i.patientname }}</td>  <td>{{ i.email }}</td>  <td>{{ i.Test\_name }}</td>  <td>{{ i.Haemoglobin }}</td>  <td>{{ i.Platelet\_Count }}</td>  <td>{{ i.date\_of\_test }}</td>  <td>  <a href="{% url 'user:view\_test' i.id %}">view</a>  {% comment %} <a href="{% url 'user:pdf' i.id %}">Print</a> {% endcomment %}    </td>  </tr>  {% endfor %}  </tbody>  </table>  </div>  </div>    {% endblock %} |

Step 3: ‘urls.py’ is a Python module where you define the URL patterns for your web application. It acts as a router that maps URLs to views. When a user makes a request to your Django application, Django uses the urls.py file to determine which view should handle the request.

|  |
| --- |
| from django.urls import path  from laboratoryapp.views import \*  urlpatterns = [  # signin,signup,indexpage  path('',Lab\_signin,name="lab\_signin"),  path('signup/',Lab\_signup,name="lab\_signup"),  path('lab/',lab\_home,name="index"),  #profile  path('lab\_profile/',lab\_profile,name="lab\_profile"),  #Logout  path('logout/',logout,name='logout'),  #add category  path('add\_category/',add\_category,name='add\_category'),  #view category  path('view\_category/',view\_category,name='view\_category'),  #edit category  path('edit\_category/<int:id>/',edit\_category,name='edit\_category'),  path('delete\_category/<int:id>/',delete\_category,name='del\_category'),  #appointment show  path('show\_appointment/',show\_appo,name='show\_appointment'),  # reject appointment  # path('delete\_test/<int:id>',delete\_appointment,name='delete\_appo'),  #book appointment  path('edit/<int:id>/',edit\_appo,name="edit\_appo"),  # show approved appointment  #book appointment  path('show\_test',view\_approvedappo,name="view\_approvedappo"),  #take test  path('take\_test/<int:id>/',take\_test,name='take\_test'),  #view test  path('view\_test/<int:id>/',view\_test,name='view\_test'),  path('delete\_test/<int:id>/',delete\_test,name='delete\_test'),  path('read\_feedback/',view\_feedback,name='view\_feedback'),  path('read\_feedback/<int:id>/',show\_feedback,name='show\_feedback'),  ] |

|  |
| --- |
| from django.urls import path  from patientapp.views import \*  urlpatterns = [  #User Signin,Signup,Logout,IndexpagenIndexpage1  path('',signin,name="signin"),  path('index/',index,name="index"),  path('signup/',signup,name="signup"),  path('logout/',logout,name='logout'),  path('book\_appointment/',book\_appointment,name="book\_appointment"),  path('view\_test/<int:id>',view\_test,name='view\_test'),  path('view\_appointment\_status/',view\_appo\_status,name="view\_approvedappo"),  path('delete\_appo/<int:id>',delete\_appointment,name='delete\_appointment'),  path('send\_feedback/',user\_feedback,name='user\_feedback'),  path('user\_profile/',user\_profile,name='user\_profile'),  # path('Generatedpdf/<int:id>',GeneratePdf,name="pdf"),  ] |

**Day 5**

**“Patient Details”**

**Prompt: All the information about patients**

Step 1: Go to ‘views.py’ location and write code.

|  |
| --- |
| def view\_test(request,id):  if 'user' in request.session.keys():  b=Test\_category.objects.get(pk=id)  c=UserRegister.objects.get(uid=b.email)    return render(request,'datatable.html',{'b':b,'c':c})  else:  return redirect('user:signin') |

Step 2: Create ‘take\_bloodtest.html’

|  |
| --- |
| {% extends 'admin\_base.html' %}  {% load static %}  {% block content %}  <div class="dashboard">    <div class="card" style="margin-top:100px; padding:0px 10px 0px 10px ;">  <div class="baner" style="margin: auto; margin-top:-4%; margin-bottom:20px;">  <h3 class="text-center" style="font-weight: bold; padding: 10px 0px 10px 0px;"> Patient Approved Records</h3>  </div>    <table class="table table-hover table-striped" id="myTable">  <thead>  <tr>  <th>No</th>  <th>Patient Name</th>  <th>Phone</th>  <th>Email</th>  <th>Schedule</th>  <th>Test type</th>    <th>Action</th>    </tr>  </thead>    <tbody>    {% for x in b %}  <tr>  <td>{{forloop.counter}}</td>  <td>{{x.name}}</td>  <td>{{x.phone}}</td>  <td>{{x.email}}</td>  <td>{{x.schedule}}</td>  <td>{{x.Test\_name}}</td>  <td><a class="btn btn-danger" href="{% url 'laboratory:take\_test' x.id %}">Test</a></td>  </tr>  {% endfor %}    </tbody>  </tbody>    </table>    </div>  </div>    {% endblock %} |

Step 3: Go to ‘models.py’ location and build model for product.

|  |
| --- |
| # models.py  class Test\_category(models.Model):  l\_name=models.CharField(max\_length=200,default='',verbose\_name='Lab Assistance Name')  lab\_name=models.CharField(max\_length=200,default='',verbose\_name='Lab Name')  Test\_name=models.CharField(max\_length=200,default='',verbose\_name='Blood Test Name')  patientname=models.CharField(max\_length=200,default='',verbose\_name='Patient Name')  email = models.EmailField(default="")  Haemoglobin=models.PositiveIntegerField(default="")  RBC\_Count=models.PositiveIntegerField(default="")  PCV=models.PositiveIntegerField(default="")  MCV=models.PositiveIntegerField(default="")  MCH=models.PositiveIntegerField(default="")  MCHC=models.PositiveIntegerField(default="")  RDW=models.PositiveIntegerField(default="")  Total\_WBC\_Count=models.PositiveIntegerField(default="")  Neutrophils=models.PositiveIntegerField(default="")  Lymphocytes=models.PositiveIntegerField(default="")  Eosinophils=models.PositiveIntegerField(default="")  Monocytes=models.PositiveIntegerField(default="")  Basophils=models.PositiveIntegerField(default="")  Platelet\_Count =models.PositiveIntegerField(default="")  WBCs\_on\_PS=models.CharField(max\_length=50,default="")  date\_of\_test=models.DateField(auto\_created=True,auto\_now=True)    def \_\_str\_\_(self):  return str(self.Test\_name)  class Reference\_Test\_category(models.Model):  Haemoglobin\_reference = models.TextField(default="male : 14 - 16g \n Female : 12 - 14 g")  RBC\_reference = models.TextField(default="14 - 16g%")  PCV\_reference = models.TextField(default="35 - 45 %")  MCV\_reference = models.TextField(default="80 - 99 fl")  MCH\_reference = models.TextField(default="28 - 32 pg")  MCHC\_reference = models.TextField(default="30 - 34 %")  RDW\_reference = models.TextField(default="9 - 17 fl")  Total\_WBC\_Count\_reference = models.TextField(default="4000 - 11000 / cu.mm")  Neutrophils\_reference = models.TextField(default="40 - 75 %")  Lymphocytes\_reference = models.TextField(default="20 - 45 %")  Eosinophils\_reference = models.TextField(default="00 - 06 %")  Monocytes\_reference = models.TextField(default="00 - 10 %")  Basophils\_reference = models.TextField(default="00 - 01 %")  Platelet\_Count\_reference = models.TextField(default="150000 - 450000 / cu.mm")    def \_\_str\_\_(self):  return str(self.Test\_name) |

Step 4: Migrate Models, after defining models, run the `makemigrations` and `migrate` commands to create corresponding database tables based on the model definitions.

|  |
| --- |
| python manage.py makemigrations  python manage.py migrate |

**Day 6**

**“Feedbacks”**

**Prompt: Get and view feedbacks from the patients**

Step 1: Go to ‘views.py’ location and write code for feedback.

|  |
| --- |
| def user\_feedback(request):  if 'user' in request.session.keys():  c=UserRegister.objects.get(id=request.session['userId'])  print(c)  if request.POST:  name=request.POST['name']  email=request.POST['email']  Feedback=request.POST['Feedback']  a=Userfeedback()  a.username=name  a.useremail=email  a.feedback=Feedback  a.save()  return redirect('user:index')  return render(request,'user\_feedback.html',{'n':c})  else:  return redirect('user:signin') |

|  |
| --- |
| def view\_feedback(request):  if 'Lid' in request.session:  obj=Userfeedback.objects.all()  return render(request,'view\_feedback.html',{'b':obj})  return redirect('laboratory:lab\_signin')  def show\_feedback(request,id):  if 'Lid' in request.session:  obj=Userfeedback.objects.get(id=id)  return render(request,'show\_feedback.html',{'b':obj})  return redirect('laboratory:lab\_signin') |

Step 2: Create templates for BTS.

|  |
| --- |
| <!—user\_feedback.html -->  {% extends 'user\_base.html' %}  {% load static %}  {% block content %}  <div></div>  <div class="dashboard">  <div class="card" style="margin-top:100px; padding:0px 10px 0px 10px ;">  <div class="baner" style="margin: auto; margin-top:-4%; margin-bottom:20px;">  <h3 class="text-center" style="font-weight: bold; padding: 10px 0px 10px 0px;"> Feedback</h3>  </div>  <form method="post" class="col-8 mt-5" style="margin:auto;">  {% csrf\_token%}  <div class="form-group">  <label>Patient name</label>  <input type="text" placeholder="Enter name" class="form-control" name="name"  value="{{n.userfname}}" readonly>  </div>  <div class="form-group">  <label>Patient email</label>  <input type="email" placeholder="Enter email" class="form-control" name="email"  value="{{n.uid}}" readonly>  </div>  <div class="form-group">  <label>write Feedback</label>  <textarea type="text" placeholder="Enter password" class="form-control"  name="Feedback" required></textarea>  </div>  <br>  <input type="submit" class="btn btn-block" value="submit" />  </form>  </br>  </div>  </div>  {% endblock %} |

|  |
| --- |
| <!—view\_feedback.html 🡪  {% extends 'admin\_base.html' %}  {% load static %}  {% block content %}  <div class="dashboard">    <div class="card" style="margin-top:100px; padding:0px 10px 0px 10px ;">  <div class="baner" style="margin: auto; margin-top:-4%; margin-bottom:20px;">  <h3 class="text-center" style="font-weight: bold; padding: 10px 0px 10px 0px;"> Patient Feedback Records </h3>  </div>  <table class="table table-hover table-striped" id="myTable">  <thead>  <tr>  <th>No</th>  <th>Patient NAME</th>  <th>Patient Email</th>  <th>feedback Date</th>  <th>Read</th>  </tr>  </thead>  <tbody>  {% for i in b %}  <tr>  <td>{{forloop.counter}}</td>  <td>{{ i.username }}</td>  <td>{{ i.useremail }}</td>  <td>{{ i.date\_of\_feedback }}</td>    <td>  <a href="{% url 'laboratory:show\_feedback' i.id %}">view</a>    </td>  </tr>  {% endfor %}  </tbody>  </table>  </div>  </div>  {% endblock %} |

|  |
| --- |
| <!—show\_feedback.html 🡪  {% extends 'admin\_base.html' %}  {% load static %}  {% block content %}  <div></div>  <div class="dashboard">  <div class="card" style="margin-top:100px; padding:0px 10px 0px 10px ;">  <div class="baner" style="margin: auto; margin-top:-4%; margin-bottom:20px;">  <h3 class="text-center" style="font-weight: bold; padding: 10px 0px 10px 0px;"> Feedback</h3>  </div>  <form method="post" class="col-8 mt-5" style="margin:auto;">  {% csrf\_token%}  <div class="form-group">  <label>Patient name</label>  <input type="text" class="form-control" name="name"  value="{{b.username}}" readonly>  </div>  <div class="form-group">  <label>Patient email</label>  <input type="email" class="form-control" name="email"  value="{{b.useremail}}" readonly>  </div>  <div class="form-group">  <label>write Feedback</label>  <textarea type="text" placeholder="Enter password" class="form-control"  required>{{b.feedback}}</textarea>  </div>  <br>    </form>  </br>  </div>  </div>  {% endblock %} |

Step 3: Go to ‘models.py’ location and build model for feedback.

|  |
| --- |
| # app1 /models.py  class Userfeedback(models.Model):  username=models.CharField(max\_length=30,default='',verbose\_name='Patient Name')  useremail=models.EmailField(max\_length=50,verbose\_name='Patient Email')  feedback=models.TextField(verbose\_name='Patient feedback Area')  date\_of\_feedback=models.DateField(auto\_created=True,auto\_now=True)  def \_\_str\_\_(self):  return self.username |

Step 4: Migrate Models, after defining models, run the `makemigrations` and `migrate` commands to create corresponding database tables based on the model definitions.

|  |
| --- |
| python manage.py makemigrations  python manage.py migrate |

**Day 7**

**“Patient Appointments”**

**Prompt: Patients Appointment Information**

Step 1: Go to ‘views.py’ location and write code.

|  |
| --- |
| import datetime  from django.utils.dateparse import parse\_datetime  def book\_appointment(request):  if 'user' in request.session.keys():  a=Name\_category.objects.all()  b=UserRegister.objects.get(id=request.session['userId'])  form = appointmentform(request.POST)  #print(form)  if form.is\_valid():  c=Appointment.objects.filter(email=request.POST['email']) & Appointment.objects.filter(status='pending')  if len(c)<=0:  cDate = datetime.datetime.today()  postDate = request.POST['schedule']  postDateArray = postDate.split("T")  cDateArray = str(cDate).split(" ")  if parse\_datetime(postDateArray[0]) >= parse\_datetime(cDateArray[0]):  form.save()  messages.success(request,'Your appoinement is booked.')  else:  return render(request,'book\_appointment.html',{'form':form,'ab':a,'b':b,'m':'select future date only'})  else:  return render(request,'book\_appointment.html',{'form':form,'ab':a,'b':b,'m':'appointment alredy booked'})  return render(request,'book\_appointment.html',{'form':form,'ab':a,'b':b})  return redirect('user:signin')  def view\_appo\_status(request):  if 'user' in request.session.keys():  b=Appointment.objects.filter(email=request.session['user'])  return render(request,'view\_appointment.html',{'b':b})  else:  return redirect('user:signin')  def delete\_appointment(request,id):  if 'user' in request.session.keys():  obj=Appointment.objects.get(pk=id)  obj.delete()  return redirect('user:view\_approvedappo')  return redirect('user:signin') |

Step 2: Create templates for patient appointments.

|  |
| --- |
| <!—book\_appointment..html -->  {% load static %}  <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>Book\_appoinement</title>  <!-- <link rel="stylesheet" type="text/css" href="{% static 'css/style.css' %}"> -->  <style>  body{  margin: 0;  padding: 300px;    background: #34495e;  background-size: 100%;  background-position: 100%;  font-family: sans-serif;  }  body{  margin: 0;  padding: 0;  font-family: sans-serif;  background: #34495e;  }  .book-appoinement{  width: 1px;  height: 470px;  background: black;  color: #fff;  top: 50%;  left: 50%;  position: absolute;  transform: translate(-50%,-50%);  box-sizing: border-box;  padding: 70px 30px;  }  .book-appoinement input[type="text"], input[type="email"]  {  border: none;  border-bottom: 1px solid #fff;  padding-right: 10%;  background: transparent;  outline: none;  height: 40px;  color: #fff;  font-size: 19px;  }  .box input[type = "datetime-local"]{  border:0;  background: none;  display: block;  margin: 20px auto;  text-align: center;  border: 2px solid #3498db;  padding: 14px 10px;  width: 200px;  outline: none;  color: white;  border-radius: 24px;  transition: 0.25s;  }  .box input[type = "text"]:focus{  width: 280px;  border-color: #2ecc71;  }  .book-appoinement select[id="id\_department"]{  background-color: transparent;    color: white;  padding-right: 5%;  }  .button {  background-color: #4c90af;  border: none;  color: white;  padding: 15px 32px;  text-align: center;  text-decoration: none;  display: inline-block;  font-size: 19px;  }  .time {  background-color: #6f7a6f81; /\* Green \*/  border: none;  color: white;  margin-right: 10%;  /\* text-align: center; \*/  text-decoration: none;  /\* display: inline-block; \*/  font-size: 16px;  }  option{  color: rgb(48, 47, 47);  font-size: larger;  }  .box{  width: 300px;  padding: 40px;  position: absolute;  top: 50%;  left: 50%;  transform: translate(-50%,-50%);  background: #191919;  text-align: center;  }  </style>  </head>  <body>    <div class="book-appoinement">    <center>  <!-- style="padding-top: 250px;  font-size: x-large;" -->  <form method="POST" class="box">  {% csrf\_token %}  <div>  <h1>Book AppointMent</h1>  <label>Name:</label>  <input type="text" name="name" value="{{b.userfname}}" readonly>  </div>  <br>  <div class="row">  <div>  <label>Email:</label>  <input type="email" name="email" value="{{b.uid}}" readonly >  </div></div>  <br>  <div>  <label>Phone:</label>  <input type="text" name="phone" placeholder="Enter phone number" value="{{b.usercontactno}}" readonly>  </div>  <br>  <div class="d-department">  <label for="id\_department">Test Name:</label>    <select name="Test\_name" required id="id\_department">  {% for i in ab %}  <option value="{{i.id}}"  selected="selected">{{i.category\_name}}</option>  {% endfor %}  </select>    </div>  <br>  <div>  <label>Schedule:</label>  <input type="datetime-local" name="schedule" class="time">  </div>  <br>  <input type="hidden" name="status" placeholder="status" value="pending" readonly>  {% if messages %}  {% for result in messages %}  <center> <b style=color:green;>{{result}}</b></center>  {% endfor %}  {% endif %}  <br>    <button class="button">Submit here</button>  <br><br>  <a href="{% url 'user:index' %}" style="color: teal;"> Home Page </a>  </form>  </center>  </div>  {% if m %}  <script>alert("{{m}}")</script>  {% endif %}  </body>  </html> |

Step 3: Go to ‘models.py’ location and build model for appointment.

|  |
| --- |
| # models.py  class Appointment(models.Model):  g = (  ('pending', 'pending'),  ('approved', 'approved'),  ('rejected', 'rejected'),  )  name = models.CharField(max\_length=500)  phone = models.IntegerField()  email = models.EmailField(max\_length=100)  schedule = models.DateTimeField()  Test\_name=models.ForeignKey(Name\_category, on\_delete=models.CASCADE)  status = models.CharField(max\_length=12, choices=g, default = 'pending')  appointment\_booked=models.BooleanField(default=False)  def \_\_str\_\_(self):  return self.name |

Step 4: Migrate Models, after defining models, run the `makemigrations` and `migrate` commands to create corresponding database tables based on the model definitions.

|  |
| --- |
| python manage.py makemigrations  python manage.py migrate |

**Day 8**

**“Status Details”**

**Prompt: Appointment Status on Lab Side**

Step 1: Go to ‘app1/views.py’ location and write code for HBS.

|  |
| --- |
| def show\_appo(request):  results=Appointment.objects.filter(status='pending')  return render(request,'show\_appointment.html',{'book':results})  #book appointment  def edit\_appo(request,id):  if 'Lid' in request.session.keys():  a=LaboratoryRegister.objects.get(Lid=request.session['Lid'])  book=Appointment.objects.get(pk=id)  if request.POST:  book.status=request.POST['boked']  book.save()  return redirect('laboratory:show\_appointment')  return render(request,'edit\_appointment.html',{'book':book,'owner\_data':a})  else:  return redirect('laboratory:lab\_signin')  def view\_approvedappo(request):  if 'Lid' in request.session:  b=Appointment.objects.filter(status='approved') & Appointment.objects.filter(appointment\_booked=False)    return render(request,'take\_bloodtest.html',{'b':b})  else:  return redirect('laboratory:lab\_signin') |

Step 2: Create template “edit\_appointment.html” for BTS.

|  |
| --- |
| {% extends 'admin\_base.html' %}  {% load static %}  {% block content %}  <!--/header-->  <section class="container">  <div class="row justify-content-center p-4">  <div class="col-12 col-md-6 p-5 card">  <h3 class="text-uppercase font-weight-bold text-center">Appointment Edit</h3>  <form method="POST" enctype="multipart/form-data" >  {% csrf\_token %}  <br />  <div class="form-group row">  <label for="staticName" class="col-4 col-form-label">Patient Name</label>  <div class="col-auto">  <input type="text" readonly class="form-control" id="staticName" value="{{ book.name }}" />  </div>  </div>  <div class="form-group row">  <label for="staticOwner" class="col-4 col-form-label">Phone</label>  <div class="col-auto">  <input  type="text"  readonly  class="form-control"  id="staticOwner"  value="{{ book.phone }}"  />  </div>  </div>  <div class="form-group row">  <label for="staticProperty" class="col-4 col-form-label"  >Email</label  >  <div class="col-auto">  <input  type="text"  readonly  class="form-control"  id="staticProperty"  value="{{ book.email }}"  />  </div>  </div>  <div class="form-group row">  <label for="staticName" class="col-4 col-form-label">Schedule</label>  <div class="col-auto">  <input type="text" readonly class="form-control" id="staticName" value="{{ book.schedule }}" />  </div>  </div>  <div class="form-group row">  <label for="staticName" class="col-4 col-form-label">Test type</label>  <div class="col-auto">  <input type="text" readonly class="form-control" id="staticName" value="{{ book.Test\_name }}" />  </div>  </div>  <div class="form-group row">  <label  class="col-auto col-form-label d-inline-block"  for="customCheck1"  >Want to Approve:</label  >  <div class="col-7">  <select class="form-control w-100 text-center" id="boked" name="boked">  <option value="none" selected>select</option>  {% comment %} <option value="pending">pending</option> {% endcomment %}  <option value="approved">Approve</option>  <option value="rejected">Reject</option>  </select>  </div>  </div>  <div class="form-group row">  <div class="col-12">  <input type="submit" value="submit" class="btn btn-primary btn-block" />  </div>  </div>  </form>  </div>  </div>  </section>  <!-- //News section -->  <!-- footers 20 -->  {% endblock content %} |

**Day 9**

**“Profile”**

**Prompt: Fetch profile details**

Step 1: Go to ‘views.py’ location and write code for profile.

|  |
| --- |
| def lab\_profile(request):  if 'Lid' in request.session:  a=LaboratoryRegister.objects.get(Lid=request.session['Lid'])  if request.POST:  name=request.POST['name']  email=request.POST['email']  password=request.POST['passowrd']  a.Laboratoryfname=name  a.Lid=email  a.Laboratorypwd=password  a.save()  return redirect('laboratory:index')  return render(request,'lab\_profile.html',{'n':a})  else:  return redirect('laboratory:lab\_signin')  def user\_profile(request):  if 'user' in request.session.keys():  a=UserRegister.objects.get(id=request.session['userId'])  if request.POST:  name=request.POST['name']  email=request.POST['email']  password=request.POST['passowrd']  a.userfname=name  a.uid=email  a.userpwd=password  a.save()  return redirect('user:index')  return render(request,'user\_profile.html',{'n':a})  else:  return redirect('user:signin') |

Step 2: Template “lab\_profile.html”.

|  |
| --- |
| {% extends 'admin\_base.html' %}  {% load static %}  {% block content %}  <div></div>  <div class="dashboard">  <div class="card" style="margin-top:100px; padding:0px 10px 0px 10px ;">  <div class="baner" style="margin: auto; margin-top:-4%; margin-bottom:20px;">  <h3 class="text-center" style="font-weight: bold; padding: 10px 0px 10px 0px;"> Edit Laboratory Profile</h3>  </div>  <form method="post" class="col-8 mt-5" style="margin:auto;">  {% csrf\_token%}  <div class="form-group">  <label>Enter name</label>  <input type="text" placeholder="Enter name" class="form-control" name="name"  value="{{n.Laboratoryfname}}" required>  </div>  <div class="form-group">  <label>Enter email</label>  <input type="email" placeholder="Enter email" class="form-control" name="email"  value="{{n.Lid}}" readonly>  </div>  <div class="form-group">  <label>Enter password</label>  <input type="password" placeholder="Enter password" class="form-control"  name="passowrd" value="{{n.Laboratorypwd}}" required>  </div>  <br>  <input type="submit" class="btn btn-block" value="submit" />  </form>  </br>  </div>  </div>  {% endblock %} |

**Day 10**

**“Test Reports”**

**Prompt: Final Test Reports Information**

Step 1: Go to ‘views.py’ location and write code.

|  |
| --- |
| def take\_test(request,id):  if 'Lid' in request.session:  a=request.session['fname']  c=request.session['name']  b=Appointment.objects.get(pk=id)  form =Testform(request.POST)  if form.is\_valid():  b.appointment\_booked=request.POST['test']  form.save()  b.save()    return redirect('laboratory:index')  return render(request,'bloodtest.html',{'a':a,'b':b,'c':c})  else:  return redirect('laboratory:lab\_signin')  def view\_test(request,id):  if 'user' in request.session.keys():  b=Test\_category.objects.get(pk=id)  c=UserRegister.objects.get(uid=b.email)    return render(request,'datatable.html',{'b':b,'c':c})  else:  return redirect('user:signin') |

Step 2: Go to ‘models.py’ location and build model.

|  |
| --- |
| # models.py  class Reference\_Test\_category(models.Model):  Haemoglobin\_reference = models.TextField(default="male : 14 - 16g \n Female : 12 - 14 g")  RBC\_reference = models.TextField(default="14 - 16g%")  PCV\_reference = models.TextField(default="35 - 45 %")  MCV\_reference = models.TextField(default="80 - 99 fl")  MCH\_reference = models.TextField(default="28 - 32 pg")  MCHC\_reference = models.TextField(default="30 - 34 %")  RDW\_reference = models.TextField(default="9 - 17 fl")  Total\_WBC\_Count\_reference = models.TextField(default="4000 - 11000 / cu.mm")  Neutrophils\_reference = models.TextField(default="40 - 75 %")  Lymphocytes\_reference = models.TextField(default="20 - 45 %")  Eosinophils\_reference = models.TextField(default="00 - 06 %")  Monocytes\_reference = models.TextField(default="00 - 10 %")  Basophils\_reference = models.TextField(default="00 - 01 %")  Platelet\_Count\_reference = models.TextField(default="150000 - 450000 / cu.mm")    def \_\_str\_\_(self):  return str(self.Test\_name) |

Step 4: Migrate Models, after defining models, run the `makemigrations` and `migrate` commands to create corresponding database tables based on the model definitions.

|  |
| --- |
| python manage.py makemigrations  python manage.py migrate |

**Day 11**

**“Queries handling in Django framework”**

Object-Relational Mapping (ORM) is an integral part of the framework, enabling developers to interact with databases using Python objects. Django's ORM abstracts away the complexity of SQL queries, allowing developers to focus on application logic rather than database operations directly. Here's an overview of how ORM queries work in Django:

1. Define Models: Django models are Python classes that represent database tables. Each attribute in the class corresponds to a field in the table. Models are defined in the `models.py` file of Django apps.

|  |
| --- |
| class Blog(models.Model):  name = models.CharField(max\_length=50)  tagline = models.TextField()  # author = models.CharField(max\_length=20,default = "")  def \_\_str\_\_(self):  return self.name  class Author(models.Model):  name = models.CharField(max\_length=200)  email = models.EmailField()  def \_\_str\_\_(self):  return self.name |

2. Migrate Models: After defining models, run the `makemigrations` and `migrate` commands to create corresponding database tables based on the model definitions.

|  |
| --- |
| python manage.py makemigrations  python manage.py migrate |

3. Perform Queries: Django provides a powerful API for querying databases using model objects. Various methods are available to perform CRUD operations, filter data, perform aggregations, and more.

|  |
| --- |
| def table(request):  a = userregister.objects.get(name = 'a')  print(a)  for i in a:  print(i.email)  return render(request,'table.html',{'data':a}) |
|  |

**Day 12**

**Custom Validations in the Django Models**

Custom field validations in Django models can be implemented using the `clean()` method or by overriding the `save()` method. Here's how you can implement custom field validations using both methods:

|  |
| --- |
| class vendor(models.Model):  name = models.CharField(max\_length=50)  email = models.EmailField()  mob = models.CharField(max\_length=10)  add = models.TextField()  password = models.CharField(max\_length=10)  typeofbusiness = models.CharField(max\_length = 100)  def \_\_str\_\_(self):  return self.name  class order(models.Model):  userid = models.CharField(max\_length = 10)  proid = models.CharField(max\_length = 10)  totalamount = models.CharField(max\_length = 10)  qty = models.CharField(max\_length = 10)  add = models.TextField()  city = models.CharField(max\_length = 30)  state =models.CharField(max\_length = 30)  pincode = models.CharField(max\_length = 6)  paymentype = models.CharField(max\_length = 30)  transactionid = models.CharField(max\_length = 100)  datetime = models.DateTimeField(auto\_now = True)  def \_\_str\_\_(self):  return self.userid |

The `clean()` method is overridden to perform custom validation on the `your\_field` attribute. If the validation fails, a `ValidationError` is raised. Make sure to call the parent class's `clean()` method at the end of your custom `clean()` method.

|  |
| --- |
| class cart(models.Model):  productid = models.CharField(max\_length=10)  userid = models.CharField(max\_length=10)  quantity = models.CharField(max\_length=10)  totalprice = models.CharField(max\_length=10)  orderid = models.CharField(max\_length=10)  def \_\_str\_\_(self):  return self.userid |

The `save()` method is overridden to perform custom validation before saving the object. If the validation fails, a `ValidationError` is raised. Make sure to call the parent class's `save()` method at the end of your custom `save()` method.

Choose the method that best fits your use case. Generally, the `clean()` method is preferred for field-specific validations, while the `save()` method is more suitable for object-level validations.

In Django, the `Meta` inner class within a model allows you to define metadata about the model. This metadata can include options such as database table name, ordering, indexes, permissions, and more. Here's an example of how you can use the `Meta` class in Django models:

- The `Meta` class is used to define metadata for the `YourModel` class.

- `verbose\_name` specifies a human-readable name for the model. It's used in various Django admin interfaces.

- `verbose\_name\_plural` specifies the plural form of the model name.

- `ordering` specifies the default ordering for queries on this model. In this case, it orders the query results by the `field2` attribute in descending order.

You can include other options in the `Meta` class based on your requirements. Here are some other common options:

- `db\_table`: Specifies the name of the database table for the model.

- `unique\_together`: Specifies sets of fields that, taken together, must be unique.

- `indexes`: Specifies database indexes for one or more fields.

- `permissions`: Specifies permissions associated with the model.

**Day 13**

**Filters in Python**

In Python, filters are a way to selectively extract elements from a collection (like a list) based on a condition. They're particularly useful when you want to apply a certain criterion to each element of a collection and only retain those elements that satisfy the criterion.

The `filter()` function in Python takes two arguments: a function (or `None`) and an iterable (like a list). It returns an iterator yielding those items of the iterable for which the function returns true. If the function is `None`, it simply returns the elements that are true.

Here's a basic example to illustrate how `filter()` works:

|  |
| --- |
| def index(request):  if 'email' in request.session:  b = userregister.objects.get(email = request.session['email'])  a = category.objects.all()  return render(request,'index.html',{'category':a,'session':b})  elif 'vendoremail' in request.session:  a = category.objects.all()  b = vendor.objects.get(email = request.session['vendoremail'])  return render(request,'index.html',{'category':a,'vensession':b})  else:  a = category.objects.all()  return render(request,'index.html',{'category':a}) |

|  |
| --- |
| def reg(request):  # c = userregister.objects.get(name = 'b')  if request.method == 'POST':  a = userregister()  a.name = request.POST['uname']  a.email = request.POST['email']  a.mob = request.POST['mob']  a.add = request.POST['add']  pasword = request.POST['password']  print(pasword,"1111111111111111111111111111111111111")  encrypted\_pass = encrypt(pasword)  print(encrypted\_pass,"2222222222222222222222")  # check\_password(encrypted\_pass, pasword)  a.password = encrypted\_pass  b = userregister.objects.filter(email = request.POST['email'])  error\_msg = None  if a.email:  if len(a.mob) == 10:  if len(b) > 0:  return render(request,'register.html',{'email':'This email is already registered..'})  else:  if request.POST['password'] == request.POST['cp']:  a.save()  return render(request,'register.html',{'save':'Data stored succesfully....'})  else:  return render(request,'register.html',{'pass':'Passwords did not matched...'})  else:  error\_msg = "Phone number must be 10 didgits.."  return render(request,'register.html',{'error':error\_msg})  else:  error\_msg = "Email field is required.."  return render(request,'register.html',{'error':error\_msg})  else:  return render(request,'register.html',{}) |

This is a basic example of how you can implement product functionality using ORM queries with SQLAlchemy in Python. Depending on your specific requirements and database schema, you may need to adjust the code accordingly.

**Day 14**

**Web Page and Navigation using Session**

To build a common web page with navigation using sessions in Python, you typically use a web framework like Flask or Django. Here's an example using Flask, a lightweight and easy-to-use web framework:

1. First, you need to install Flask if you haven't already:

|  |
| --- |
| pip install flask |

2. Now, let's create a simple Flask application that demonstrates session-based navigation. We'll have three pages: Home, About, and Contact.

|  |
| --- |
| from flask import Flask, render\_template, session, redirect, url\_for  app = Flask(\_\_name\_\_)  app.secret\_key = 'your\_secret\_key' # Needed for session encryption  @app.route('/')  def home():  return render\_template('home.html')  @app.route('/about')  def about():  return render\_template('about.html')  @app.route('/contact')  def contact():  return render\_template('contact.html')  @app.route('/login')  def login():  # Here you can set session variables like user authentication status, etc.  session['logged\_in'] = True  return redirect(url\_for('home'))  @app.route('/logout')  def logout():  # Clear session variables  session.clear()  return redirect(url\_for('home'))  if \_\_name\_\_ == '\_\_main\_\_':  app.run(debug=True) |

3. Create HTML templates for each page. You can use Jinja2 templating engine provided by Flask for dynamic content.

|  |
| --- |
| `templates/home.html`:  <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>Home</title>  </head>  <body>  <h1>Home Page</h1>  <p>Welcome to our website!</p>  <nav>  <a href="/">Home</a> |  <a href="/about">About</a> |  <a href="/contact">Contact</a>  {% if session.logged\_in %}  | <a href="/logout">Logout</a>  {% else %}  | <a href="/login">Login</a>  {% endif %}  </nav>  </body>  </html> |

|  |
| --- |
| `templates/about.html`:  <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>About</title>  </head>  <body>  <h1>About Us</h1>  <p>We are a company that does amazing things.</p>  <nav>  <a href="/">Home</a> |  <a href="/about">About</a> |  <a href="/contact">Contact</a>  {% if session.logged\_in %}  | <a href="/logout">Logout</a>  {% else %}  | <a href="/login">Login</a>  {% endif %}  </nav>  </body>  </html> |

|  |
| --- |
| `templates/contact.html`:  <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>Contact</title>  </head>  <body>  <h1>Contact Us</h1>  <p>You can reach us at: example@example.com</p>  <nav>  <a href="/">Home</a> |  <a href="/about">About</a> |  <a href="/contact">Contact</a>  {% if session.logged\_in %}  | <a href="/logout">Logout</a>  {% else %}  | <a href="/login">Login</a>  {% endif %}  </nav>  </body>  </html> |

4. Run your Flask application:

|  |
| --- |
| python your\_app\_name.py |

Now, you can navigate through the pages and see that the navigation bar updates based on whether the user is logged in or not. The `session` object is used to keep track of the user's authentication status across requests. When the user logs in or logs out, the `session` object is updated accordingly.

**Activity 2:Wishlist Management for the ecommerce in python with django**

**Prompt: Wishlist Management for the ecommerce in python with django**

Implementing wishlist management for an e-commerce platform using Django involves creating models, views, templates, and handling user authentication. Here's a step-by-step guide to achieve this:

1. Set up Django Project: If you haven't already, create a Django project and app:

```bash

django-admin startproject ecommerce\_project

cd ecommerce\_project

python manage.py startapp wishlist

```

2. \*\*Define Models\*\*: Define the models for the wishlist functionality. A basic model might look like this:

```python

# In wishlist/models.py

from django.db import models

from django.contrib.auth.models import User

class WishlistItem(models.Model):

user = models.ForeignKey(User, on\_delete=models.CASCADE)

product\_name = models.CharField(max\_length=100)

product\_description = models.TextField()

added\_at = models.DateTimeField(auto\_now\_add=True)

def \_\_str\_\_(self):

return self.product\_name

```

3. \*\*Create Views\*\*: Create views to handle adding, viewing, and removing items from the wishlist.

```python

# In wishlist/views.py

from django.shortcuts import render, redirect

from django.contrib.auth.decorators import login\_required

from .models import WishlistItem

@login\_required

def wishlist(request):

wishlist\_items = WishlistItem.objects.filter(user=request.user)

return render(request, 'wishlist/wishlist.html', {'wishlist\_items': wishlist\_items})

@login\_required

def add\_to\_wishlist(request, product\_id):

# Logic to add product to wishlist

pass

@login\_required

def remove\_from\_wishlist(request, product\_id):

# Logic to remove product from wishlist

pass

```

4. \*\*Create URLs\*\*: Define URLs for the wishlist views.

```python

# In wishlist/urls.py

from django.urls import path

from . import views

urlpatterns = [

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

path('add/<int:product\_id>/', views.add\_to\_wishlist, name='add\_to\_wishlist'),

path('remove/<int:product\_id>/', views.remove\_from\_wishlist, name='remove\_from\_wishlist'),

]

```

5. \*\*Set up Templates\*\*: Create templates for displaying the wishlist.

```html

<!-- In templates/wishlist/wishlist.html -->

{% extends 'base.html' %}

{% block content %}

<h1>Wishlist</h1>

{% if wishlist\_items %}

<ul>

{% for item in wishlist\_items %}

<li>{{ item.product\_name }} - {{ item.product\_description }}</li>

{% endfor %}

</ul>

{% else %}

<p>Your wishlist is empty.</p>

{% endif %}

{% endblock %}

```

6. \*\*Wire up Authentication\*\*: Ensure user authentication is set up properly in your Django project.

7. \*\*Include Wishlist URLs\*\*: Include wishlist URLs in the main project's URL configuration.

```python

# In ecommerce\_project/urls.py

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

path('admin/', admin.site.urls),

path('wishlist/', include('wishlist.urls')),

]

```

8. \*\*Run Migrations\*\*: Run migrations to create the necessary database tables.

```bash

python manage.py makemigrations

python manage.py migrate

```

9. \*\*Add Links to Wishlist\*\*: In your e-commerce platform's product pages, add links to add/remove products from the wishlist.

This is a basic outline to get you started with wishlist management in Django. Depending on your specific requirements, you may need to extend this functionality further, such as adding AJAX support for adding/removing items, handling quantity, etc.

**Day 15**

**SQLite Database integration in Django**

SQLite is a lightweight and serverless database engine that is well-suited for development and small-scale applications. Django, a high-level Python web framework, supports SQLite out of the box, making it easy to get started with database-driven web applications.

Here's an overview of how to use SQLite database with Django and how to establish a connection:

1. \*\*Configure Django Settings\*\*:

In your Django project's settings file (`settings.py`), ensure that Django is configured to use SQLite as the default database engine. By default, Django comes pre-configured to use SQLite for development purposes.

```python

# In settings.py

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.sqlite3',

'NAME': BASE\_DIR / 'db.sqlite3',

}

}

```

2. \*\*Create Django Models\*\*:

Define your application's data models using Django's ORM (Object-Relational Mapping). These models will be used to create corresponding database tables.

```python

# In models.py

from django.db import models

class Product(models.Model):

name = models.CharField(max\_length=100)

description = models.TextField()

price = models.DecimalField(max\_digits=10, decimal\_places=2)

```

3. \*\*Make Migrations\*\*:

Django's migration system allows you to propagate changes you make to your models (adding fields, deleting models, etc.) into your database schema.

```bash

python manage.py makemigrations

```

This command creates migration files based on the changes detected in your models.

4. \*\*Apply Migrations\*\*:

Apply the migrations to create the corresponding database tables.

```bash

python manage.py migrate

```

This command executes the migration files to create the database tables.

5. \*\*Interact with the Database\*\*:

Now that your database is set up, you can interact with it using Django's ORM. You can perform CRUD (Create, Read, Update, Delete) operations on your models.

```python

# In views.py

from django.shortcuts import render

from .models import Product

def product\_list(request):

products = Product.objects.all()

return render(request, 'product\_list.html', {'products': products})

```

6. \*\*Run Django Development Server\*\*:

Start the Django development server to test your application.

```bash

python manage.py runserver

```

7. \*\*Access Admin Interface (Optional)\*\*:

Django provides a built-in admin interface that allows you to perform CRUD operations on your models. To access it, you need to create a superuser first.

```bash

python manage.py createsuperuser

```

Follow the prompts to create a superuser, and then you can access the admin interface at `http://localhost:8000/admin`.

That's it! You've established a connection to SQLite database in Python with Django and created models to interact with the database. You can now start building your web application with Django and SQLite.

To use MySQL with Django, you need to set up your Django project to use the MySQL database engine. Here's how you can create and operate a MySQL database in Python with Django:

1. \*\*Install MySQL Database Server\*\*:

First, ensure that you have MySQL installed on your system. You can download and install MySQL from the official website: [MySQL Downloads](https://dev.mysql.com/downloads/)

2. \*\*Install MySQL Client Library\*\*:

You also need to install the MySQL client library for Python. You can install it using pip:

```bash

pip install mysqlclient

```

3. \*\*Create MySQL Database\*\*:

Create a new database in your MySQL server that will be used by your Django project.

4. \*\*Configure Django Settings\*\*:

In your Django project's settings file (`settings.py`), update the database configuration to use MySQL.

```python

# In settings.py

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.mysql',

'NAME': 'your\_database\_name',

'USER': 'your\_mysql\_username',

'PASSWORD': 'your\_mysql\_password',

'HOST': 'localhost', # Or the hostname of your MySQL server

'PORT': '3306', # Default MySQL port

}

}

```

Replace `'your\_database\_name'`, `'your\_mysql\_username'`, and `'your\_mysql\_password'` with your actual MySQL database name, username, and password.

5. \*\*Create Django Models\*\*:

Define your Django models as you would normally. Here's an example:

```python

# In models.py

from django.db import models

class Product(models.Model):

name = models.CharField(max\_length=100)

description = models.TextField()

price = models.DecimalField(max\_digits=10, decimal\_places=2)

```

6. \*\*Make Migrations and Apply Migrations\*\*:

Run the following commands to create and apply migrations:

```bash

python manage.py makemigrations

python manage.py migrate

```

7. \*\*Interact with the Database\*\*:

Now you can interact with your MySQL database using Django's ORM. For example, you can create, read, update, and delete records:

```python

# In views.py

from django.shortcuts import render

from .models import Product

def product\_list(request):

products = Product.objects.all()

return render(request, 'product\_list.html', {'products': products})

```

8. \*\*Run Django Development Server\*\*:

Start the Django development server to test your application:

```bash

python manage.py runserver

```

By following these steps, you can create and operate a MySQL database in Python with Django. Ensure that your MySQL server is running and accessible from your Django project.

**Day 16**

**Razorpay Django**

To integrate Razorpay payment gateway with Django, you need to follow these steps:

1. \*\*Create a Razorpay Account\*\*:

Sign up for a Razorpay account if you haven't already. Once signed up, you will get access to the Razorpay dashboard where you can generate API keys.

2. \*\*Install Razorpay Python SDK\*\*:

You can use the official Razorpay Python SDK to integrate Razorpay with your Django project. Install it using pip:

```bash

pip install razorpay

```

3. \*\*Configure Razorpay API Keys\*\*:

In your Django settings file (`settings.py`), add your Razorpay API keys:

```python

# In settings.py

RAZORPAY\_API\_KEY = 'your\_api\_key'

RAZORPAY\_API\_SECRET = 'your\_api\_secret'

```

4. \*\*Create Payment Form\*\*:

Create a form in your Django template where users can enter payment details:

```html

<!-- In payment\_form.html -->

<form id="paymentForm" action="{% url 'process\_payment' %}" method="POST">

{% csrf\_token %}

<input type="text" name="amount" placeholder="Amount">

<input type="submit" value="Pay Now">

</form>

```

5. \*\*Process Payment View\*\*:

Create a view in Django to handle payment processing:

```python

# In views.py

from django.shortcuts import render

from django.http import JsonResponse

import razorpay

from django.conf import settings

def process\_payment(request):

if request.method == 'POST':

amount = int(request.POST['amount']) \* 100 # Convert amount to paisa (Razorpay expects amount in paisa)

client = razorpay.Client(auth=(settings.RAZORPAY\_API\_KEY, settings.RAZORPAY\_API\_SECRET))

payment\_data = {

'amount': amount,

'currency': 'INR',

'receipt': 'order\_rcptid\_11',

'payment\_capture': '1'

}

payment = client.order.create(data=payment\_data)

return JsonResponse(payment)

return render(request, 'payment\_form.html')

```

6. \*\*Handle Payment Response\*\*:

After the payment is processed, Razorpay will send a webhook or redirect the user back to a URL specified by you. You need to handle this response to update the payment status in your database.

7. \*\*Update Payment Status View\*\*:

Create a view to handle the payment status update:

```python

# In views.py

def update\_payment\_status(request):

if request.method == 'POST':

# Process Razorpay payment webhook or redirect response

# Update payment status in your database

return JsonResponse({'status': 'Payment status updated successfully'})

```

8. \*\*URL Configuration\*\*:

Configure URLs to map views:

```python

# In urls.py

from django.urls import path

from . import views

urlpatterns = [

path('', views.process\_payment, name='process\_payment'),

path('update\_payment\_status/', views.update\_payment\_status, name='update\_payment\_status'),

]

```

9. \*\*Update Templates\*\*:

Update your templates to include the payment form and handle payment responses.

10. \*\*Testing\*\*:

Test the payment flow in your development environment. Ensure that payments are processed successfully and payment status is updated in your database.

Remember to handle exceptions and errors gracefully, and follow best practices for security, such as validating input and using HTTPS for sensitive transactions. Additionally, refer to the Razorpay documentation for more details on handling webhooks and verifying payment responses.

**Day 17**

**Git hub version control**

Version control and collaboration using Git and GitHub are essential for managing Python and Django projects effectively, especially when working in teams or contributing to open-source projects. Here's an overview of how Git and GitHub are used in Python and Django development:

### Version Control with Git:

1. \*\*What is Version Control?\*\*

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later. Git is one of the most popular distributed version control systems.

2. \*\*Basic Concepts:\*\*

- \*\*Repository (Repo):\*\* A repository is a collection of files and their revision history. It can exist locally on your machine or remotely on a server.

- \*\*Commit:\*\* A commit is a snapshot of the repository at a specific point in time. It records changes to one or more files.

- \*\*Branch:\*\* A branch is a parallel version of a repository. It allows you to work on new features or fixes without affecting the main codebase.

- \*\*Merge:\*\* Merging is the process of combining changes from one branch into another.

- \*\*Pull Request (PR):\*\* A pull request is a request to merge changes from one branch into another. It's commonly used for code review and collaboration.

3. \*\*Workflow:\*\*

- \*\*Feature Branch Workflow:\*\* Developers create a new branch for each new feature or bug fix. Once the changes are complete, they create a pull request to merge their branch into the main branch.

### Collaboration with GitHub:

1. \*\*What is GitHub?\*\*

GitHub is a web-based platform that provides hosting for Git repositories. It offers collaboration features such as issue tracking, pull requests, and project management tools.

2. \*\*Key Features:\*\*

- \*\*Repositories:\*\* GitHub hosts repositories where you can store your code.

- \*\*Issues:\*\* You can create, assign, and track issues (bugs, feature requests, etc.) using GitHub's issue tracker.

- \*\*Pull Requests:\*\* Pull requests facilitate code review and collaboration. They allow contributors to propose changes and discuss them before merging into the main codebase.

- \*\*Branch Protection:\*\* GitHub allows you to protect certain branches from direct pushes and require pull requests for changes, ensuring code quality and stability.

- \*\*Actions and Workflows:\*\* GitHub Actions enable you to automate workflows such as testing, building, and deployment directly from your repository.

3. \*\*Collaboration Workflow:\*\*

- \*\*Forking:\*\* Contributors typically fork a repository to their account, make changes in their forked repository, and then create a pull request to merge those changes back into the original repository.

- \*\*Code Review:\*\* Pull requests enable code review by allowing team members to comment, suggest changes, and discuss code changes before merging.

- \*\*Continuous Integration (CI) / Continuous Deployment (CD):\*\* GitHub Actions can be used to set up CI/CD pipelines, automating the testing and deployment process.

### Summary:

Version control with Git and collaboration with GitHub are fundamental for managing Python and Django projects efficiently. By leveraging these tools, developers can track changes, collaborate effectively, and maintain code quality throughout the development lifecycle. Whether working solo or in a team, understanding version control and collaboration workflows is crucial for successful project management in Python and Django development.

Activity: Exception handling and ORM Queries for Login Management

Prompt: Exception handling and ORM Queries for Login Management in Python with Django

Exception handling and ORM queries are crucial aspects of implementing login management in Python with Django. Below, I'll provide a basic example of how to handle exceptions and perform ORM queries for login management in a Django project:

1. \*\*Exception Handling\*\*:

Exception handling is essential to gracefully handle errors that may occur during login operations, such as invalid credentials or database errors. Django provides built-in mechanisms for handling exceptions.

```python

from django.contrib.auth import authenticate, login

from django.contrib.auth.models import User

from django.http import HttpResponse

from django.shortcuts import render, redirect

def user\_login(request):

if request.method == 'POST':

username = request.POST.get('username')

password = request.POST.get('password')

user = authenticate(request, username=username, password=password)

if user is not None:

login(request, user)

return redirect('home')

else:

# Handle invalid credentials

return render(request, 'login.html', {'error': 'Invalid username or password'})

else:

return render(request, 'login.html')

```

2. \*\*ORM Queries\*\*:

Django's ORM (Object-Relational Mapping) provides a powerful interface for interacting with the database. You can use ORM queries to retrieve user information from the database for authentication purposes.

```python

from django.contrib.auth.models import User

def user\_login(request):

if request.method == 'POST':

username = request.POST.get('username')

password = request.POST.get('password')

try:

user = User.objects.get(username=username)

if user.check\_password(password):

login(request, user)

return redirect('home')

else:

# Handle invalid password

return render(request, 'login.html', {'error': 'Invalid password'})

except User.DoesNotExist:

# Handle user not found

return render(request, 'login.html', {'error': 'User does not exist'})

else:

return render(request, 'login.html')

```

In this example, we attempt to retrieve the user from the database using `User.objects.get()` and then validate the password using `user.check\_password()`. We use exception handling (`User.DoesNotExist`) to handle cases where the user does not exist in the database.

Ensure that you have a login template (`login.html`) where users can input their credentials and submit the login form. Handle and display any error messages appropriately in the login template.

By combining exception handling and ORM queries, you can implement robust login management in Python with Django, ensuring that your application gracefully handles various scenarios that may arise during the authentication process.

**Day 18**

**Django Email integration**

Integrating email functionality into a Django project is a common requirement, whether it's for user registration, password reset, or sending notifications. Django provides a convenient way to handle email functionality through its built-in `send\_mail()` function and EmailMessage class. Here's how you can integrate email functionality into your Django project:

1. \*\*Configure Email Settings in `settings.py`\*\*:

First, configure the email backend settings in your Django project's `settings.py` file. You can use SMTP for sending emails. Here's an example configuration using Gmail SMTP:

```python

# settings.py

EMAIL\_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'

EMAIL\_HOST = 'smtp.gmail.com'

EMAIL\_PORT = 587

EMAIL\_USE\_TLS = True

EMAIL\_HOST\_USER = 'your\_email@gmail.com'

EMAIL\_HOST\_PASSWORD = 'your\_email\_password'

```

Replace `'your\_email@gmail.com'` and `'your\_email\_password'` with your Gmail email address and password. Alternatively, you can use other SMTP servers or email providers.

2. \*\*Sending Email Using `send\_mail()` Function\*\*:

You can use the `send\_mail()` function provided by Django to send simple emails. Here's an example of sending a basic email:

```python

from django.core.mail import send\_mail

send\_mail(

'Subject here',

'Here is the message.',

'from@example.com',

['to@example.com'],

fail\_silently=False,

)

```

You can call this function from any Django view or task where you want to send an email.

3. \*\*Sending Email Using `EmailMessage` Class\*\*:

For more advanced email functionality, such as sending HTML emails or attaching files, you can use the `EmailMessage` class. Here's an example:

```python

from django.core.mail import EmailMessage

email = EmailMessage(

'Subject here',

'Here is the message.',

'from@example.com',

['to@example.com'],

)

email.content\_subtype = "html" # Set the content type to HTML

email.attach\_file('/path/to/file.pdf') # Attach a file

email.send()

```

4. \*\*Sending Templated Emails\*\*:

You can use Django's template system to create HTML email templates. First, create an HTML template for your email (e.g., `email\_template.html`), then render it using Django's template engine:

```python

from django.core.mail import EmailMessage

from django.template.loader import render\_to\_string

html\_content = render\_to\_string('email\_template.html', {'context\_variable': 'value'})

email = EmailMessage(

'Subject here',

html\_content,

'from@example.com',

['to@example.com'],

)

email.content\_subtype = "html" # Set the content type to HTML

email.send()

```

Ensure that you have created a template file (`email\_template.html`) in your Django templates directory.

5. \*\*Asynchronous Email Sending\*\*:

For long-running tasks or when you want to offload email sending to a background process, you can use Django's `django.core.mail.send\_mail()` method with `asyncio` or `celery`.

Here's a basic example using `asyncio`:

```python

import asyncio

from django.core.mail import send\_mail

async def send\_email\_async():

await asyncio.sleep(1) # Simulate asynchronous operation

send\_mail(

'Subject here',

'Here is the message.',

'from@example.com',

['to@example.com'],

fail\_silently=False,

)

asyncio.run(send\_email\_async())

```

For `celery`, you need to set up a Celery worker and task to handle email sending asynchronously.

With these steps, you can integrate email functionality into your Python Django project, enabling you to send various types of emails, including simple text emails, HTML emails, templated emails, and asynchronously sent emails.

**Day 19**

**Ecom Cart**

Implementing cart management for an e-commerce website using Python and Django involves several steps. Below is a basic guide on how to implement cart functionality:

1. \*\*Create Cart Model\*\*:

First, create a model to represent the cart and its items in your Django application.

|  |
| --- |
| def cartdata(request):  if 'email' in request.session:  prolist = []  b = userregister.objects.get(email = request.session['email'])  a = cart.objects.filter(userid = request.session['userid'],orderid = "0")  print(a,11111111111111111111111111111111111111111)  totalamount = 0  for i in a:  totalamount += int(i.totalprice)  pro = product.objects.get(id = i.productid)  prodict = {'id':i.productid,'proimage':pro.image,'proprice':pro.price,'total':i.totalprice,'userqty':i.quantity}  prolist.append(prodict)  if 'stock' in request.session:  del request.session['stock']  return render(request,'cart.html',{'session':b,'prolist':prolist,'totalamount':totalamount,'stock':"Kale avje!!"})  else:  return render(request,'cart.html',{'session':b,'prolist':prolist,'totalamount':totalamount})  return render(request,'cart.html',{'session':b,'prolist':prolist,'totalamount':totalamount})  else:  return redirect('login') |

2. \*\*Add Cart\*\*:

Associate each user with their cart. You can do this by extending the user model or using a OneToOneField to link users to their carts.

|  |
| --- |
| def additem(request,id):  if 'email' in request.session:  # b = userregister.objects.get(email = request.session['email'])  a = cart.objects.get(userid = request.session['userid'],productid = id,orderid = "0")  b = product.objects.get(id = a.productid)  if b.qty <= 0:  request.session['stock'] = 0  return redirect('cart')  else:  a.quantity = int(a.quantity) + 1  a.totalprice= int(a.totalprice) + int(b.price)  a.save()  b.qty = b.qty - 1  b.save()  print(a,"cart qunatity 11111111111111111111111111111111111")  return redirect('cart')  else:  return redirect('login') |

**Day 20**

**Remove Cart Function**

|  |
| --- |
| def minus(request,id):  if 'email' in request.session:  a = cart.objects.get(userid = request.session['userid'],productid = id,orderid = "0")  b = product.objects.get(id = a.productid)  if int(a.quantity) <= 1:  a.delete()  return redirect('cart')  else:  a.quantity = int(a.quantity) - 1  a.totalprice= int(a.totalprice) - int(b.price)  a.save()  b.qty = int(b.qty) + 1  b.save()  return redirect('cart')  else:  return redirect('login') |

Remove all items makes cart empty, so user have to add again products into the cart. It will also completely remove the data from the model itself.

|  |
| --- |
| def removeall(request):  if 'email' in request.session:  a = cart.objects.filter(userid = request.session['userid'])  for i in a:  b = product.objects.get(id = i.productid)  b.qty = int(b.qty) + int(i.quantity)  b.save()  a.delete()  return redirect('cart')  else:  return redirect('login') |

**Day 21**

**Cart Template**

1. cart.html

|  |
| --- |
| {% extends "nav.html" %}  {% load static %}  {% block abc %}  <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8" />  <meta name="viewport" content="width=device-width, initial-scale=1.0" />  <title>E-commerce</title>  <link  rel="stylesheet"  45  href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"  />  <link  rel="stylesheet"  href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.2/css/all.min.css"  integrity="sha512-z3gLpd7yknf1YoNbCzqRKc4qyor8gaKU1qmn+CShxbuBusANI9QpRohGBreCFkKxLhei6S9CQXFEbbKuqLg0DA=="  crossorigin="anonymous"  referrerpolicy="no-referrer"  />  <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js"></script>  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js"></script>  <link rel="stylesheet" href={% static "css/ecom.css" %} />  </head>  <body>  <!--====================== Navbar ======================-->  <!--====================== Cart Section ======================-->  <div class="container mt-5">  <h1 class="mb-4">Your Shopping Cart</h1>  <div class="row">  <div class="col-md-8">  <table class="table">  <thead>  <tr>  <th>Product</th>  <th>Price</th>  <th>Quantity</th>  <th>Total</th>  <th class="align-bottom pb-2">  <a href={% url "removeall" %}><button class="btn btn-outline-danger">Remove All</button></a>  </th>  </tr>  </thead>  <tbody>  {% for i in prolist %}  <tr class="align-content-center">  <td>  <img src={{i.proimage.url}} alt="" height="200px" width="200px" />  </td>  <td>₹{{i.proprice}}</td>  <td>  <a href={% url "add1" i.id %}><button class="increase items-count btn mr-1" type="button">  <i class="fa fa-plus text-dark"></i>  </button></a>  <button class="border-0 btn">{{i.userqty}}</button>  <a href={% url "minus" i.id %}><button class="increase items-count btn" type="button">  <i class="fa fa-minus text-dark"></i>  </button></a>  </td>  <td>₹{{i.total}}</td>  <td class="align-bottom pb-3">  <button class="btn btn-outline-danger">Remove</button>  </td>  </tr>  {% endfor %}  </tbody>  </table>  </div>  <div class="col-md-3 ml-auto mt-0 pt-0">    <div class="card p-2">  <div class="card-body">  <h5 class="card-title">Summary</h5>  <hr />  <p><strong>Total: </strong>₹{{totalamount}}</p>  <a href={% url "checkout" %}>  <button class="btn btn-outline-success btn-block">  Checkout  </button>  </a>  </div>  </div>  </div>  {% if stock %}  <script>  alert('{{stock}}')  </script>  {% endif %}  </div>  </div>  <!--======================Footer Section======================-->  <footer class="bg-dark text-light py-4">  <div class="container">  <div class="row">  <div class="col-md-4">  <h5>Contact Us</h5>  <p>Email: info@example.com</p>  <p>Phone: 123-456-7890</p>  </div>  <div class="col-md-4">  <h5>Links</h5>  <ul class="list-unstyled">  <li><a href="#">Home</a></li>  <li><a href="#">Products</a></li>  <li><a href="#">About Us</a></li>  <li><a href="#">Contact Us</a></li>  </ul>  </div>  <div class="col-md-4">  <h5>Follow Us</h5>  <ul class="list-inline">  <li class="list-inline-item">  <a href="#"><i class="fab fa-facebook"></i></a>  </li>  <li class="list-inline-item">  <a href="#"><i class="fab fa-twitter"></i></a>  </li>  <li class="list-inline-item">  <a href="#"><i class="fab fa-instagram"></i></a>  </li>  </ul>  </div>  </div>  <hr />  <div class="row">  <div class="col text-center">  <p>&copy; 2023 Your E-Commerce Store</p>  </div>  </div>  </div>  </footer>  </body>  </html>  {% endblock abc %} |

**Day 22**

**Checkout Template**

1. checkout.html

|  |
| --- |
| {% extends 'nav.html' %}  {% block abc %}  <!doctype html>  <html lang="en">  <head>  <meta charset="utf-8">  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">  <meta name="description" content="">  <meta name="author" content="Mark Otto, Jacob Thornton, and Bootstrap contributors">  <meta name="generator" content="Hugo 0.101.0">  <title>Checkout example · Bootstrap v4.6</title>  <link rel="canonical" href="https://getbootstrap.com/docs/4.6/examples/checkout/">    <!-- Bootstrap core CSS -->  <link href="https://getbootstrap.com/docs/4.6/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-xOolHFLEh07PJGoPkLv1IbcEPTNtaed2xpHsD9ESMhqIYd0nLMwNLD69Npy4HI+N" crossorigin="anonymous">  <!-- Favicons -->  <link rel="apple-touch-icon" href="/docs/4.6/assets/img/favicons/apple-touch-icon.png" sizes="180x180">  <link rel="icon" href="/docs/4.6/assets/img/favicons/favicon-32x32.png" sizes="32x32" type="image/png">  <link rel="icon" href="/docs/4.6/assets/img/favicons/favicon-16x16.png" sizes="16x16" type="image/png">  <link rel="manifest" href="/docs/4.6/assets/img/favicons/manifest.json">  <link rel="mask-icon" href="/docs/4.6/assets/img/favicons/safari-pinned-tab.svg" color="#563d7c">  <link rel="icon" href="/docs/4.6/assets/img/favicons/favicon.ico">  <meta name="msapplication-config" content="/docs/4.6/assets/img/favicons/browserconfig.xml">  <meta name="theme-color" content="#563d7c">  <style>  .bd-placeholder-img {  font-size: 1.125rem;  text-anchor: middle;  -webkit-user-select: none;  -moz-user-select: none;  -ms-user-select: none;  user-select: none;  }  @media (min-width: 768px) {  .bd-placeholder-img-lg {  font-size: 3.5rem;  }  }  .reveal-if-active {  opacity: 0;  max-height: 0;  overflow: hidden;  }  input[type="radio"]:checked ~ .reveal-if-active{  opacity: 1;  max-height: 100px; /\* little bit of a magic number :( \*/  overflow: visible;  }    </style>    <!-- Custom styles for this template -->  <link href="form-validation.css" rel="stylesheet">  </head>  <body class="bg-light">    <div class="container">  <div class="py-5 text-center">  {% comment %} <img class="d-block mx-auto mb-4" src="https://getbootstrap.com/docs/4.6/assets/brand/bootstrap-solid.svg" alt="" width="72" height="72"> {% endcomment %}  <h2>Checkout form</h2>  {% comment %} <p class="lead">Below is an example form built entirely with Bootstrap’s form controls. Each required form group has a validation state that can be triggered by attempting to submit the form without completing it.</p> {% endcomment %}  </div>  <div class="row">  <div class="col-md-4 order-md-2 mb-4">  <h4 class="d-flex justify-content-between align-items-center mb-3">  <span class="text-muted">Your cart</span>  <span class="badge badge-secondary badge-pill">3</span>  </h4>  {% for i in prolist %}  <ul class="list-group mb-3">  <li class="list-group-item d-flex justify-content-between lh-condensed">  <div>  <div>  <img src="{{i.proimg.url}}" alt="" style="height: 25%; width: 25%;">  </div>  <h6 class="my-0">{{i.proname}}</h6>  <small class="text-muted">Discrption:{{i.prodis}}</small>  <div>  <span class="text-muted">Price:{{i.proprice}}</span>  </div>  <span class="text-muted">Quantity:{{i.cartqty}}</span>  <span class="text-muted">price of your qty:{{i.prototalprice}}</span>  </div>  </li>  {% endfor %}  <li class="list-group-item d-flex justify-content-between">  <span>Total </span>  <strong>{{grandtotal}}</strong>  </li>    </ul>    <form class="card p-2">  <div class="input-group">  {% comment %} <input type="text" class="form-control" placeholder="Promo code">  <div class="input-group-append">  <button type="submit" class="btn btn-secondary">Redeem</button>  </div> {% endcomment %}  </div>  </form>  </div>  <div class="col-md-8 order-md-1">  <h4 class="mb-3">Billing address</h4>  <form class="needs-validation" novalidate method="post">  {% csrf\_token %}  <div class="row">  <div class="col-md-6 mb-3">  <label for="firstName">First name</label>  <input type="text" class="form-control" id="firstName" placeholder="" value="{{session.name}}" required name="name">  <div class="invalid-feedback">  Valid first name is required.  </div>  </div>  </div>  <div class="mb-3">  <label for="email">Email <span class="text-muted"></span></label>  <input type="email" class="form-control" id="email" placeholder="you@example.com" name='email' required value={{session.email}} readonly>  <div class="invalid-feedback">  Please enter a valid email address for shipping updates.  </div>  </div>  <div class="mb-3">  <label for="email">Mobile Number <span class="text-muted"></span></label>  <input type="text" class="form-control" id="email" placeholder="+91 " name='mob' required value={{session.mob}} >  <div class="invalid-feedback">  Please enter a valid email address for shipping updates.  </div>  </div>  <div class="mb-3">  <label for="address">Address</label>  <input type="text" class="form-control" id="address" placeholder="1234 Main St" required name='add' value="{{session.add}}" >  <div class="invalid-feedback">  Please enter your shipping address.  </div>  </div>  <div class="mb-3">  <label for="address2">City <span class="text-muted"></span></label>  <input type="text" class="form-control" id="address2" placeholder="city" name='city' required value="">  </div>  <div class="row">    <div class="col-md-4 mb-3">  <label for="state">State</label>  <input type="text" class="form-control" id="address2" placeholder="state" name='state' required value="{{a.state}}">  </div>  <div class="col-md-3 mb-3">  <label for="zip">Zip</label>  <input type="text" class="form-control" id="zip" placeholder="Pincode" required name='pin' value="{{a.pincode}}">  <div class="invalid-feedback">  Zip code required.  </div>  </div>  </div>  <hr class="mb-4">  <div class="custom-control custom-checkbox">  <input type="checkbox" class="custom-control-input" id="same-address">  <label class="custom-control-label" for="same-address">Shipping address is the same as my billing address</label>  </div>  <div class="custom-control custom-checkbox">  <input type="checkbox" class="custom-control-input" id="save-info">  <label class="custom-control-label" for="save-info">Save this information for next time</label>  </div>  <hr class="mb-4">  <h4 class="mb-3">Payment</h4>  <div class="d-block my-3">  <div class="custom-control custom-radio">  <input id="online" name="paymentvia" type="radio" class="custom-control-input" value="online">  <label class="custom-control-label" for="online">Online</label>  </div>  <div class="custom-control custom-radio">  <input id="cod" name="paymentvia" type="radio" class="custom-control-input" value="cod">  <label class="custom-control-label" for="cod">Cash on Delivery</label>  </div>  </div>  <hr class="mb-4">  <button class="btn btn-primary btn-lg btn-block" type="submit">Continue to checkout</button>  {% comment %} <button class="btn btn-primary btn-lg btn-block" type="submit" id="pay-btn">Make Payment</button> {% endcomment %}  </form>  </div>  </div>  <footer class="my-5 pt-5 text-muted text-center text-small">  <p class="mb-1">&copy; 2017-2022 Company Name</p>  <ul class="list-inline">  <li class="list-inline-item"><a href="#">Privacy</a></li>  <li class="list-inline-item"><a href="#">Terms</a></li>  <li class="list-inline-item"><a href="#">Support</a></li>  </ul>  </footer>  </div>  <script src="https://cdn.jsdelivr.net/npm/jquery@3.5.1/dist/jquery.slim.min.js" integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj" crossorigin="anonymous"></script>  <script>window.jQuery || document.write('<script src="/docs/4.6/assets/js/vendor/jquery.slim.min.js"><\/script>')</script><script src="/docs/4.6/dist/js/bootstrap.bundle.min.js" integrity="sha384-Fy6S3B9q64WdZWQUiU+q4/2Lc9npb8tCaSX9FK7E8HnRr0Jz8D6OP9dO5Vg3Q9ct" crossorigin="anonymous"></script>    </body>  </html>  {% endblock abc %} |

**Day 23**

**Product Template**

1. product.html

|  |
| --- |
| {% extends "nav.html" %}  {% block abc %}  {% load static %}  <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8" />  <meta name="viewport" content="width=device-width, initial-scale=1.0" />  <title>E-commerce</title>  <link  rel="stylesheet"  45  href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"  />  <link  rel="stylesheet"  href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.2/css/all.min.css"  integrity="sha512-z3gLpd7yknf1YoNbCzqRKc4qyor8gaKU1qmn+CShxbuBusANI9QpRohGBreCFkKxLhei6S9CQXFEbbKuqLg0DA=="  crossorigin="anonymous"  referrerpolicy="no-referrer"  />  <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js"></script>  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js"></script>  <link rel="stylesheet" href={% static "css/ecom.css" %} />  </head>  <body>  <!--====================== Navbar ======================-->  <!--====================== Main Section ======================-->  <div class="container my-5">  <div class="row mt-4 mr-auto ml-auto">  <div class="col-lg-4">    <img  src={{prodetails.image.url}}  alt="Product Image"  class="img-fluid"  height="500px"  width="300px"  />    </div>  <div class="col-lg-6 mr-auto ml-auto mt-5">  <h1>{{prodetails.name}}</h1>  <p>  {{prodetails.discrption}}  </p>  <p><strong>Price:</strong> ₹{{prodetails.price}}</p>  <form action="" method="post">  {% csrf\_token %}  <input type="number" minlength="1" name="qty">  <i class="fa-solid fa-star ratingcolor"></i>  <i class="fa-solid fa-star ratingcolor"></i>  <i class="fa-solid fa-star ratingcolor"></i>  <i class="fa-regular fa-star-half-stroke ratingcolor"></i>  <i class="fa-regular fa-star ratingcolor"></i>  <br />  <a href="cart.html">  <button class="btn btn-outline-primary mt-3">Add to Cart</button>  </a>  </form>  {% if chej %}  <script>  alert('{{chej}}')  </script>  {% endif %}  </div>  </div>  </div>  <!--======================Footer Section======================-->  <footer class="bg-dark text-light py-4">  <div class="container">  <div class="row">  <div class="col-md-4">  <h5>Contact Us</h5>  <p>Email: info@example.com</p>  <p>Phone: 123-456-7890</p>  </div>  <div class="col-md-4">  <h5>Links</h5>  <ul class="list-unstyled">  <li><a href="#">Home</a></li>  <li><a href="#">Products</a></li>  <li><a href="#">About Us</a></li>  <li><a href="#">Contact Us</a></li>  </ul>  </div>  <div class="col-md-4">  <h5>Follow Us</h5>  <ul class="list-inline">  <li class="list-inline-item">  <a href="#"><i class="fab fa-facebook"></i></a>  </li>  <li class="list-inline-item">  <a href="#"><i class="fab fa-twitter"></i></a>  </li>  <li class="list-inline-item">  <a href="#"><i class="fab fa-instagram"></i></a>  </li>  </ul>  </div>  </div>  <hr />  <div class="row">  <div class="col text-center">  <p>&copy; 2023 Your E-Commerce Store</p>  </div>  </div>  </div>  </footer>  </body>  </html>  {% endblock abc %} |

**Day 24**

**“Deployment”**

**Prompt: Website Deployment on Server**

Deploying a Django project typically involves the following steps:

1. \*\*Prepare your Django project for deployment:\*\*

- Ensure your project is properly configured for production, including settings related to database, static files, media files, security, and debugging.

- Update your `ALLOWED\_HOSTS` setting in `settings.py` to include the domain names or IP addresses of your production server(s).

2. \*\*Choose a hosting provider:\*\*

- Select a hosting provider that supports Django applications. Popular options include Heroku, DigitalOcean, AWS (Amazon Web Services), Google Cloud Platform, PythonAnywhere, and others.

- Consider factors such as pricing, scalability, performance, ease of use, and support when choosing a hosting provider.

3. \*\*Set up your server environment:\*\*

- Provision a server instance (virtual private server or VPS) with your hosting provider.

- Install necessary software on the server, such as Python, a web server (e.g., Nginx or Apache), and a database server (e.g., PostgreSQL, MySQL, or SQLite).

- Set up a domain name and configure DNS settings to point to your server's IP address.

4. \*\*Deploy your code to the server:\*\*

- Upload your Django project files to the server using SSH, FTP, or a version control system like Git.

- Set up a virtual environment on the server and install project dependencies using pip.

- Configure the server to serve static files and media files (if applicable) using the appropriate web server configuration.

5. \*\*Configure the web server:\*\*

- Set up Nginx or Apache to serve as a reverse proxy for your Django application.

- Configure the web server to pass requests to the Django application using WSGI (Web Server Gateway Interface) or ASGI (Asynchronous Server Gateway Interface).

6. \*\*Set up database and environment variables:\*\*

- Create a database for your Django project and configure the database connection settings in your `settings.py` file.

- Set up environment variables for sensitive information such as database credentials, secret keys, and API keys.

7. \*\*Collect static files:\*\*

- Run the `collectstatic` management command to collect static files from your Django apps into one location.

- Configure your web server to serve static files directly or use a content delivery network (CDN) for improved performance.

8. \*\*Test your deployment:\*\*

- Access your Django application through the domain name or IP address of your server.

- Test all functionality to ensure everything is working correctly in the production environment.

- Monitor server logs and error messages for any issues that may arise.

9. \*\*Implement security measures:\*\*

- Enable HTTPS to encrypt traffic between the client and server.

- Implement security best practices, such as using strong passwords, keeping software up-to-date, and configuring firewalls.

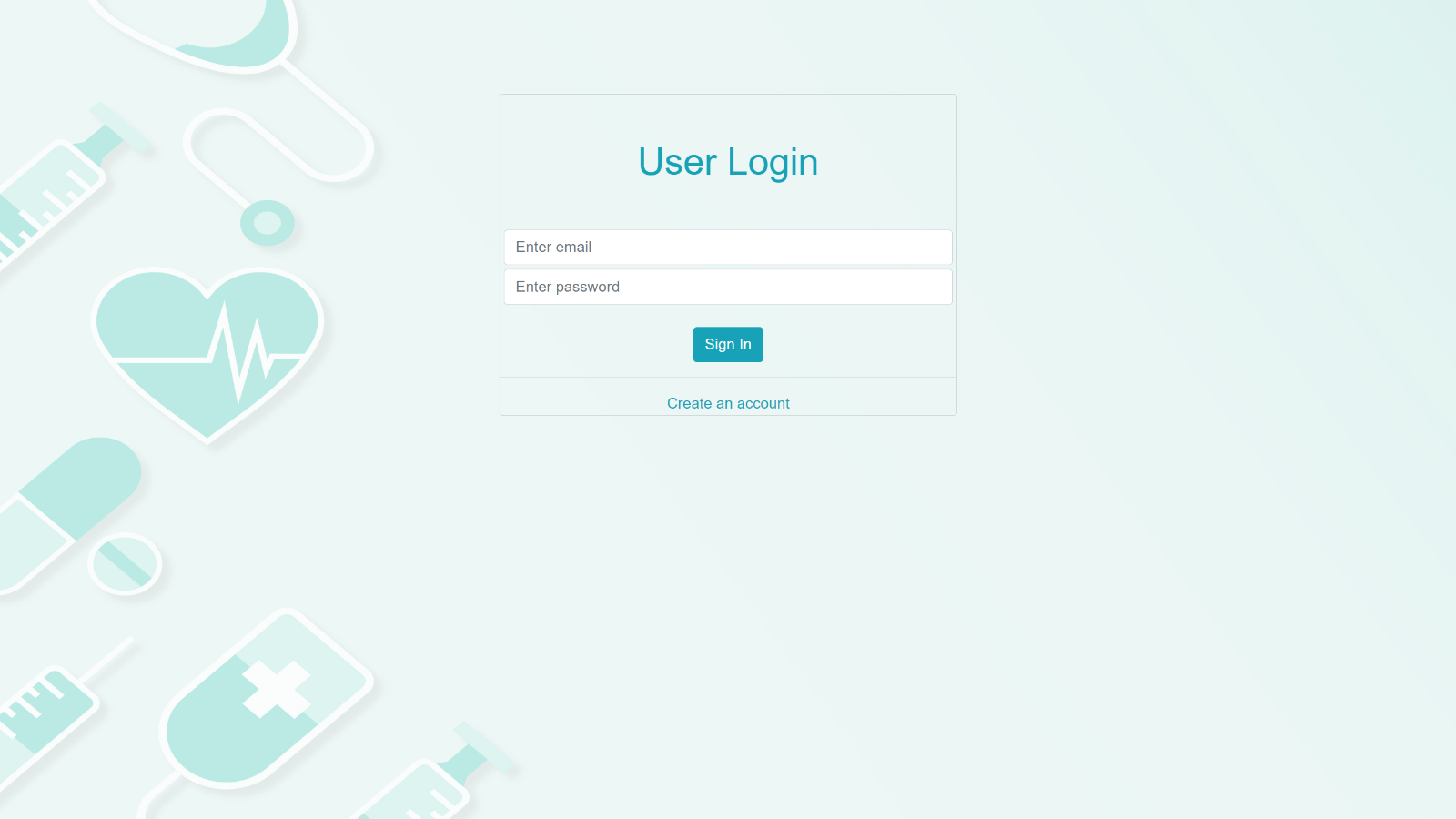
10. \*\*Set up backups and monitoring:\*\*

- Configure regular backups of your Django project files, database, and server configuration.

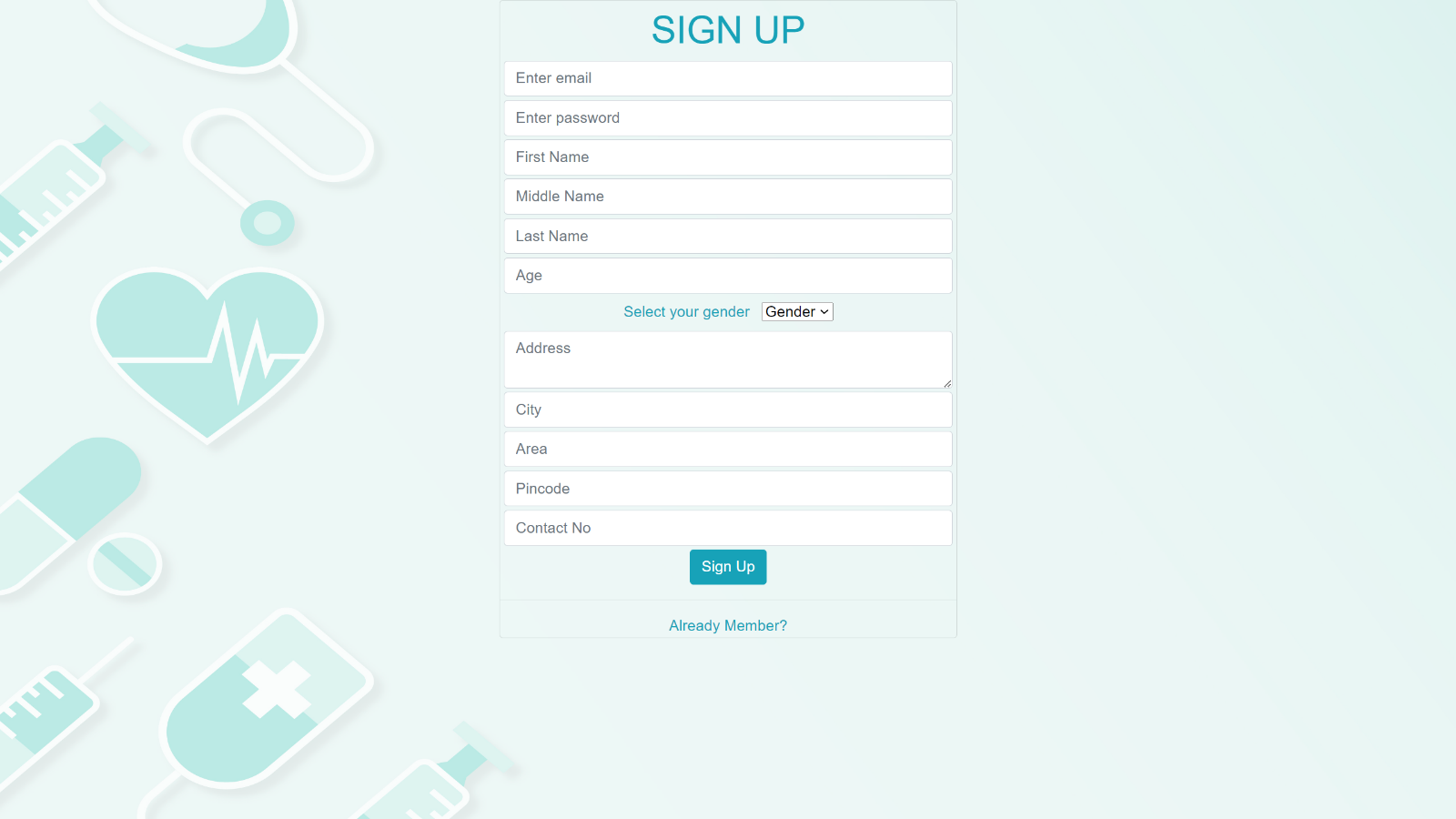
- Set up monitoring tools to track server performance, uptime, and security threats.

Remember to refer to the documentation of your hosting provider and follow their specific guidelines for deploying Django applications. Additionally, consider automating the deployment process using tools like Fabric, Ansible, or Docker for easier management and scalability.

**Day 25**

****

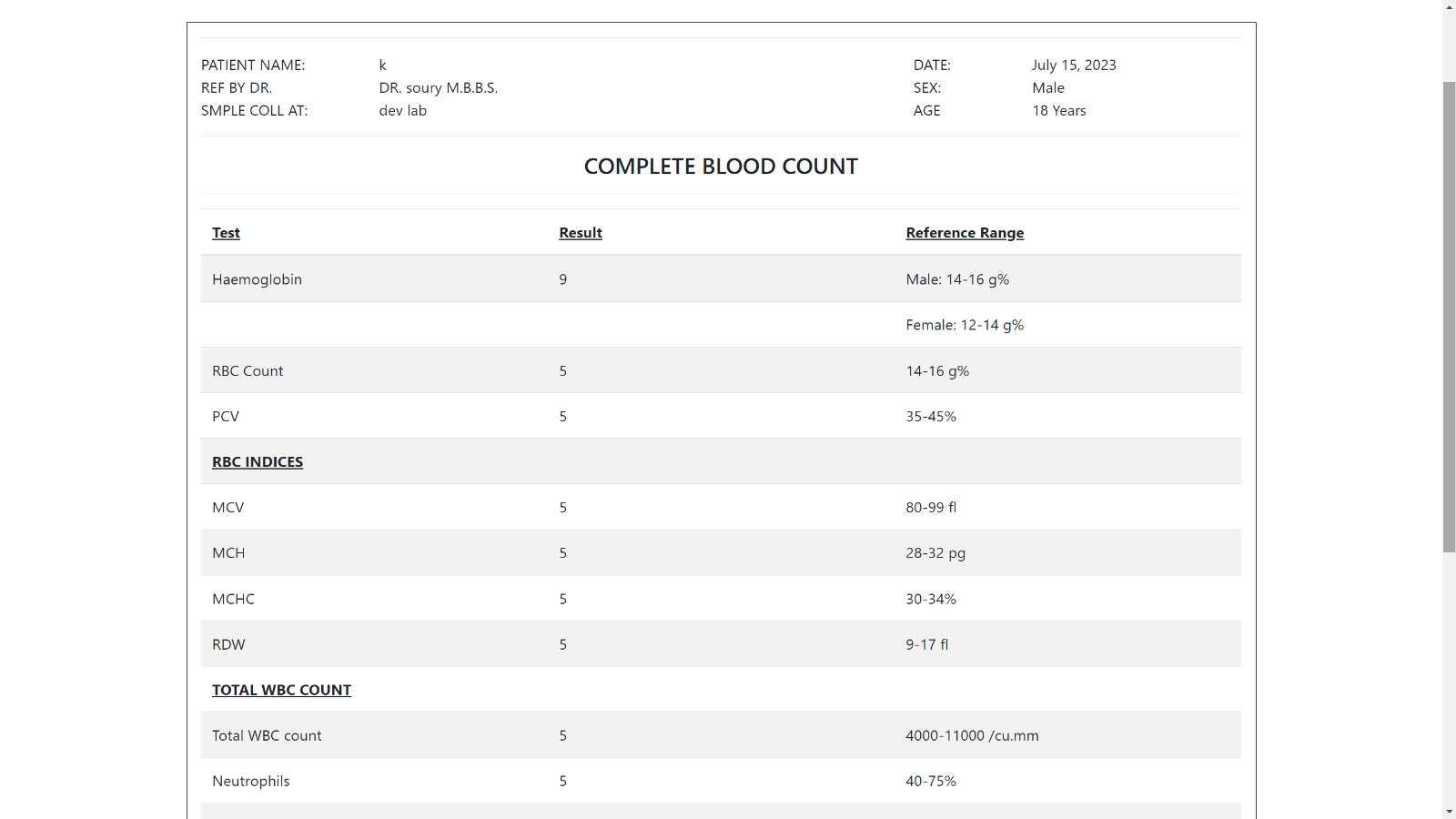
**Day 26**

****

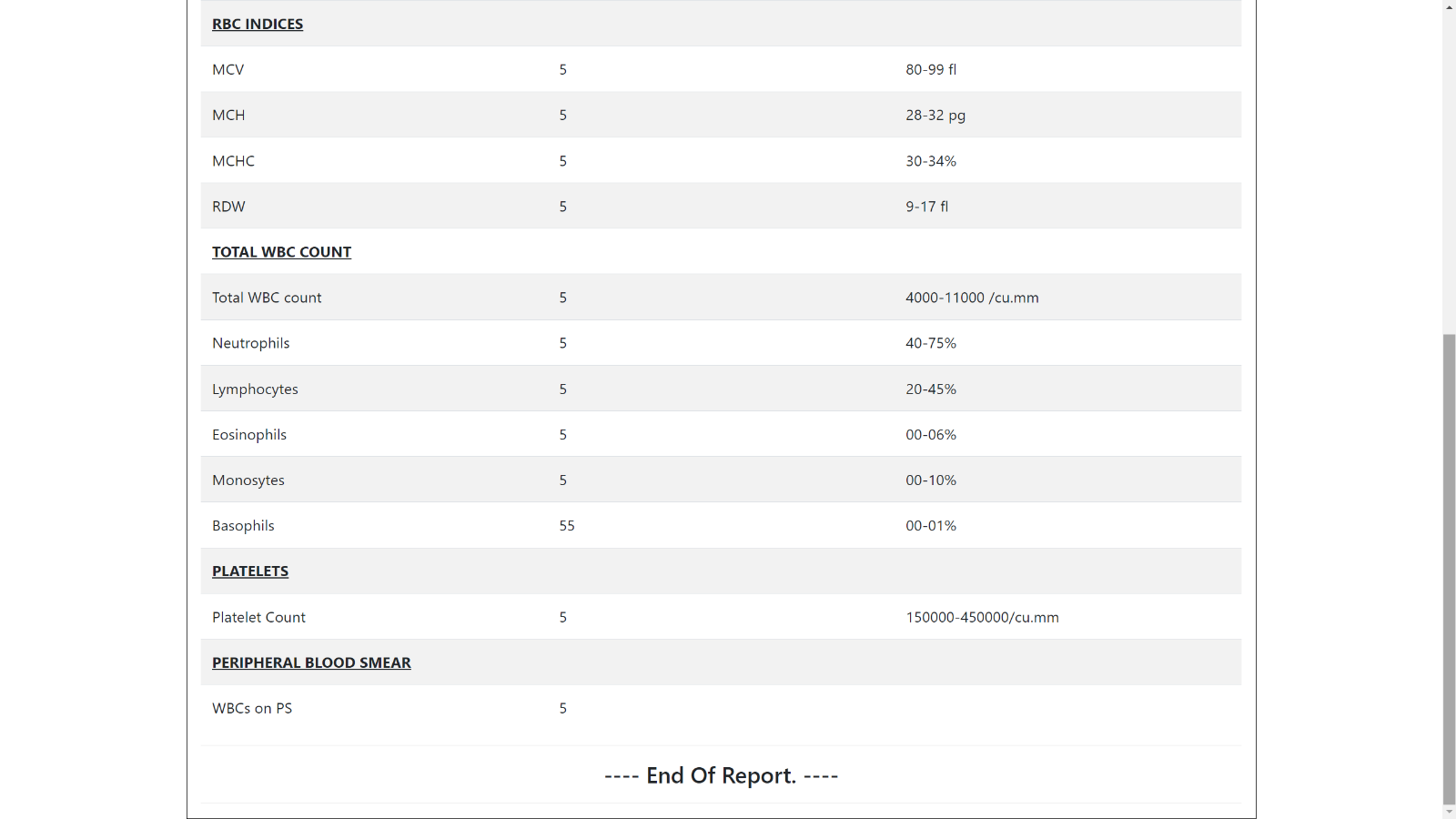
**Day 27**

****

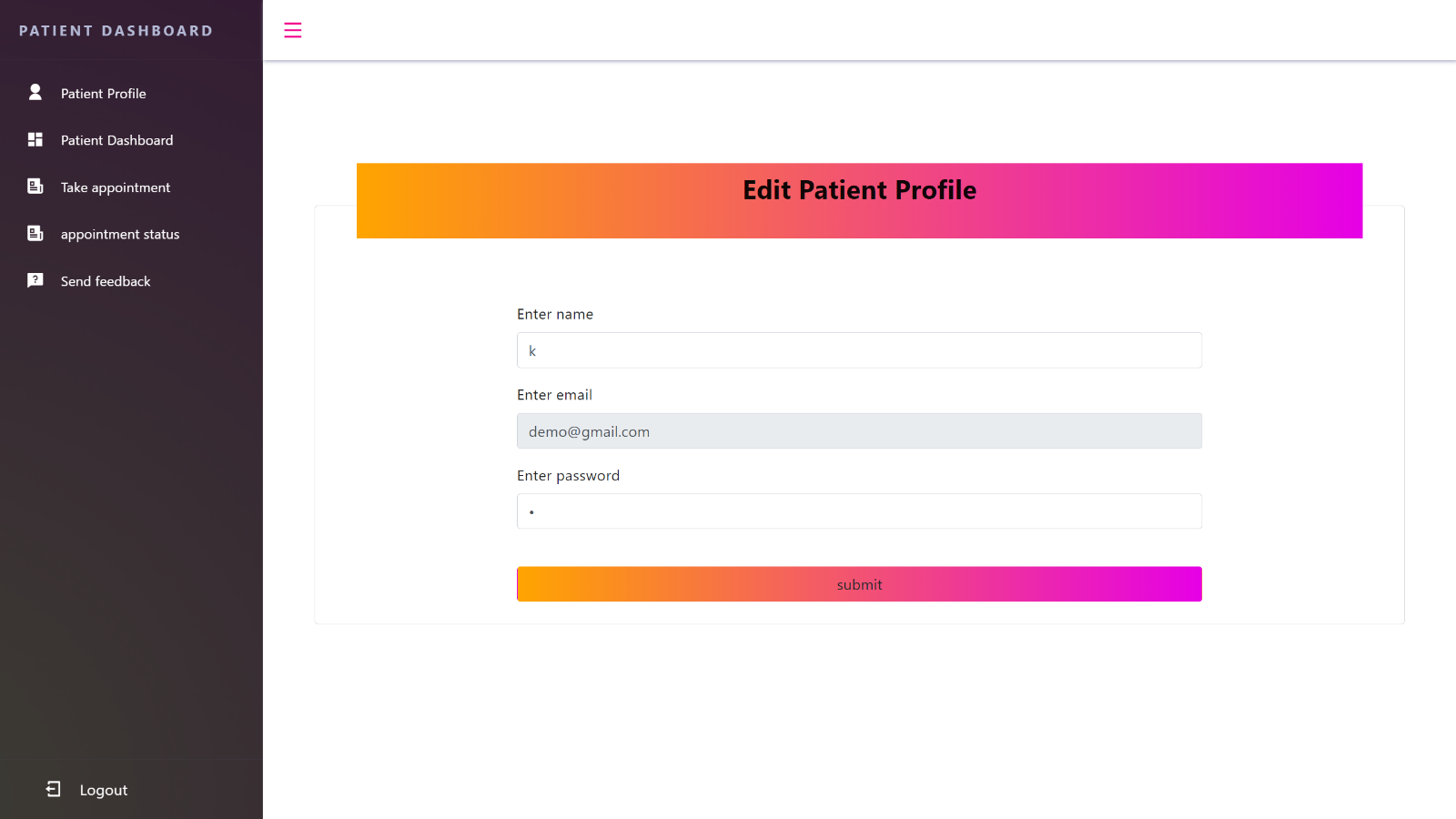
**Day 28**

****

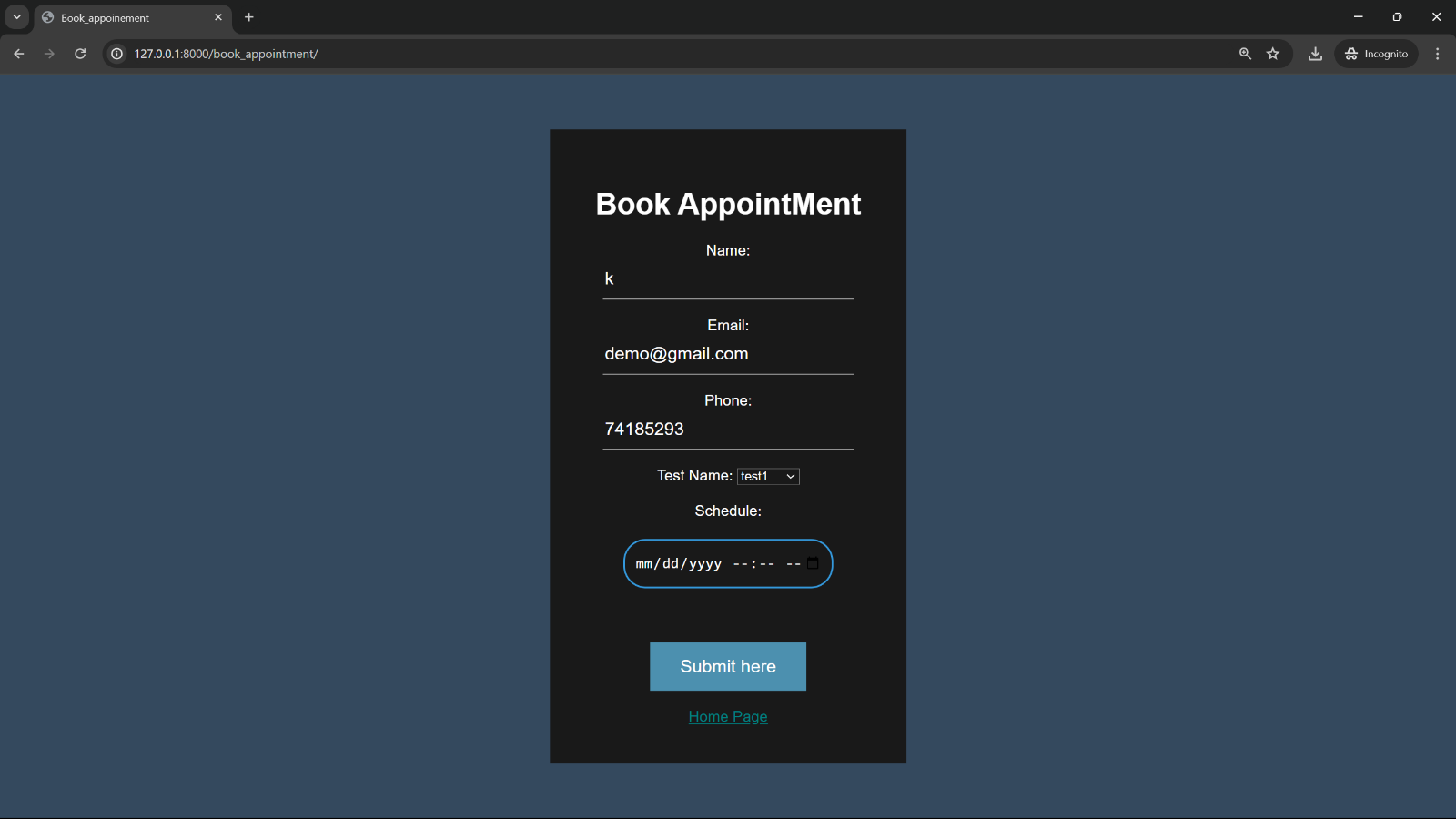
**Day 29**

****

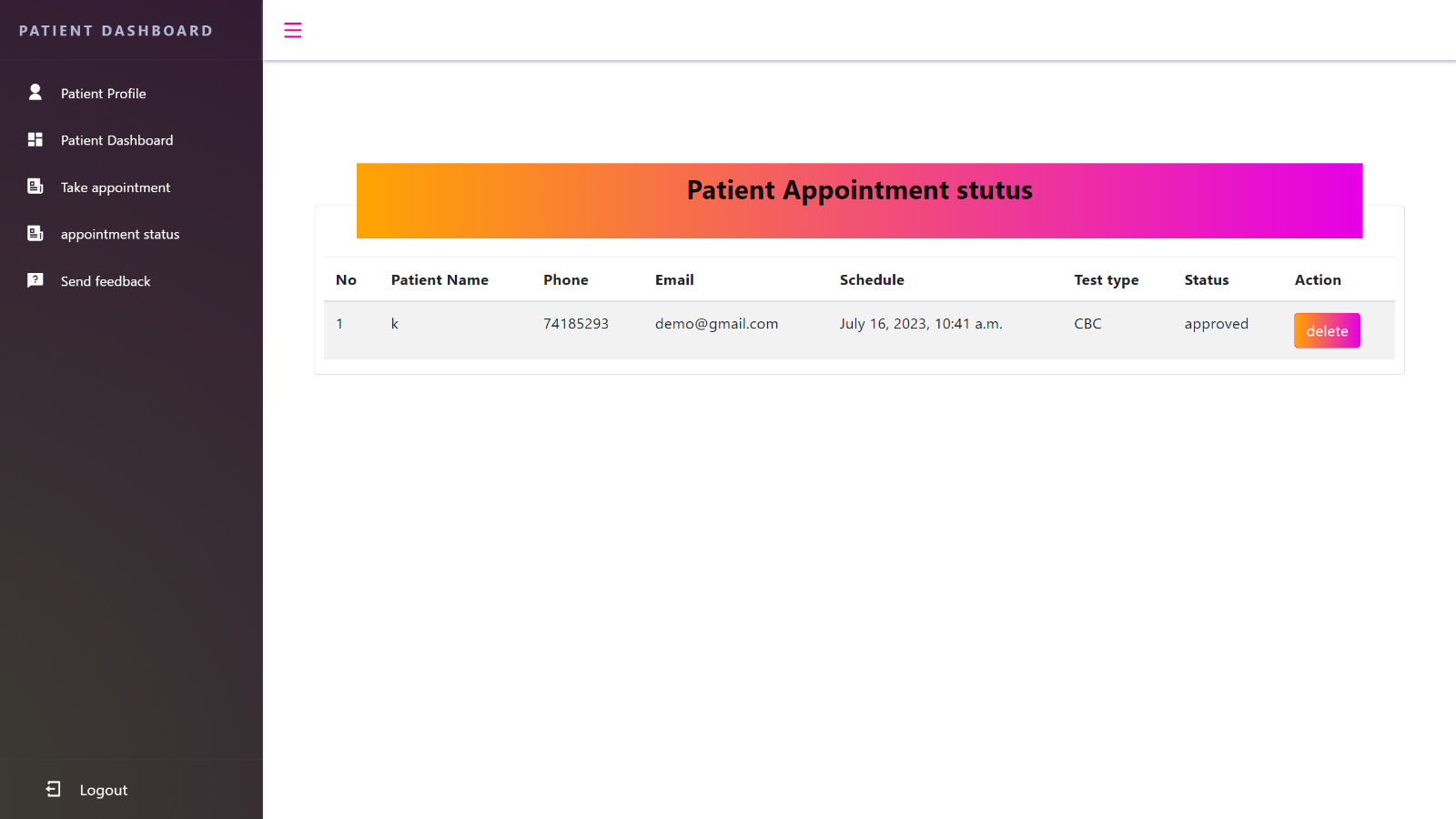
**Day 30**

****

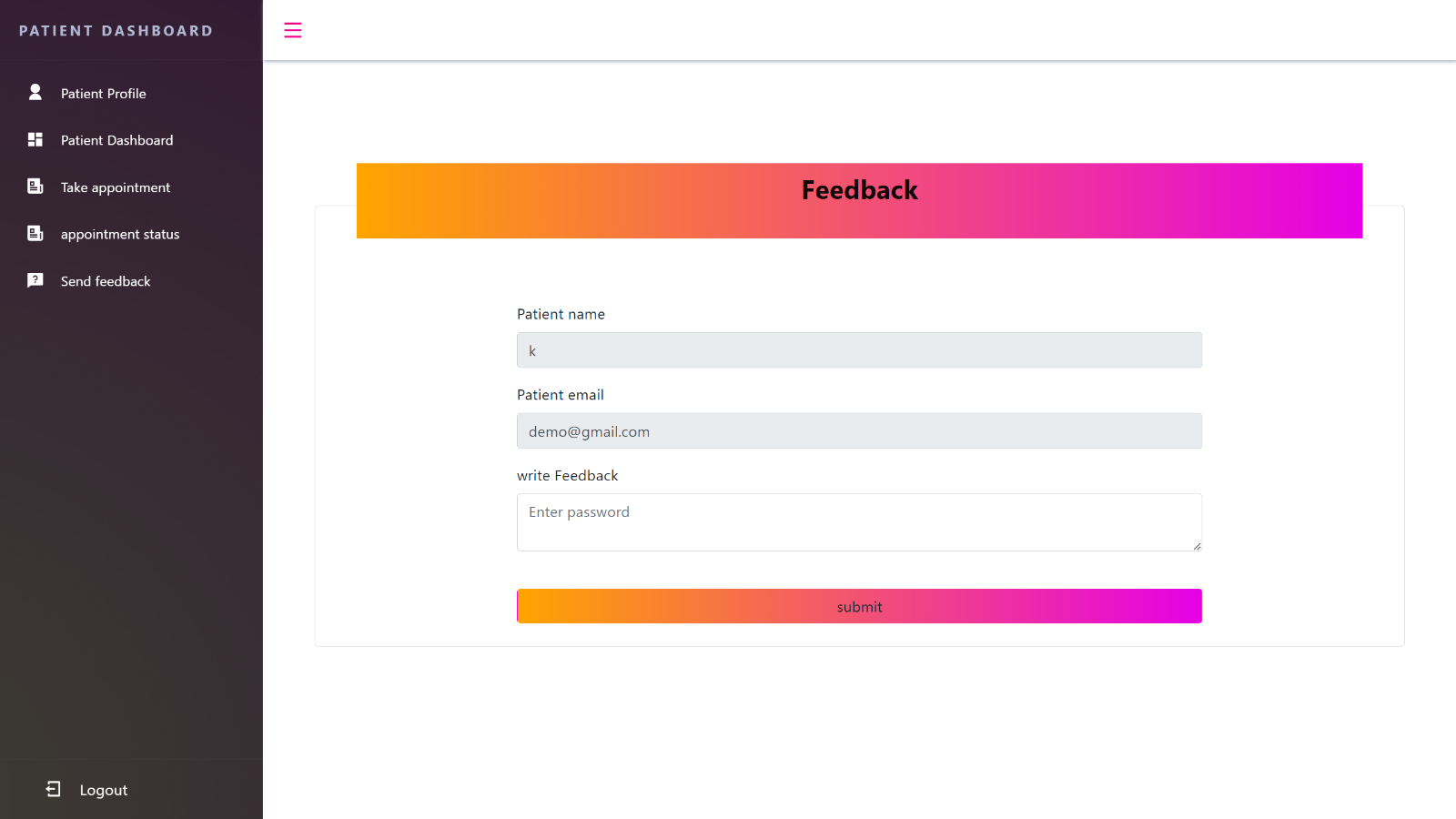
**Day 31**

****

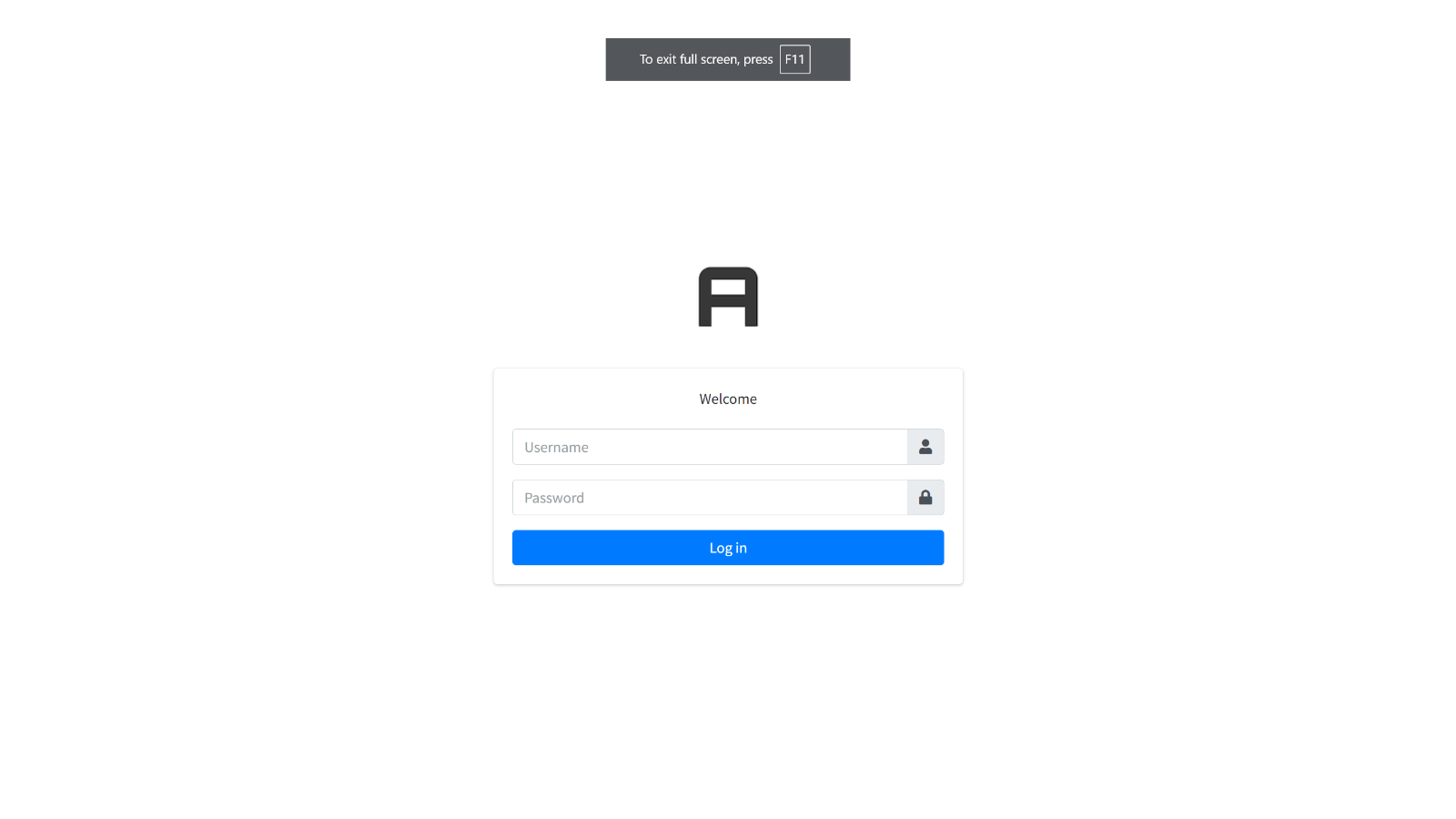
**Day 32**

****

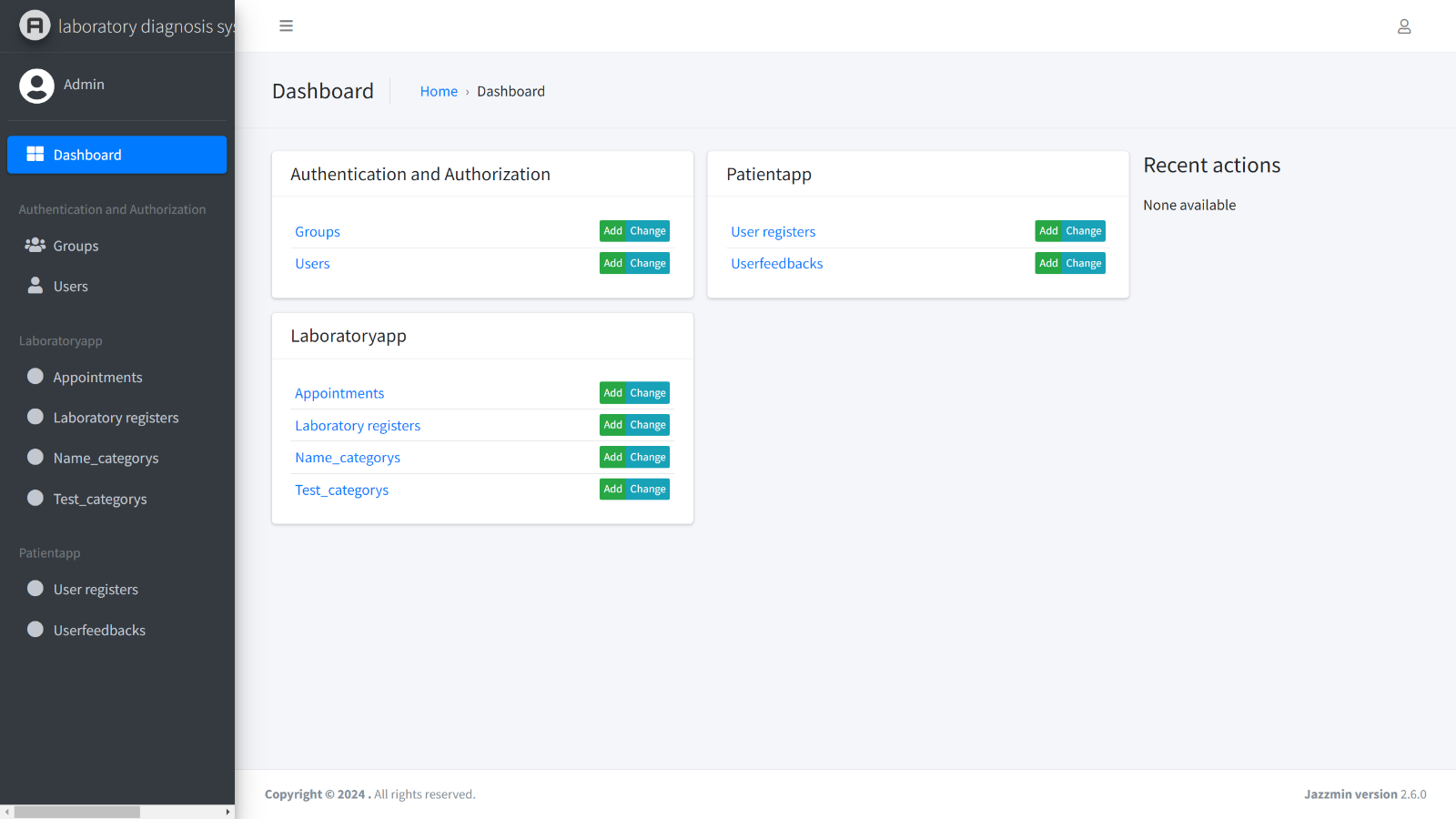
**Day 33**

****

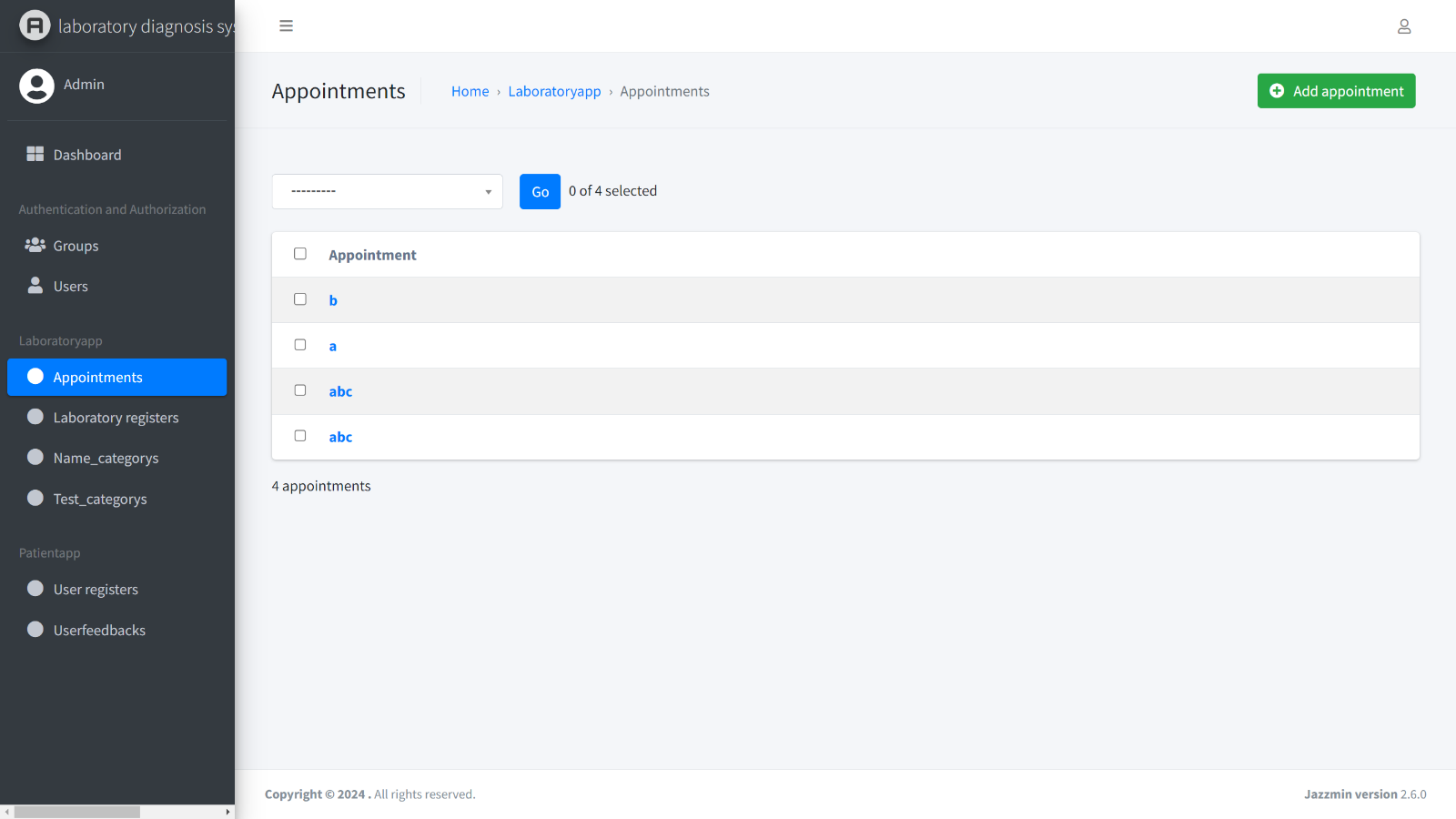
**Day 34**

****

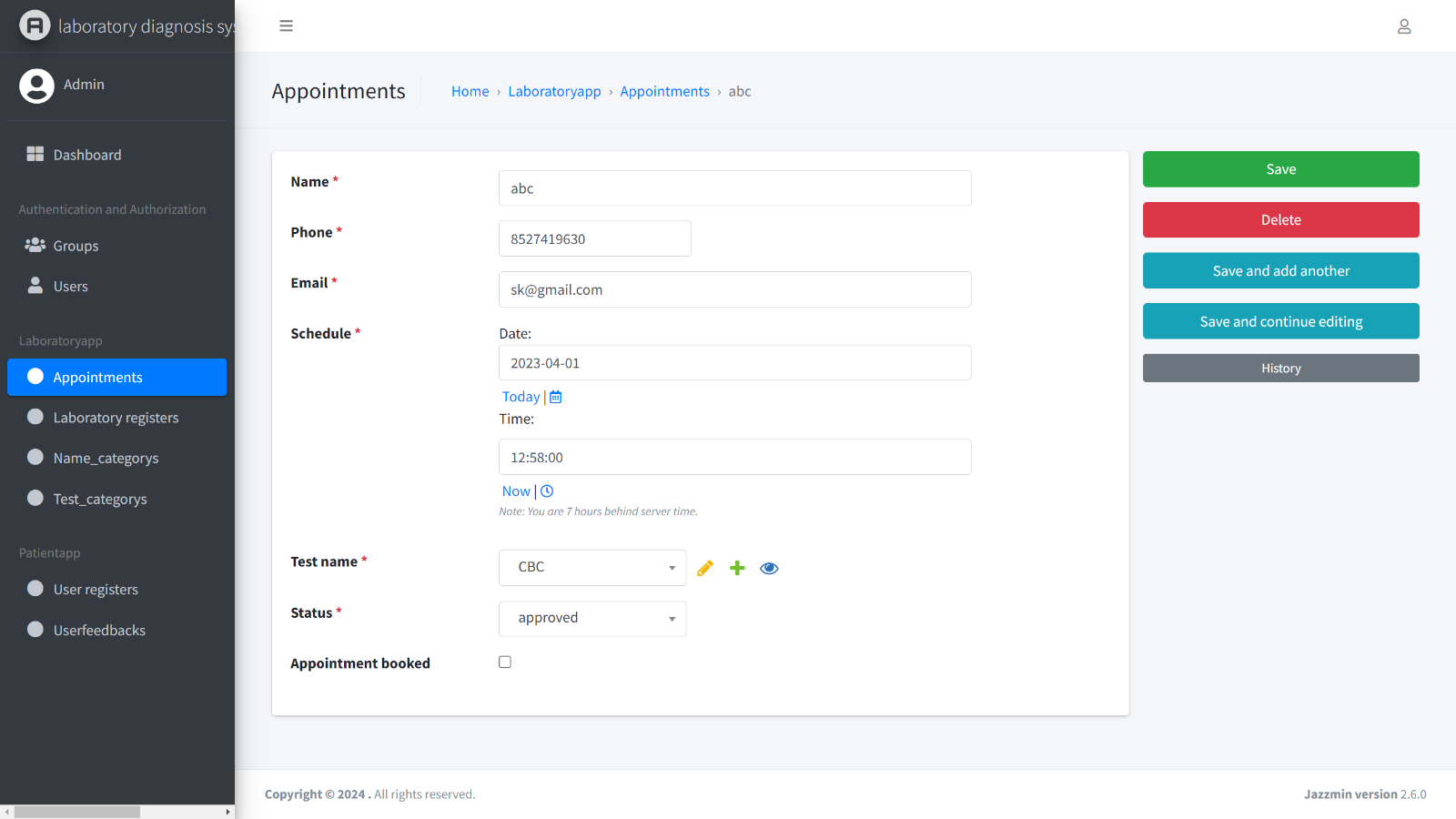
**Day 35**

****

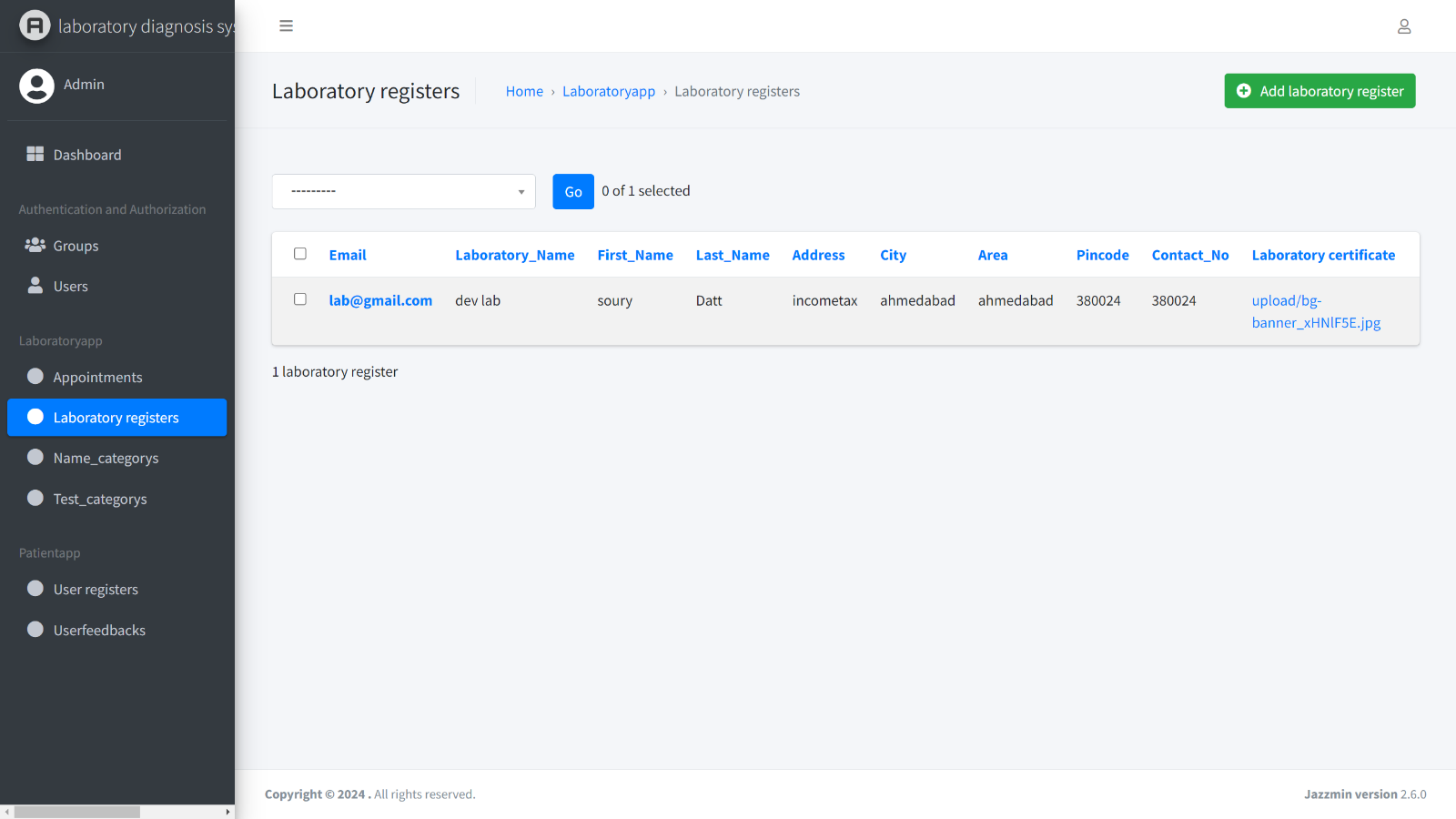
**Day 36**

****

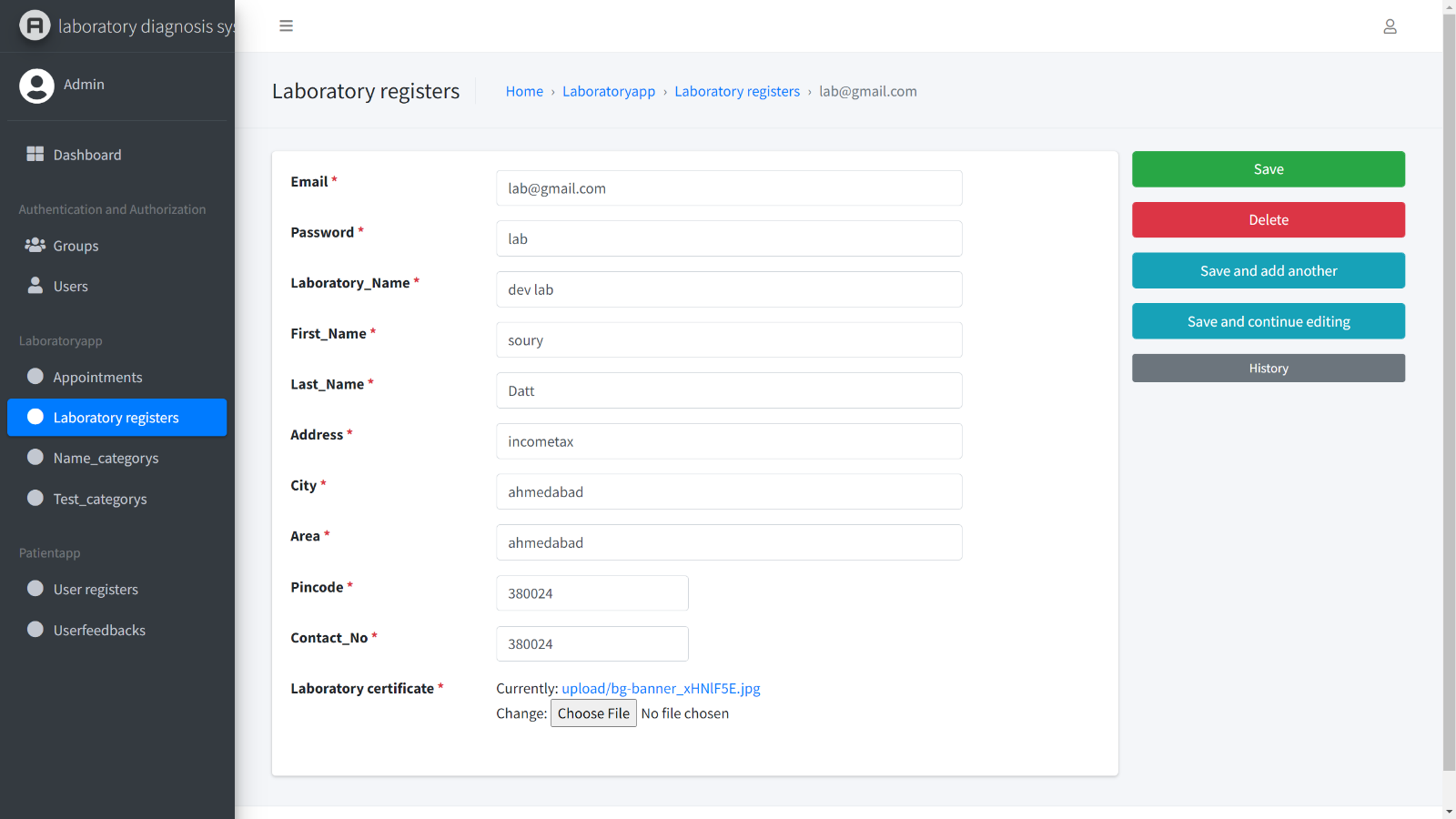
**Day 37**

****

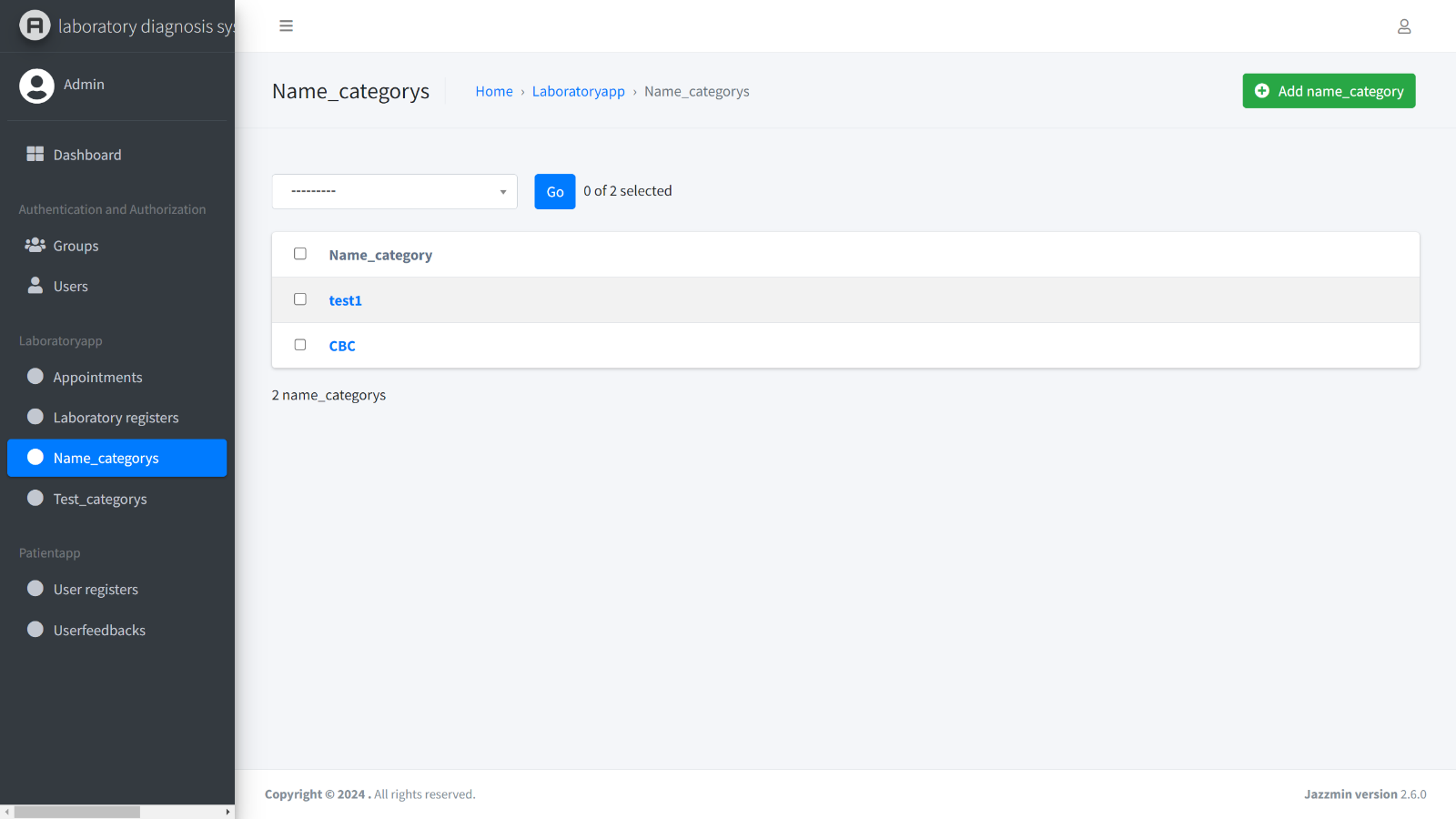
**Day 38**

****

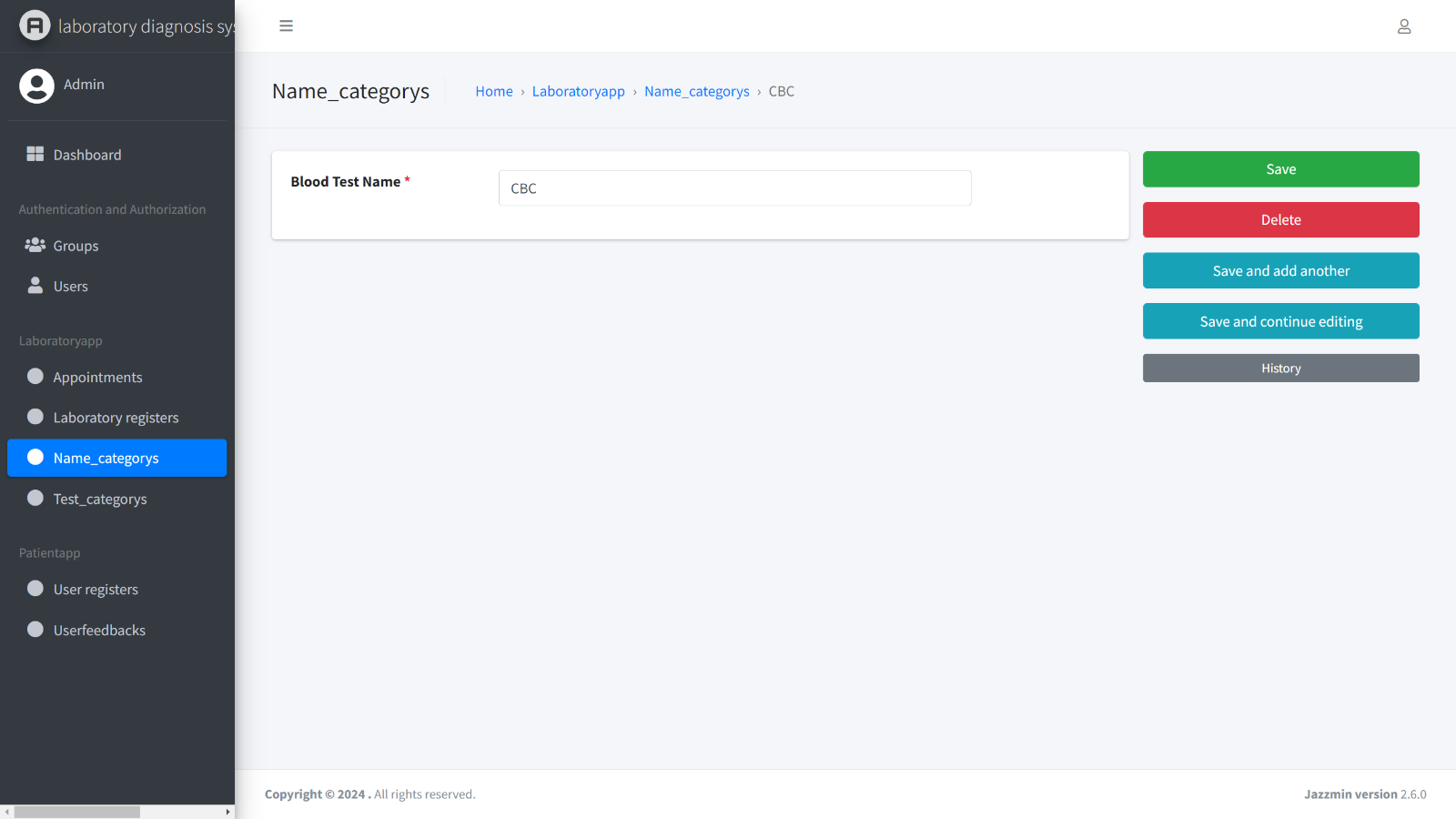
**Day 39**

****

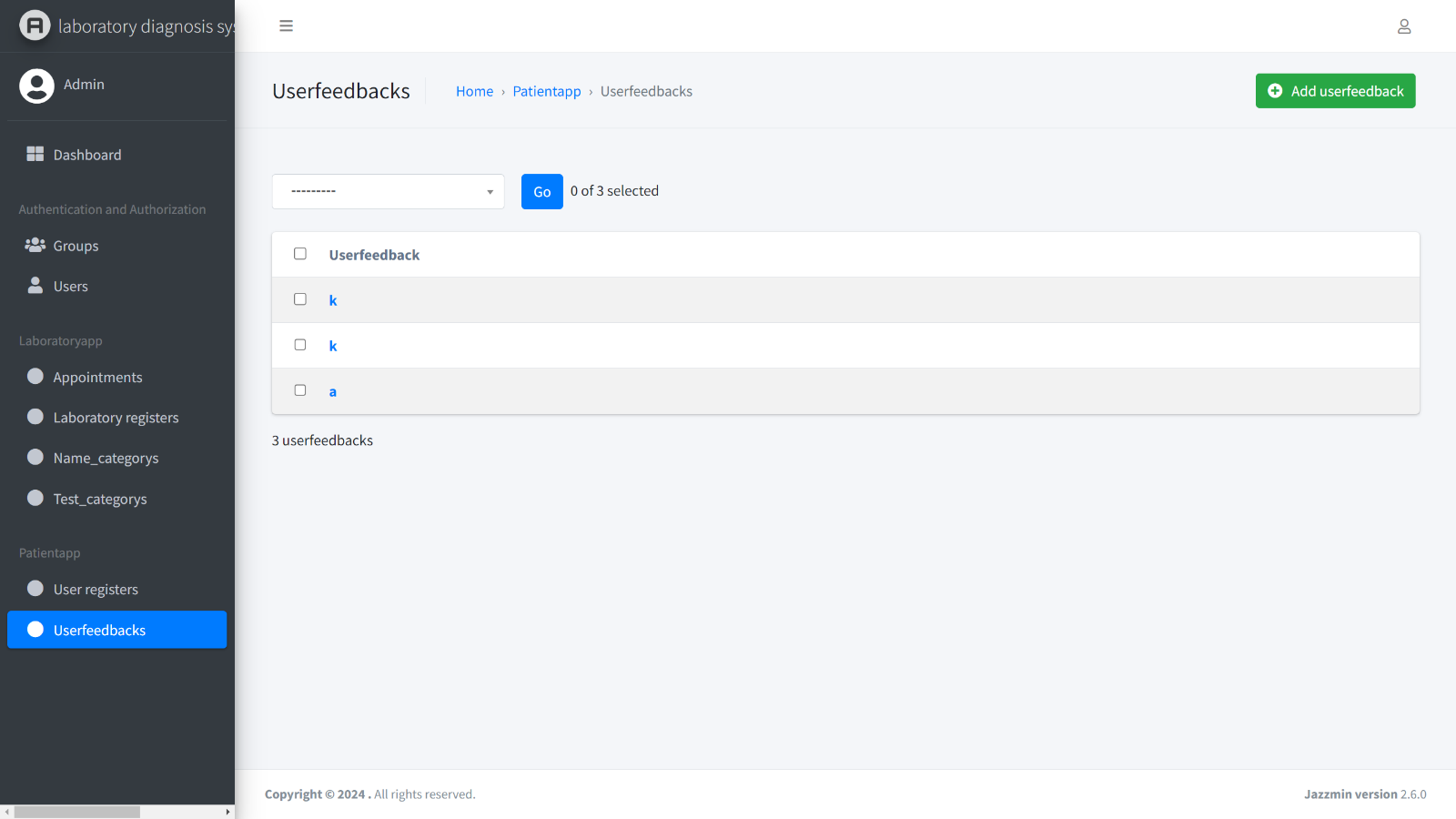
**Day 40**

****

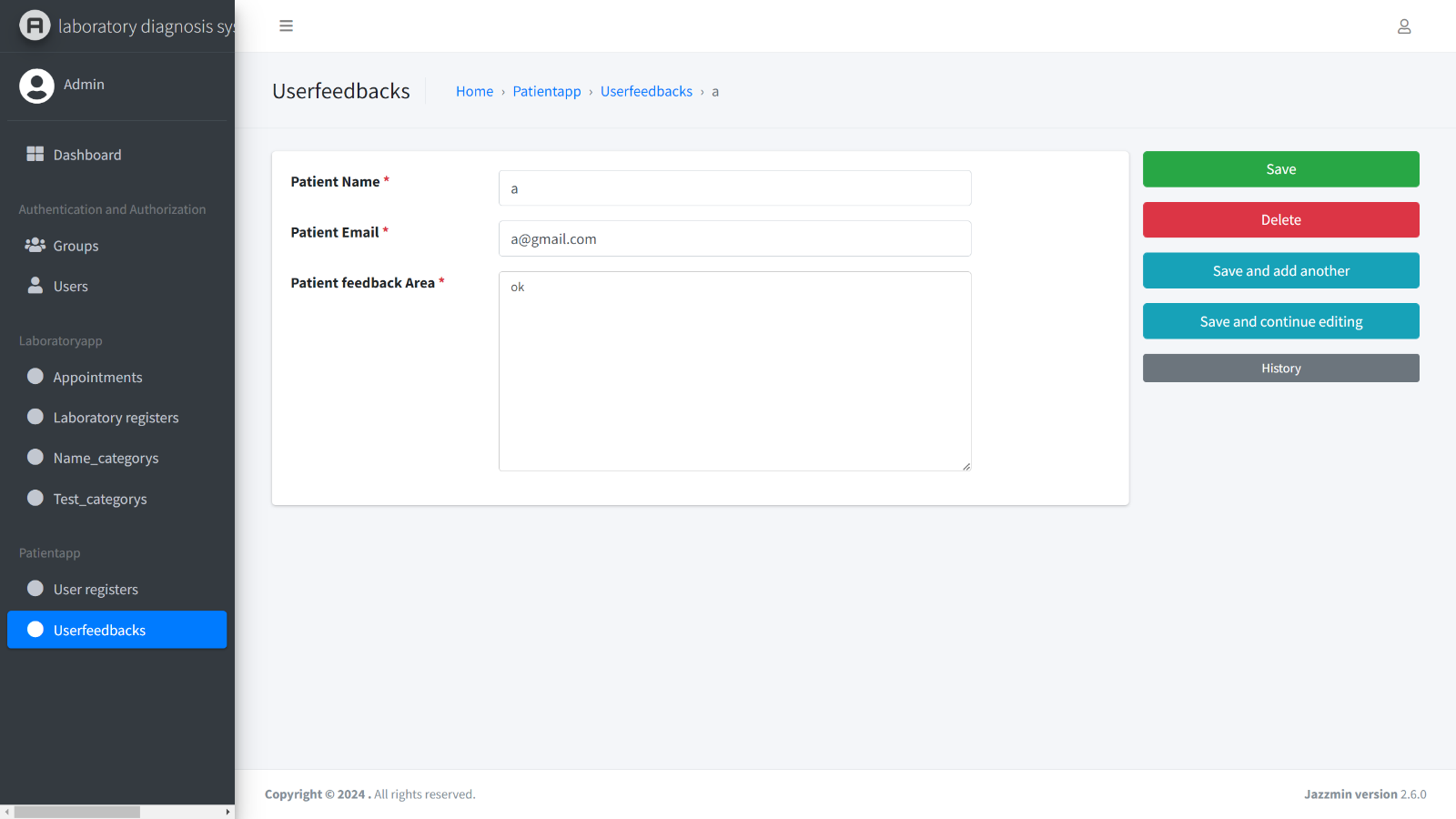
**Day 41**

****

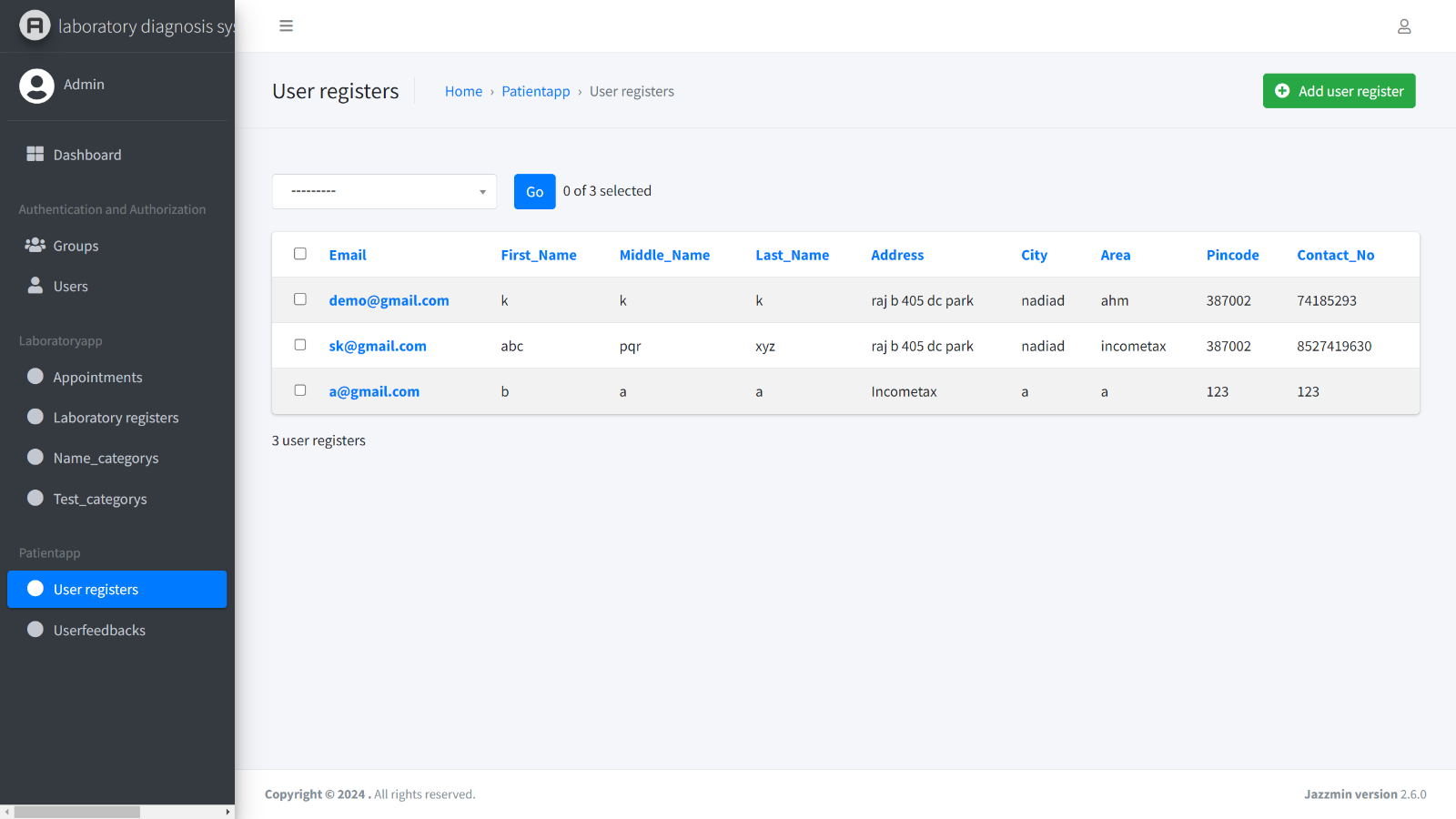
**Day 42**

****

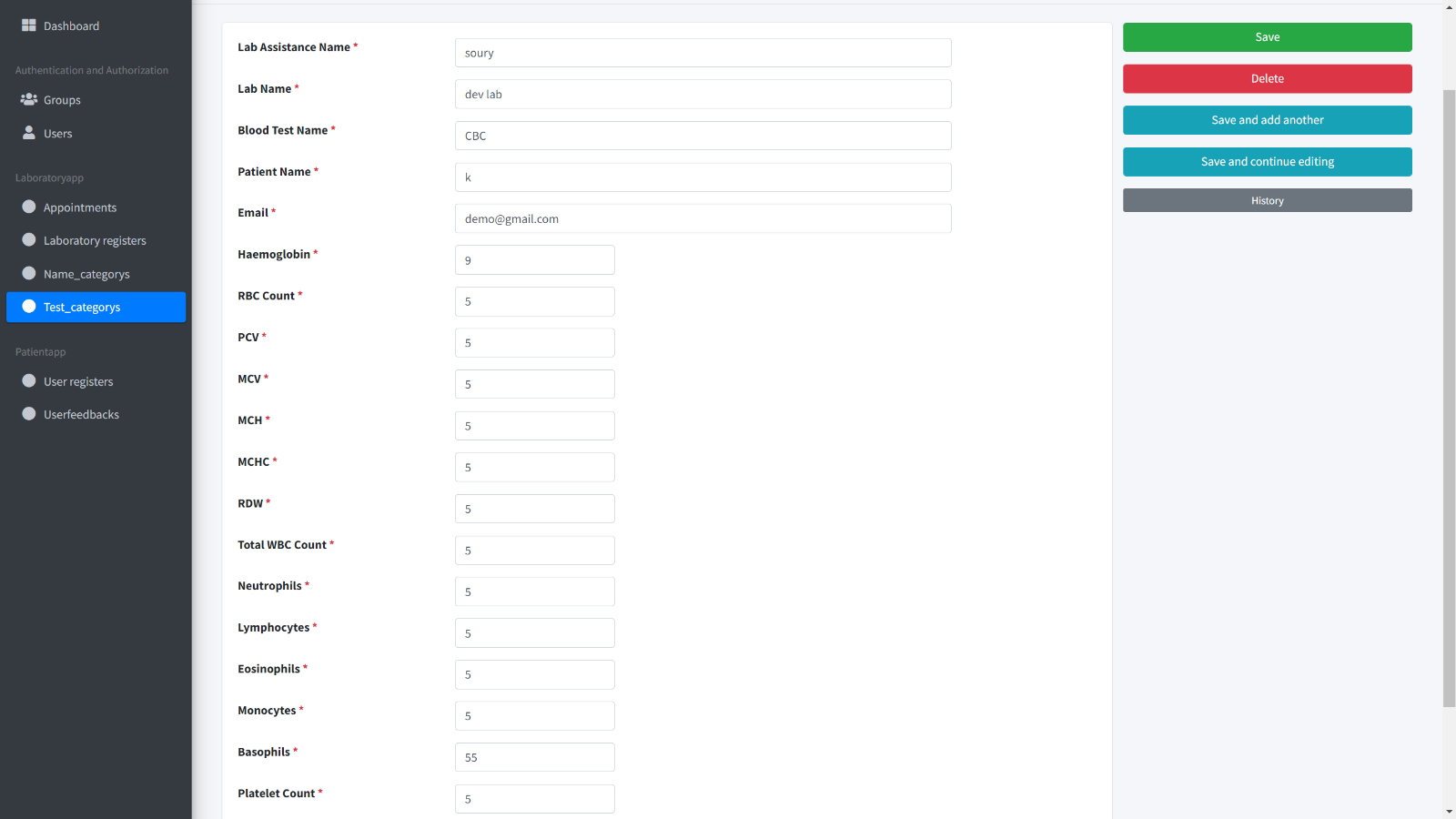
**Day 43**

****

**Day 44**

****

**Day 45**

****

**Tasks**

Task 1: Project Planning

Define project scope, objectives, and requirements

# project\_settings.py

PROJECT\_NAME = "Ecommerce Website"

SCOPE = "To build a fully functional ecommerce platform using Django."

OBJECTIVES = ["Implement user authentication", "Create product management system", "Integrate payment gateway"]

Task 2: Django Setup

Install Django framework.

pip install django

Task 3: Create Django Project

django-admin startproject ecommerce\_project

Task 4: Database Design

Design the database schema.

# models.py

from django.db import models

class Product(models.Model):

name = models.CharField(max\_length=100)

price = models.DecimalField(max\_digits=10, decimal\_places=2)

description = models.TextField()

Task 5: Research and Analysis

* Analyze competitor websites for insights.
* Conduct market research on ecommerce trends.

Task 6: User Authentication

Implement user registration and login functionality.

# views.py

from django.contrib.auth.forms import UserCreationForm

from django.contrib.auth import login, authenticate

def register(request):

form = UserCreationForm()

if request.method == 'POST':

form = UserCreationForm(request.POST)

if form.is\_valid():

user = form.save()

login(request, user)

return redirect('/')

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

Task 7: Product Management

Develop CRUD operations for products.

# views.py

from .models import Product

from django.shortcuts import render, get\_object\_or\_404

def product\_detail(request, product\_id):

product = get\_object\_or\_404(Product, pk=product\_id)

return render(request, 'product/detail.html', {'product': product})

Task 8: Shopping Cart

Create a shopping cart system.

# models.py

class Cart(models.Model):

user = models.OneToOneField(User, on\_delete=models.CASCADE)

items = models.ManyToManyField(Product)

quantity = models.PositiveIntegerField(default=1)

Task 9: Checkout Process

Design and implement a multi-step checkout process.

# views.py

def checkout(request):

if request.method == 'POST':

# Handle payment processing

return redirect('order\_confirmation')

return render(request, 'checkout.html')

Task 10: User Profiles and Orders

Develop user profile pages.

# views.py

def profile(request):

user = request.user

orders = Order.objects.filter(user=user)

return render(request, 'profile.html', {'user': user, 'orders': orders})

Task 11: Backend Administration

Create an admin panel for site management.

# admin.py

from django.contrib import admin

from .models import Product, Order

admin.site.register(Product)

admin.site.register(Order)

Task 12: Base Page

Convert wireframes into responsive web pages.

<!-- base.html -->

<html>

<head>

<title>{% block title %}{% endblock %}</title>

</head>

<body>

{% block content %}

{% endblock %}

</body>

</html>

Task 13: Encrypt Util

from cryptography.fernet import Fernet  
import base64  
import logging  
import traceback  
from django.conf import settings  
  
def encrypt(pas):  
 try:   
 pas = str(pas)  
 cipher\_pass = Fernet(settings.ENCRYPT\_KEY)  
 encrypt\_pass = cipher\_pass.encrypt(pas.encode('UTF-8'))  
 encrypt\_pass = base64.urlsafe\_b64encode(encrypt\_pass).decode("UTF-8")   
 return encrypt\_pass  
 except Exception as e:  
 logging.getLogger("error\_logger").error(traceback.format\_exc())  
 return None

Task 14: Decrypt Util

def decrypt(pas):  
 try:  
 pas = base64.urlsafe\_b64decode(pas)  
 cipher\_pass = Fernet(settings.ENCRYPT\_KEY)  
 decod\_pass = cipher\_pass.decrypt(pas).decode("UTF-8")   
 return decod\_pass  
 except Exception as e:  
 logging.getLogger("error\_logger").error(traceback.format\_exc())  
 return None

Task 15: App Info

from django.apps import AppConfig  
  
  
class App1Config(AppConfig):  
 default\_auto\_field = 'django.db.models.BigAutoField'  
 name = 'app1'

Task 16: Mapping

from django.urls import path  
from .views import \*  
  
urlpatterns = [  
 path('reg/',reg,name='userreg'),  
 # path('reg/',),  
 path('',index,name='index'),  
 path('table/',table,name='table'),  
 path('cat/',cat,name='cat'),  
 path('data/',data,name='data'),  
 path('img/',image,name='img'),  
 path('login/',login,name='login'),  
 path('logout/',logout,name='logout'),  
 path('profile/',profile,name='profile'),  
 path('card/<int:id>',card,name='card'),  
 path('pro/<int:id>',pro,name='pro'),  
 path('prodetails/<int:id>',prodetails,name='prodetails'),  
 path('cart/',cartdata,name='cart'),  
 path('add1/<int:id>',additem,name='add1'),  
 path('minus/<int:id>',minus,name='minus'),  
 path('removeall/',removeall,name='removeall'),  
 path('venreg/',venreg,name='venreg'),  
 path('venlogin/',venlogin,name='venlogin'),  
 path('addpro/',addproducts,name = 'addpro'),  
 path('venpro/',venpro,name = 'venpro'),  
 path('updatepro/<int:id>',updatepro,name='updatepro'),  
 path('deletepro/<int:id>',deletepro,name='deletepro'),  
 path('checkout/',checkout,name='checkout'),  
 path('orderhistory/',orderhistory,name='orderhistory'),  
 path('soldpro/',soldpro,name='soldpro'),  
 path('razorpay/',razorpaypayment,name= 'razorpay'),  
 path('paymenthandler/',paymenthandler,name='paymenthandler'),  
 path('otp/',otp,name='otp'),  
  
]

Task 17: Database Config

DATABASES = {  
 'default': {  
 'ENGINE': 'django.db.backends.mysql',  
 'NAME': 'app1',  
 'HOST': '127.0.0.1',  
 'PORT': '3306',  
 'USER': 'root',  
 'PASSWORD': 'root',  
 }  
}

Task 18: Static & Media Files

STATIC\_URL = '/static/'

import os  
MEDIA\_ROOT = os.path.join(BASE\_DIR,'media/img/')  
MEDIA\_URL = '/media/img/'

Task 19: Email Config

EMAIL\_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'  
EMAIL\_HOST = 'smtp.gmail.com'  
EMAIL\_HOST\_USER = 'pqr6997@gmail.com'  
EMAIL\_HOST\_PASSWORD = 'howk ouoo xljq lqwe'  
EMAIL\_USE\_TLS = True  
EMAIL\_PORT = 587

Task 20: Table Config

def table(request):  
 a = userregister.objects.get(name = 'a')  
 # print(a)  
 # for i in a:  
 # print(i.email)  
 return render(request,'table.html',{'data':a})

Task 21: Categories

def cat(request):  
 if 'email' in request.session:  
 b = userregister.objects.get(email = request.session['email'])  
 a = category.objects.all()  
 return render(request,'cat.html',{'cat':a,'session':b})  
 else:  
 a = category.objects.all()  
 return render(request,'cat.html',{'cat':a})

class category(models.Model):  
 catname = models.CharField(max\_length=50)  
 image = models.ImageField(upload\_to='imgcat')  
  
 def \_\_str\_\_(self):  
 return self.catname

Task 22: Ecom Data Store

def data(request):  
 if request.method == 'POST':  
 a = Blog()   
 a.name = request.POST['uname']  
 a.tagline = request.POST['data']  
 a.save()  
 print("data stored succesfullyy....")  
 return render(request,'form.html')  
 else:  
 print("failed to store data.......")  
 return render(request,'form.html')

Task 23: Image Processing

def image(request):  
 if request.method == 'POST' and request.FILES['img']:  
 a = category()  
 a.catname = request.POST['uname']  
 a.image = request.FILES['img']  
 a.save()  
 # return render(request,'img.html')  
 return redirect('venlogin')  
 else:   
 return render(request,'img.html')

Task 24: OTP Config

def otp(request):  
 if request.method == 'POST':  
 if int(request.session['otp']) == int(request.POST['otp']):  
 return redirect('index')  
 else:  
 return render(request,'otp.html',{'invalid':"Invalid OTP"})  
  
 else:  
 return render(request,'otp.html')

Task 25: Profile Fetch

def profile(request):  
 if 'email' in request.session:  
 a = userregister.objects.get(email = request.session['email'])  
 if request.method == 'POST':  
 a.name = request.POST['uname']  
 a.mob = request.POST['mob']  
 a.add = request.POST['add']  
 a.save()  
 return render(request,'profile.html',{'session':a})  
 else:  
 return render(request,'profile.html',{'session':a})  
 else:  
 return redirect('login')

Task 26: Card

def card(request,id):  
 if 'email' in request.session:  
 b = userregister.objects.get(email = request.session['email'])  
 a = category.objects.get(id = id)  
 return render(request,'card.html',{'cat':a,'session':b})  
 else:  
 return redirect('login')

Task 27: All Products Fetch

def pro(request,id):  
 if 'email' in request.session:  
 b = userregister.objects.get(email = request.session['email'])  
 a = product.objects.filter(category = id)  
 return render(request,'pro.html',{'pro':a,'session':b})  
 else:  
 return redirect('login')

Task 28: Products Details

def prodetails(request,id):  
 if 'email' in request.session:  
 b = userregister.objects.get(email = request.session['email'])  
 a = product.objects.get(id = id)  
 if request.method == 'POST':  
 cartdata = cart()  
 cartdata.productid = id  
 cartdata.userid = request.session['userid']  
 cartdata.quantity = request.POST['qty']  
 cartdata.totalprice = int(request.POST['qty']) \* int(a.price)  
 cartdata.orderid = "0"  
 c = cart.objects.filter(productid = id,userid = request.session['userid'],orderid = "0")   
 if c:  
 return render(request,'product.html',{'session':b,'prodetails':a,'chej':'cheejjjj!!!'})  
 else:  
 cartdata.save()  
 a.qty = a.qty - int(request.POST['qty'])  
 a.save()  
 return render(request,'product.html',{'session':b,'prodetails':a})  
 else:  
 return render(request,'product.html',{'session':b,'prodetails':a})  
 else:  
 return redirect('login')

Task 29: Update Product

def updatepro(request,id):  
 if request.method == 'POST':   
 a = product.objects.get(id = id)  
 a.price = request.POST['price']  
 a.qty = request.POST['qty']  
 a.discrption = request.POST['des']  
 a.save()  
 # return render(request,'updatepro.html',{'proupdatedetail':a})  
 return redirect('venpro')  
 else:  
 a = product.objects.get(id = id)  
 return render(request,'updatepro.html',{'proupdatedetail':a})

Task 30: Delete Product

def deletepro(request,id):  
 if 'vendoremail' in request.session:  
 ven = vendor.objects.get(email = request.session['vendoremail'])  
 a = product.objects.get(id = id)  
 a.delete()  
 return redirect('venpro')  
 else:  
 return redirect('venlogin')

Task 31: Product Sold

def soldpro(request):  
 if 'vendoremail' in request.session:  
 cartdata = cart.objects.all()  
 prolist= []  
 for i in cartdata:  
 print(i,"carrrtttt datatttttaa")  
 if i.orderid != "0":   
 print(i, "ordered dataaaaaa.......")  
 pro = product.objects.get(id = i.productid)  
 print(pro.name,"prooduct nameeeeeeeeee......")  
 print(request.session['vendorid'],pro.vendor,"vendooorrrrr idd......")  
 if int(pro.vendor) == int(request.session['vendorid']):  
 print(pro.name,"product name of vendorr......")  
 userdetails = userregister.objects.get(id = i.userid)  
 prodict = {'proimg':pro.image,'proname':pro.name,'qty':i.quantity,'uname':userdetails.name}  
 prolist.append(prodict)  
 return render(request,'soldpro.html',{'prolist':prolist,'vensession':True})  
 else:  
 return redirect('venlogin')

Tast 32: Order History

def orderhistory(request):  
 if 'email' in request.session:  
 a = cart.objects.filter(userid = request.session['userid'])  
 prolist = []  
 for i in a:  
 if i.orderid != "0":  
 orderdt = order.objects.get(id = i.orderid)  
 prodetails = product.objects.get(id = i.productid)  
 prodict = {'proname':prodetails.name,'proimg':prodetails.image,'prodis':prodetails.discrption,'datetime':orderdt.datetime,'qty':i.quantity,'tp':i.totalprice,'sp':prodetails.price}  
 prolist.append(prodict)  
 else:  
 pass  
 # print(total,"totalll producttsss........")  
 return render(request,'orderhistory.html',{'order':a,'prolist':prolist})  
 else:  
 return redirect('login')

Task 33: Landing Page

def index(request):  
 if 'email' in request.session:  
 b = userregister.objects.get(email = request.session['email'])  
 a = category.objects.all()  
 return render(request,'index.html',{'category':a,'session':b})  
 elif 'vendoremail' in request.session:  
 a = category.objects.all()  
 b = vendor.objects.get(email = request.session['vendoremail'])  
 return render(request,'index.html',{'category':a,'vensession':b})  
 else:  
 a = category.objects.all()  
 return render(request,'index.html',{'category':a})

Task 34: Razorpay Config

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Document</title>  
</head>  
<body>  
 <script src="https://checkout.razorpay.com/v1/checkout.js"></script>  
 <script>  
 var options = {  
   
 // Enter the Key ID generated from the Dashboard  
 key: "{{ razorpay\_merchant\_key }}",   
   
 // Amount is in currency subunits.  
 // Default currency is INR. Hence,   
 // 50000 refers to 50000 paise  
 amount: "{{ razorpay\_amount }}",   
 currency: 'INR',  
   
 // Your/store name.  
 name: "morning E-commerce",   
   
 // Pass the `id` obtained in the response of Step 1  
 order\_id: "{{ razorpay\_order\_id }}",   
 callback\_url: "{{ callback\_url }}",  
 };  
   
 // initialise razorpay with the options.  
 var rzp1 = new Razorpay(options);  
 rzp1.open();  
  
 </script>  
</body>  
</html>

Task 35: Vendor Product

{% extends "nav.html" %}  
{% block abc %}  
{% load static %}  
<!DOCTYPE html>  
<html lang="en">  
 <head>  
 <meta charset="UTF-8" />  
 <meta name="viewport" content="width=device-width, initial-scale=1.0" />  
 <title>E-commerce</title>  
 <link  
 rel="stylesheet"  
 45  
 href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"  
 />  
 <link  
 rel="stylesheet"  
 href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.2/css/all.min.css"  
 integrity="sha512-z3gLpd7yknf1YoNbCzqRKc4qyor8gaKU1qmn+CShxbuBusANI9QpRohGBreCFkKxLhei6S9CQXFEbbKuqLg0DA=="  
 crossorigin="anonymous"  
 referrerpolicy="no-referrer"  
 />  
 <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js"></script>  
 <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js"></script>  
 <link rel="stylesheet" href= {% static "ecom.css" %}/>  
 </head>  
 <body>   
 <!--====================== Navbar ======================-->  
   
 <!--======================Carousel Section======================-->  
 <div id="myCarousel" class="carousel slide" data-ride="carousel">  
 <ol class="carousel-indicators">  
 <li data-target="#myCarousel" data-slide-to="0" class="active"></li>  
 <li data-target="#myCarousel" data-slide-to="1"></li>  
 <li data-target="#myCarousel" data-slide-to="2"></li>  
 </ol>  
  
 <div class="carousel-inner">  
 <div class="carousel-item active">  
 <img src="Img/151.jpg" alt="First Slide" />  
 </div>  
 <div class="carousel-item">  
 <img src="Img/6068656.jpg" alt="Second Slide" />  
 </div>  
 <div class="carousel-item">  
 <img src="Img/3671942.jpg" alt="Third Slide" />  
 </div>  
 </div>  
  
 <a  
 class="carousel-control-prev"  
 href="#myCarousel"  
 role="button"  
 data-slide="prev"  
 >  
 <span class="carousel-control-prev-icon" aria-hidden="true"></span>  
 <span class="sr-only">Previous</span>  
 </a>  
 <a  
 class="carousel-control-next"  
 href="#myCarousel"  
 role="button"  
 data-slide="next"  
 >  
 <span class="carousel-control-next-icon" aria-hidden="true"></span>  
 <span class="sr-only">Next</span>  
 </a>  
 </div>  
 <!--======================Service Section======================-->  
 <section class="container-fluid py-2">  
 <div class="container">  
 <div class="row">  
 <div  
 class="col-4 d-flex justify-content-center align-items-center p-4"  
 >  
 <i class="fa-solid fa-truck-fast fa-2x mr-3"></i>  
 <p class="text-left p-0">  
 Lorem ipsum dolor sit amet, consectetur adipisicing elit. Dolorum  
 </p>  
 </div>  
 <div  
 class="col-4 d-flex justify-content-center align-items-center p-4"  
 >  
 <i class="fa fa-plane-circle-check fa-2x mr-3"></i>  
 <p class="text-left p-0">  
 Lorem ipsum dolor sit amet, consectetur adipisicing elit. Dolorum  
 </p>  
 </div>  
 <div  
 class="col-4 d-flex justify-content-center align-items-center p-4"  
 >  
 <i class="fa fa-handshake fa-2x mr-3"></i>  
 <p class="text-left p-0">  
 Lorem ipsum dolor sit amet, consectetur adipisicing elit. Dolorum  
 </p>  
 </div>  
 </div>  
 </div>  
 </section>  
 <!--======================Collection Section======================-->  
 <section class="container">  
 <div class="mt-3">  
 <div class="row justify-content-center">  
 {% for i in prod %}  
 <div class="col-lg-3 col-md-6 p-0">  
 <div class="card p-1 border-0 scale-style">  
 <a href=>  
 <img  
 class="card-img-top img-hover"  
 src={{i.image.url}}  
 alt="Card image cap"  
 style="height: 300px"  
 />  
 </a>  
 <div class="card-body text-center">  
 <p class="card-text text-center">{{i.catname}}</p>  
 <a href={% url "deletepro" i.id %}>  
 <button type="button" class="btn btn-outline-danger mt-2">  
 Delete  
 </button>  
 </a>  
 <a href={% url "updatepro" i.id %}>  
 <button type="button" class="btn btn-outline-success mt-2">  
 Update  
 </button>  
 </a>  
 </div>  
 </div>  
 </div>  
 {% endfor %}  
  
 </div>  
 </div>  
 </section>  
 <!--======================Footer Section======================-->  
 <footer class="bg-dark text-light py-4">  
 <div class="container">  
 <div class="row">  
 <div class="col-md-4">  
 <h5>Contact Us</h5>  
 <p>Email: info@example.com</p>  
 <p>Phone: 123-456-7890</p>  
 </div>  
 <div class="col-md-4">  
 <h5>Links</h5>  
 <ul class="list-unstyled">  
 <li><a href="#">Home</a></li>  
 <li><a href="#">Products</a></li>  
 <li><a href="#">About Us</a></li>  
 <li><a href="#">Contact Us</a></li>  
 </ul>  
 </div>  
 <div class="col-md-4">  
 <h5>Follow Us</h5>  
 <ul class="list-inline">  
 <li class="list-inline-item">  
 <a href="#"><i class="fab fa-facebook"></i></a>  
 </li>  
 <li class="list-inline-item">  
 <a href="#"><i class="fab fa-twitter"></i></a>  
 </li>  
 <li class="list-inline-item">  
 <a href="#"><i class="fab fa-instagram"></i></a>  
 </li>  
 </ul>  
 </div>  
 </div>  
 <hr />  
 <div class="row">  
 <div class="col text-center">  
 <p>&copy; 2023 Your E-Commerce Store</p>  
 </div>  
 </div>  
 </div>  
 </footer>  
 </body>  
</html>  
{% endblock abc %}

Task 36: Vendor Product Fetch

def venpro(request):  
 if 'vendoremail' in request.session:  
 b = vendor.objects.get(email = request.session['vendoremail'])  
 c = product.objects.filter(vendor = b.pk)  
 return render(request,'venpro.html',{'vensession':b,'prod':c})  
 else:  
 return redirect('venlogin')

Task 37: Navigation

<!DOCTYPE html>  
<html lang="en">  
 <head>  
 <meta charset="UTF-8" />  
 <meta name="viewport" content="width=device-width, initial-scale=1.0" />  
 <title>E-commerce</title>  
 <link  
 rel="stylesheet"  
 href="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/css/bootstrap.min.css"  
 integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"  
 crossorigin="anonymous"  
 />  
 <link  
 rel="stylesheet"  
 href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.2/css/all.min.css"  
 integrity="sha512-z3gLpd7yknf1YoNbCzqRKc4qyor8gaKU1qmn+CShxbuBusANI9QpRohGBreCFkKxLhei6S9CQXFEbbKuqLg0DA=="  
 crossorigin="anonymous"  
 referrerpolicy="no-referrer"  
 />  
 <script  
 src="https://code.jquery.com/jquery-3.2.1.slim.min.js"  
 integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN"  
 crossorigin="anonymous"  
 ></script>  
 <script  
 src="https://cdn.jsdelivr.net/npm/popper.js@1.12.9/dist/umd/popper.min.js"  
 integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q"  
 crossorigin="anonymous"  
 ></script>  
 <script  
 src="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/js/bootstrap.min.js"  
 integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl"  
 crossorigin="anonymous"  
 ></script>  
 <link rel="stylesheet" href="Style.css" />  
 </head>  
<body>  
 <!--====================== Navbar ======================-->  
 <header class="container-fluid navbar-light bg-light">  
 <nav class="navbar container navbar-expand-lg navbar-light bg-light">  
 <a class="navbar-brand" href="index.html">Navbar</a>  
 <button  
 class="navbar-toggler"  
 type="button"  
 data-toggle="collapse"  
 data-target="#navbarSupportedContent"  
 aria-controls="navbarSupportedContent"  
 aria-expanded="false"  
 aria-label="Toggle navigation"  
 >  
 <span class="navbar-toggler-icon"></span>  
 </button>  
   
 <div class="collapse navbar-collapse" id="navbarSupportedContent">  
 <ul class="navbar-nav ml-auto">  
 <li class="nav-item">  
 <a class="nav-link" href={% url "index" %}>Home</a>  
 </li>  
 {% if session %}  
 <li class="nav-item">  
 <a class="nav-link" href={% url "logout" %}>Logout</a>  
 </li>  
 <li class="nav-item">  
 <a class="nav-link" href="orderhistory.html">Order History</a>  
 </li>  
 {% elif vensession %}  
 <li class="nav-item">  
 <a class="nav-link" href={% url "logout" %}>Logout</a>  
 </li>  
 <li class="nav-item">  
 <a class="nav-link" href={% url "addpro" %}>Start selling..!</a>  
 </li>  
 <li class="nav-item">  
 <a class="nav-link" href={% url "soldpro" %}>Sold Products</a>  
 </li>  
 {% else %}  
 <li class="nav-item">  
 <a class="nav-link" href={% url "login" %}>User Login</a>  
 </li>  
 <li class="nav-item">  
 <a class="nav-link" href={% url "venlogin" %}>vendor Login</a>  
 </li>  
 {% endif %}  
 <li class="nav-item">  
 <a class="nav-link" href={% url "profile" %}>Profile</a>  
 </li>  
   
 <li class="nav-item">  
 <a class="nav-link" href="addproduct.html">  
 <i class="fa-solid fa-plus-square text-dark" aria-hidden="true">  
 Add product</i  
 >  
 </a>  
 </li>  
 <li class="nav-item">  
 <input  
 class="input-group-text"  
 type="text"  
 placeholder="Search"  
 />  
 </li>  
 <li class="nav-item">  
 <a class="nav-link" href={% url "cart" %}  
 ><i class="fa fa-cart-shopping text-dark" aria-hidden="true"></i  
 ></a>  
 </li>  
 {% if session %}  
 <li class="nav-item">  
 <a class="nav-link" href="#">Welcome,{{session.name}}</a>  
 </li>  
 {% elif vensession %}  
 <li class="nav-item">  
 <a class="nav-link" href="#">Welcome,{{vensession.name}}</a>  
 </li>  
 <li class="nav-item">  
 <a class="nav-link" href={% url "venpro" %}>My products</a>  
 </li>  
 {% endif %}  
 </ul>  
 </div>  
 </nav>  
 </header>  
 <!-- content -->  
 {% block abc %}  
 {% endblock abc %}  
</body>  
</html>

Task 38: Generalized Form

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Document</title>  
</head>  
<body>  
 <form method="post">  
 {% csrf\_token %}  
 <label for="">Enter name: </label>  
 <input type="text" name="uname">  
  
 <label for="">Enter text: </label>  
 <input type="text" name="data">  
  
 <button type="submit"> Submit </button>  
 </form>  
</body>  
</html>

Task 39: Payment Success Check

<h2>Payment Success.......</h2>

Task 40: Necessary Settings

INSTALLED\_APPS = [  
 'jazzmin',  
 'django.contrib.admin',  
 'django.contrib.auth',  
 'django.contrib.contenttypes',  
 'django.contrib.sessions',  
 'django.contrib.messages',  
 'django.contrib.staticfiles',  
 'app1',  
  
]

Task 41: Define Encryption Key

import os  
ENCRYPT\_KEY = b'DJ3r34r9zSRPM8OWucrsv2PDBEA6BYaGpfJx67C9\_us='

Task 42: Testing and Optimization

Write unit tests for each module and functionality.

# tests.py  
from django.test import TestCase  
from .models import Product  
  
class ProductTestCase(TestCase):  
 def setUp(self):  
 Product.objects.create(name="Test Product", price=10.0, description="Test Description")  
  
 def test\_product\_creation(self):  
 product = Product.objects.get(name="Test Product")  
 self.assertEqual(product.price, 10.0)

Task 43: Deployment

Set up hosting environment.

# Example: Heroku deployment

heroku create

git push heroku master

heroku open

Task 44: Documentation

Document codebase, APIs, and configurations.

## API Documentation

### Products

- \*\*GET /api/products\*\*: Get all products

- \*\*POST /api/products\*\*: Create a new product

## Configuration

- \*\*Database\*\*: PostgreSQL

- \*\*Payment Gateway\*\*: Stripe

Task 45: Post-Deployment Support

* Monitor website performance and user feedback.
* Provide ongoing maintenance and support.