Criterion C : Development

List of Techniques Used :

1. Use of Various Languages
2. Use of Additional Libraries
3. Backend Modules / Apps
4. Models
    a. One-to-One Relations
    b. Django Signals on model save
5. Views
6. Templates
7. URLs
8. Model Forms
9. Static Files Configuration
10. Custom Validators
11. Template Context Processors
12. Overwriting Django Default Templates
13. Modifying the Admin Site
14. Notification Emails using SMTP
15. Cryptographic Token Generation for Password Reset

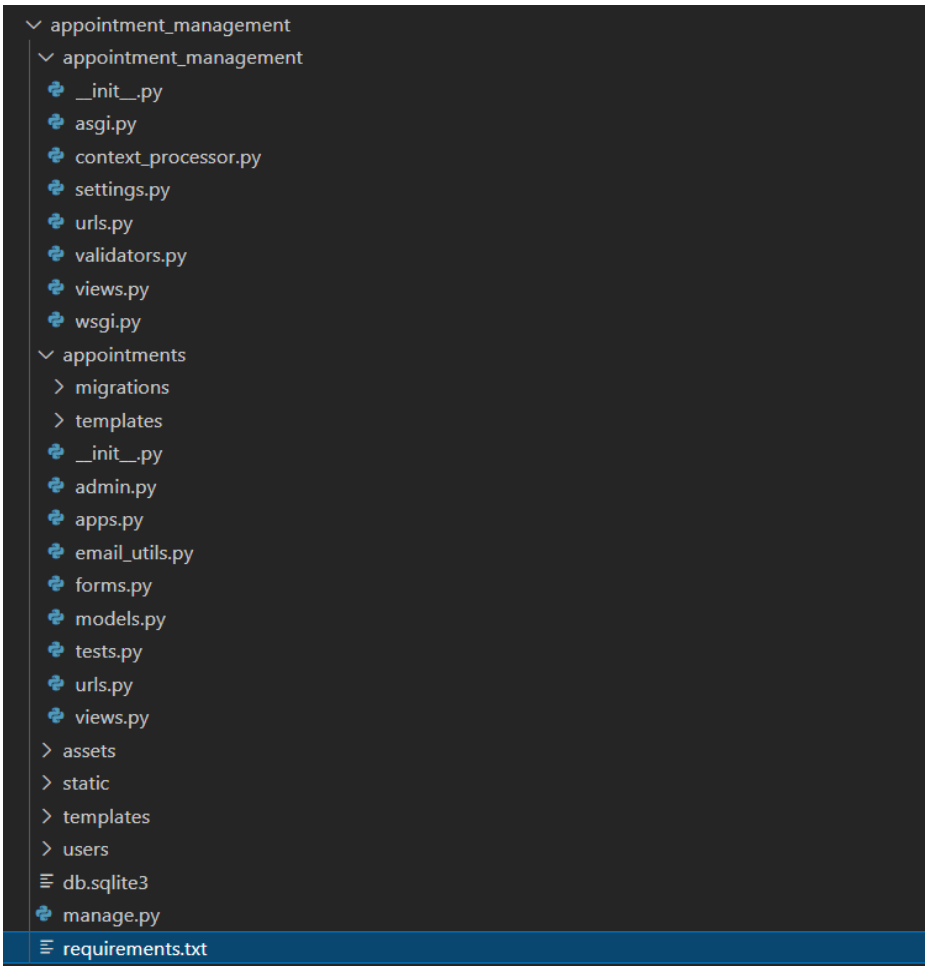
1. Use of Various Languages

The project is a combination of backend and front-end programming languages.

a. The backend is developed using the Django framework in Python.

Django is a Web Application Development Framework which is used to develop web based applications using the Python programming language. It is based on the MVT (Model View Template) design pattern.

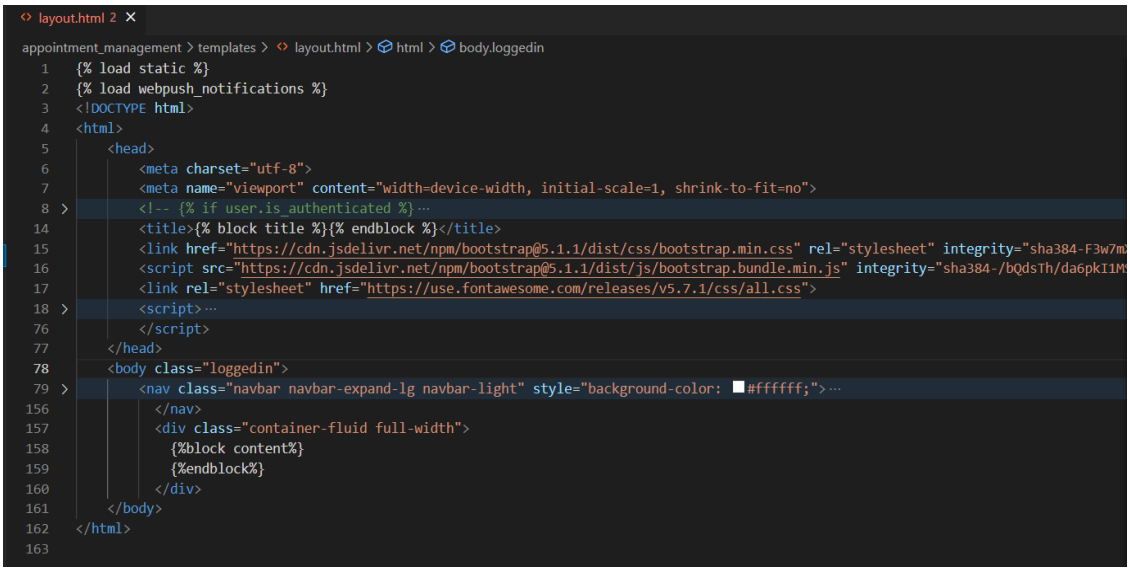
Project Structure



b. The front-end consists of 3 languages, each having its own purpose:

i. HTML 5

Hyper Text Markup Language is the standard markup language for creating Web pages. It provides structure to the webpages using a tree of various elements. We use HTML inside the django templates to create the basic layout of the pages.



ii. CSS 3

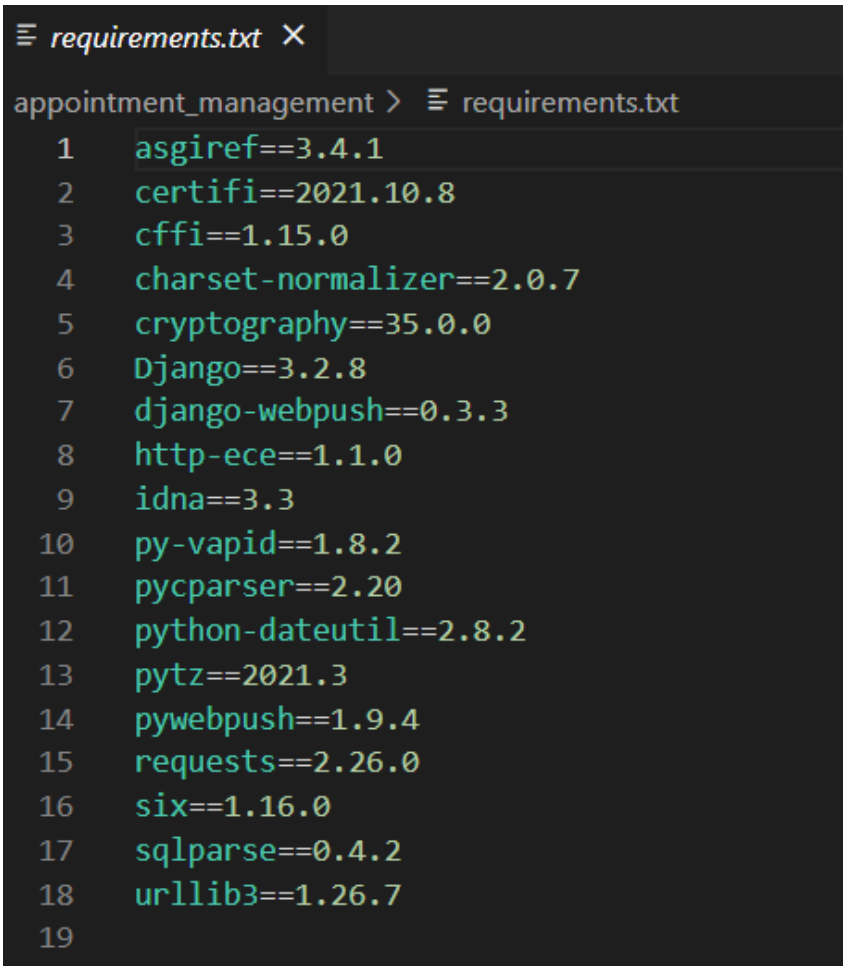
Cascading Style Sheets are used to format the layout of the web page. CSS files define how the html elements would be stylized, positioned and animated.
The CSS files for the project are placed in the `static/css` directory. Some external CSS libraries like bootstrap have also been used to provide responsiveness and stylize components of the website. These libraries are imported from CDNs provided by the library creator. Additional styling for specific elements within the pages are also enclosed within the `<style>` tag or placed inline with the element by using the `style` attribute.

iii. JS

Javascript is used to add logic and responsiveness into the front-end of the application. It works on the browser and is used to handle events, validate forms, show messages, request and display dynamic data and applets. The javascript code in the project is placed at three different sources:

a) External JS libraries like bootstrap JS, MC-Calendar JS and Clocklet JS are sourced from CDNs hosted by their respective providers.
b) The JS code responsible for the logic of each page is written directly inside the html document using `<script>` tags.
c) The repetitive code and functions that are present in various pages are placed in the `static/js` folder

2. Use of Additional Libraries
    a. The frontend libraries have been used are :
        i. Bootstrap v5.1.1 CSS and JS
        ii. FontAwesome Icons v5.7.1 CSS
        iii. Mc-calendar CSS and JS
        iv. Clocklet CSS and JS
    b. The backend libraries required are :



We use the following command to install these dependencies:
`python3 -m pip install -r requirements.txt`

3. Backend Modules / Apps

An app is a reusable and independently maintainable part of a project that provides a modular way of maintaining the system. Our project is subdivided into two applications i.e. the appointments module and the users module that manages different aspects of the application.

a. The appointments app encapsulates all the templates,views,models and other code required for the appointment booking system.
b. The user module contains all the code to manage the users of the system.

4. Models

A model is the single, definitive source of information about your data. It contains the essential fields and behaviors of the data you're storing coupled with methods to access the data. Generally, each model maps to a single database table.

User Models

models.py

appointment_management > users > models.py > ...

```python
from django.db import models
from django.contrib.auth.models import User
from django.contrib.auth.models import Group
from django.core.validators import RegexValidator
from django.db.models.signals import post_save
from django.dispatch import receiver

class SexChoices(models.TextChoices):
    MALE = "Male", 'Male'
    FEMALE = "Female", 'Female'
    OTHERS = "Others", 'Others'

class PatientProfile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE,related_name='patient')
    phone_regex = RegexValidator(regex=r'^\+?1?\d{9,15}$', message="Phone number must be entered in the format: '+999999999'. Up to 15 digits allowed.")
    phone = models.CharField(max_length=17,validators=[phone_regex], blank=True)
    address = models.CharField(max_length=200,blank=True,default="")
    dob = models.DateField(blank=True)
    sex = models.TextField(max_length=10,choices=SexChoices.choices,blank=True)
    history = models.CharField(max_length=300,blank=True,default="")
    def __str__(self):
        return self.user.get_full_name()
@receiver(post_save, sender=PatientProfile, dispatch_uid="set_user_patient")
def patient_postsave(sender, instance, **kwargs):
    Group.objects.get_or_create(name = 'patient')[0].user_set.add(instance.user)

class DoctorProfile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE,related_name='doctor')
    phone_regex = RegexValidator(regex=r'^\+?1?\d{9,15}$', message="Phone number must be entered in the format: '+999999999'. Up to 15 digits allowed.")
    phone = models.CharField(max_length=17,validators=[phone_regex], blank=True)
    qualifications = models.CharField(max_length=70, blank=True,default="")
    def __str__(self):
        return self.user.get_full_name()
@receiver(post_save, sender=DoctorProfile, dispatch_uid="set_user_doctor")
def doctor_postsave(sender, instance, **kwargs):
    Group.objects.get_or_create(name = 'doctor')[0].user_set.add(instance.user)
```

One-to-One Relations

The user profile models are one-to-one linked to django's default User model which provides the basic fields like username,email,password along with authentication and authorization functions.

Django Signals on model save

Django includes a "signal dispatcher" which helps decoupled applications get notified when actions occur elsewhere in the framework. In a nutshell, signals allow certain senders to notify a set of receivers that some action has taken place. In our case, we use the post-save signal from

the user models to assign the users to their respective groups. The Group model is a django default model used to decide permissions and other logic based on user type.

Appointments Model

The Appointment model stores all the appointment data and has **customized methods** to save and retrieve the data.

```python
class Appointment(models.Model):
    doctor = models.ForeignKey(DoctorProfile, on_delete= models.CASCADE,related_name="doctor",null=True)
    status = models.IntegerField(choices=AppointmentStatus.choices,default=AppointmentStatus.OPEN)
    date = models.DateField(default=timezone.now)
    start_time = models.TimeField(blank=False)
    end_time = models.TimeField(blank=True,null=True)
    notes = models.CharField(max_length=300,blank=True,default="")
    patient = models.ForeignKey(PatientProfile, on_delete=models.CASCADE,blank=True,null=True,default=None,related_name="patient")
    apt_type = models.CharField(max_length=50,blank=True,default="")
    billing_amount = models.FloatField(default=0.00,blank=True)
    payment_status = models.IntegerField(choices=PaymentStatus.choices, default=PaymentStatus.UNSET)
    def save(self, *args, **kwargs):
        # overriding save method
        if self.status == AppointmentStatus.OPEN:
            self.notes=""
            self.patient=None
            self.apt_type=""
            self.billing_amount=0.00
            self.payment_status=PaymentStatus.UNSET
        super(Appointment, self).save(*args, **kwargs)
    def get_string_fields(self):
        return [
            ("doctor",self.doctor.user.get_full_name() if self.doctor else ""),
            ("date",self.date.strftime("%d-%m-%Y")),
            ("start_time",self.start_time.strftime("%I:%M %p") if self.start_time is not None else ""),
            ("end_time",self.end_time.strftime("%I:%M %p") if self.end_time is not None else ""),
            ("status",self.get_status_display()),
            ("patient",self.patient.user.get_full_name() if self.patient else ""),
            ("apt_type",self.apt_type),
            ("billing_amount",self.billing_amount),
            ("payment_status",self.get_payment_status_display),
            ("notes",self.notes),
        ]
    class Meta:
        ordering = ['date']
    def __str__(self):
        return self.doctor.user.get_full_name()+ " " + self.date.strftime("%d-%m-%Y") + " " + self.start_time.strftime("%H:%M") + "
        " + str(self.status)
```

These are the choices fields and choice model used by the appointment model:

```python
class AppointmentStatus(models.IntegerChoices):
    OPEN = 0, 'Open'
    BOOKED = 1, 'Booked'
    COMPLETED = 2, 'Completed'
    CANCELLED = 3, 'Cancelled'
    CHECKED_IN = 4, 'Checked In'
    MISSED = 5, 'Missed'

class PaymentStatus(models.IntegerChoices):
    DUE = 0, 'Due'
    PAID = 1, 'Paid'
    UNSET = 2, 'Unset'

class AppointmentTypes(models.Model):
    type = models.CharField(max_length=50,blank=True,default="")
    def __str__(self):
        return self.type

```

5. Views

A view function is a Python function that takes a web request, performs the necessary logic and operations and returns a web response. We have used views to perform all the business logic of our apps. The views return templates that are then rendered by the browser for the user.

User App Views

a. Login/Logout Views:

i. The loginpage view is responsible for checking usernames and passwords and giving users access to the system.

ii. The logout_next view renders the “Thank You” page when the user logs out of the system.

appointment_management > users > views.py > ...

```
179
180  def loginpage(request):
181      if(request.user.is_authenticated):
182          return redirect('home')
183      if request.method == 'POST':
184          username = request.POST.get('username')
185          password = request.POST.get('password')
186          user = authenticate(username=username, password=password)
187          if user is not None:
188              if user.is_active:
189                  login(request, user)
190                  return redirect('home')
191              else:
192                  return render(request,'login.html',{"msg":"Account is not active"})
193          else:
194              try:
195                  user_temp = User.objects.get(username=username)
196                  if user_temp.is_active:
197                      return render(request,'login.html',{"msg":"Invalid username/password"})
198                  else:
199                      return render(request,'login.html',{"msg":"Account is not active. Check Email."})
200              except:
201                  pass
202              return render(request,'login.html',{"msg":"Invalid username/password"})
203      return render(request,'login.html',{"msg":""})
204
205  def logout_next(request):
206      return render(request,'logout_next.html',{"msg":""})
```

b. Register Views

The Register views handle the registration logic and renders the registration templates. The doctor’s and patient registration views are similar but they take in different information relevant to the user type.

```python
def doctor_register(request):
    if request.method == 'GET':
        return render(request,'register_dr.html',{"msg":password_validators_help_text_html()})
    elif request.method == 'POST':
        if request.POST.get('password') != request.POST.get('password_confirm'):
            return render(request,'register_dr.html',{"msg":"<ul><li>Passwords do not match</ul></li>"
            +password_validators_help_text_html()})
        try:
            validate_password(request.POST.get('password'))
            user = User.objects.create_user(username=request.POST.get('email'),first_name=request.POST.get('first_name'),
            last_name=request.POST.get('last_name'), password=request.POST.get('password'), email=request.POST.get('email'))
            user.is_active = False
            user.save()
        except ValidationError as e:
            a="<ul>"
            for i in e.messages:
                a+="<li>"+i+"</li>"
            a+="</ul>"
            print(a)
            return render(request,'register_dr.html',{"msg":a+password_validators_help_text_html()})
        except:
            return render(request,'register_dr.html',{"msg":"<ul><li>User already exists, Please try with another email.</ul></li>"
            +password_validators_help_text_html()})
        DoctorProfile.objects.create(user = user, qualifications = request.POST.get('qualifications'), phone = request.POST.get
        ('phone'))
        current_site = get_current_site(request)
        mail_subject = 'Activate your Skin Studio Account.'
        message = render_to_string('email_activation.html', {
            'user': user,
            'domain': current_site.domain,
            'uid': urlsafe_base64_encode(force_bytes(user.pk)),
            'token': account_activation_token.make_token(user),
        })
        to_email = request.POST.get('email')
        email = EmailMessage(
            mail_subject, message, to=[to_email]
        )
        email.send()
        return render(request,'register.html',{"msg":"You have been successfully registered. To activate your account please check
        your email."})
```

Returns the registration form on GET request

Checks all inputs on POST request and returns errors for invalid inputs.

Creates a new profile if everything is correct and sends an account activation email

c. Profile Views

The profile view is used to view and modify the user's personal details.

```
@login_required
def profile(request):
    if request.method == 'GET':
        data={}
        if is_user_doctor(request.user):
            profile = DoctorProfile.objects.get(user=request.user)
            data['qualifications'] = profile.qualifications
            data['phone']=profile.phone
            return render(request,'profile.html',data)
        elif is_user_patient(request.user):
            profile = PatientProfile.objects.get(user=request.user)
            data['user']=request.user
            data['address']=profile.address
            data['dob']=profile.dob.strftime('%Y-%m-%d')
            data['phone']=profile.phone
            data['history']=profile.history
            print(data)
            return render(request,'profile.html',data)
        else:
            return redirect('/admin')
    elif request.method == 'POST':
        print(request.POST)
        if is_user_doctor(request.user):
            profile = DoctorProfile.objects.get(user=request.user)
            user = User.objects.get(username=request.user.username)
            user.first_name = request.POST.get('first_name')
            user.last_name = request.POST.get('last_name')
            user.email = request.POST.get('email')
            user.username = request.POST.get('email')
            profile.phone = request.POST.get('phone')
            profile.qualifications = request.POST.get('qualifications')
            user.save()
            profile.save()
            return HttpResponseRedirect('/?message=10')
        elif is_user_patient(request.user):
            profile = PatientProfile.objects.get(user=request.user)
            user = User.objects.get(username=request.user.username)
            user.first_name = request.POST.get('first_name')
            user.last_name = request.POST.get('last_name')
            user.email = request.POST.get('email')
            user.username = request.POST.get('email')
            profile.address = request.POST.get('address')
            profile.dob = request.POST.get('dob')
            profile.phone = request.POST.get('phone')
            profile.history = request.POST.get('history')
            user.save()
            profile.save()
            return HttpResponseRedirect('/?message=10')
```

d. Patient Profile View

This view is used to give doctors access to their patient's medical and appointment history.

```python
def patient_profile(request,id):
    if request.method == 'GET':
        data={}
        if is_user_doctor(request.user):
            profile = PatientProfile.objects.get(id=id)
            data['patient'] = profile
            data['user']=profile.user
            data['address']=profile.address
            data['dob']=profile.dob.strftime('%Y-%m-%d')
            data['phone']=profile.phone
            data['history']=profile.history
            print(data)
            data['appointments'] = Appointment.objects.filter(patient=profile, doctor=DoctorProfile.objects.get(user = request.user))
            return render(request,'patient_profile.html',data)
        else:
            return redirect('/')
```

Appointments App Views

a. Appointment ListView

The appointment list view is responsible for the appointment listing and filters. This is a class based view. The get_queryset method is used to retrieve the list of appointments based on the filters and user type. The get_context_data method provides additional data to the template. This class is used to render the apt_list.html template.

```
class AppointmentsList(ListView):
    template_name = "apt_list.html"
    model = Appointment
    context_object_name = "object_list"
    ordering = ["start_time"]

    def get_queryset(self):
        try:
            user = self.request.user
            date = self.request.GET.get("date", None)
            status = self.request.GET.get("status", None)
            doc = self.request.GET.get("doc", None)
            if is_user_patient(user):
                patient = PatientProfile.objects.get(user=user)
                appt = self.model.objects.filter((Q(patient=patient) | Q(status=AppointmentStatus.OPEN)))
                if doc:
                    appt = appt.filter(doctor=DoctorProfile.objects.get(pk=doc))
                if date:
                    date = datetime.strptime(date, "%Y-%m-%d").date()
                    appt = appt.filter(Q(date=date))
                if status == "0":
                    appt = appt.filter(Q(status=status) & Q(date__gte=datetime.today()))
                if status == "5":
                    appt.filter(
                        Q(status=AppointmentStatus.BOOKED) & Q(date__lt=datetime.today())).update(status=AppointmentStatus.MISSED)
                    appt = appt.filter(Q(status=status))
                elif status:
                    appt = appt.filter(Q(status=status))
                else:
                    appt = appt.filter(Q(patient=patient) | Q(date__gte=datetime.today()))
                return appt
            elif is_user_doctor(user):
                doctor = DoctorProfile.objects.get(user=user)
                appt = self.model.objects.filter(doctor=doctor)
                if date == "all":
                    pass
                elif date:
                    date = datetime.strptime(date, "%Y-%m-%d").date()
                    appt = appt.filter(Q(date=date))
                if status == "0":
                    appt = appt.filter(Q(status=status) & Q(date__gte=datetime.today()))
                if status == "5":
                    appt.filter(
                        Q(status=AppointmentStatus.BOOKED)& Q(date__lt=datetime.today())).update(status=AppointmentStatus.MISSED)
                    appt = appt.filter(Q(status=status))
                elif status:
                    appt = appt.filter(Q(status=status))
                else:
                    appt = appt.filter(Q(date__gte=datetime.today()))
                return appt
            else:
                return self.model.objects.filter(status=AppointmentStatus.OPEN)
        except Exception as e:
            print(e)
            return self.model.objects.filter(status=AppointmentStatus.OPEN)

    def get_context_data(self, **kwargs):
        context = super(AppointmentsList, self).get_context_data(**kwargs)
        status = self.request.GET.get("status", None)
        context["reschedule"] = self.request.GET.get("reschedule", None)
        context["apttype"] = AppointmentTypes.objects.all()
        if is_user_patient(self.request.user):
            doc = self.request.GET.get("doc", None)
            patient = PatientProfile.objects.get(user=self.request.user)
            if doc:
                context["aptdate"] = self.model.objects.filter((Q(patient=patient) | Q(status=AppointmentStatus.OPEN)) & Q(doctor=DoctorProfile.objects.get(pk=doc)))
            else:
                context["aptdate"] = self.model.objects.filter((Q(patient=patient) | Q(status=AppointmentStatus.OPEN)))
            context["doctors"] = DoctorProfile.objects.all()
        elif is_user_doctor(self.request.user):
            context["aptdate"] = self.model.objects.filter(doctor=DoctorProfile.objects.get(user=self.request.user))
            context["doctors"] = DoctorProfile.objects.get(user=self.request.user)
        if status == "0":
            context["aptdate"] = context["aptdate"].filter(Q(status=status) & Q(date__gte=datetime.today()))
        elif status:
            context["aptdate"] = context["aptdate"].filter(Q(status=status))
        else:
            context["aptdate"] = context["aptdate"].filter(Q(date__gte=datetime.today()))
        return context
```

b. Add Appointment

```python
@login_required
def add_appointment(request):
    context = {}
    context["apttype"] = AppointmentTypes.objects.all()
    print(context["apttype"])
    print("ADD APPT")
    if request.method == "GET":
        if is_user_patient(request.user):
            return HttpResponse("Unauthorized")
        form = appointmentForm()
        context["form"] = form
        context["fntype"] = "Create"
        return render(request, "apt_create_view.html", context)
    elif request.method == "POST":
        print("POST")
        doctor = DoctorProfile.objects.get(user=request.user)
        form = appointmentForm(request.POST)
        if form.is_valid():
            instance = form.save()
            instance.doctor = doctor
            instance.save()
            send_doctor_email(instance, request, "0")
            return HttpResponseRedirect("/appointments/?status=0&message=0")
        else:
            print(form.errors)
            context["form"] = form
            context["fntype"] = "Create"
            return render(request, "apt_create_view.html", context)
```

c. Edit Appointment

```python
@login_required
def edit_appointment(request, id):
    print("EDIT APPT")
    context = {}
    context["apttype"] = AppointmentTypes.objects.all()
    print(context["apttype"])
    if request.method == "GET":
        appointment = get_object_or_404(Appointment, pk=id)
        if is_user_patient(request.user):
            if appointment.patient.user == request.user:
                form = patientAppointmentForm(instance=appointment)
            else:
                return HttpResponse("Unauthorized")
        else:
            form = appointmentForm(instance=appointment)
        context["form"] = form
        context["fntype"] = "Edit"
        return render(request, "apt_create_view.html", context)
    elif request.method == "POST":
        appointment = get_object_or_404(Appointment, pk=id)
        if is_user_patient(request.user):
            if appointment.patient.user == request.user:
                form = patientAppointmentForm(request.POST, instance=appointment)
            else:
                return HttpResponse("Unauthorized")
        else:
            form = appointmentForm(request.POST, instance=appointment)
        if form.is_valid():
            form.save()
            # Send notification to user about changes
            send_doctor_email(appointment, request, "2")
            send_patient_email(appointment, request, "2")
            return HttpResponseRedirect(
                "/appointments/?message=2&status=" + str(appointment.status)
            )
        else:
            print(form.errors)
            context["form"] = form
            context["fntype"] = "Edit"
            return render(request, "apt_create_view.html", context)
```

d. Delete Appointment

```
def delete_appointment(request, id):
    appointment = get_object_or_404(Appointment, pk=id)
    if appointment.doctor.user == request.user:
        if appointment.status != AppointmentStatus.OPEN:
            appointment.status = AppointmentStatus.CANCELLED
            appointment.save()
            send_patient_email(appointment, request, "3")
            send_doctor_email(appointment, request, "3")
            return HttpResponseRedirect("/appointments/?message=3")
        appointment.delete()
        return redirect("appointments")
    else:
        return HttpResponse("You are not authorized to delete this appointment")
```

e. Cancel Appointment

```
def cancel_appointment(request, id):
    if request.method == "GET":
        appointment = get_object_or_404(Appointment, pk=id)
        if appointment.patient.user == request.user:
            if appointment.status == AppointmentStatus.BOOKED:
                # Send notification to doctor here
                appointment.status = AppointmentStatus.CANCELLED
                appointment.save()
                Appointment.objects.create(doctor=appointment.doctor, date=appointment.date, start_time=appointment.start_tim
                end_time=appointment.end_time, status=AppointmentStatus.OPEN)
                send_doctor_email(appointment, request, "4")
                send_patient_email(appointment, request, "4")
                return HttpResponseRedirect("/appointments/?message=4&status=3")
            return HttpResponseRedirect("/appointments/")
        else:
            return HttpResponse("You are not authorized to cancel this appointment")
```

f. Complete Appointment

```
def complete_appointment(request, id):
    if request.method == "GET":
        appointment = get_object_or_404(Appointment, pk=id)
        if appointment.patient.user == request.user:
            if appointment.status == AppointmentStatus.CHECKED_IN:
                # Send notification to doctor here
                appointment.status = AppointmentStatus.COMPLETED
                appointment.save()
                send_patient_email(appointment, request, "6")
                send_doctor_email(appointment, request, "6")
                return HttpResponseRedirect("/appointments/?message=6&status=2")
            return HttpResponseRedirect("/appointments/")
        else:
            return HttpResponse("You are not authorized to checkin this appointment")
```

g. Check-in appointment

```
def checkin_appointment(request, id):
    if request.method == "GET":
        appointment = get_object_or_404(Appointment, pk=id)
        if appointment.patient.user == request.user:
            if appointment.status == AppointmentStatus.BOOKED:
                # Send notification to doctor here
                appointment.status = AppointmentStatus.CHECKED_IN
                appointment.save()
                send_patient_email(appointment, request, "7")
                send_doctor_email(appointment, request, "7")
                return HttpResponseRedirect("/appointments/?message=7&status=4")
            return HttpResponseRedirect("/appointments/")
        else:
            return HttpResponse("You are not authorized to checkin this appointment")
```

h. Cancel Check-in appointment

```
def cancelCheckIn_appointment(request, id):
    if request.method == "GET":
        appointment = get_object_or_404(Appointment, pk=id)
        if appointment.patient.user == request.user:
            if appointment.status == AppointmentStatus.CHECKED_IN:
                # Send notification to doctor here
                appointment.status = AppointmentStatus.BOOKED
                appointment.save()
                send_patient_email(appointment, request, "8")
                send_doctor_email(appointment, request, "8")
                return HttpResponseRedirect("/appointments/?message=8&status=1")
            return HttpResponseRedirect("/appointments/")
        else:
            return HttpResponse("You are not authorized to checkin this appointment")
```

i. Book Appointment

```
def book_appointment(request, id):
    """
    This function is called when a patient books or reschedules an appointment
    """
    if is_user_patient(request.user):
        appointment = get_object_or_404(Appointment, pk=id)
        if appointment.status == AppointmentStatus.OPEN:
            if request.POST.get("reschedule", None):
                print("RESCHEDULE")
                old_appointment = get_object_or_404(Appointment, pk=request.POST.get("reschedule", None))
                appointment.patient = old_appointment.patient
                appointment.status = AppointmentStatus.BOOKED
                appointment.apt_type = old_appointment.apt_type
                appointment.notes = old_appointment.notes
                appointment.save()
                if old_appointment.status == AppointmentStatus.BOOKED:
                    old_appointment.status = AppointmentStatus.OPEN
                old_appointment.save()
                send_patient_email(appointment, request, "5")
                send_doctor_email(appointment, request, "5")
                return HttpResponseRedirect("/appointments/?status=1&message=5")
            else:
                appointment.status = AppointmentStatus.BOOKED
                appointment.apt_type = request.POST.get("type", "")
                appointment.patient = PatientProfile.objects.get(user=request.user)
                appointment.save()
                send_patient_email(appointment, request, "1")
                send_doctor_email(appointment, request, "1")
                return HttpResponseRedirect("/appointments/?status=1&message=1")
        else:
            return HttpResponseRedirect("/appointments/?status=1&message=11")
    else:
        return HttpResponse("You are not authorized to book this appointment")
```

j. Bulk Create Appointments

```
def bulk_apt_create(request):
    if request.method == "GET":
        if is_user_patient(request.user):
            return HttpResponse("Unauthorized")
        context = {}
        return render(request, "bulk_create.html", context)
    if request.method == "POST":
        sdt = request.POST.get("sdt", "")
        sdt = list(map(int, sdt.split("-")))
        edt = request.POST.get("edt", "")
        edt = list(map(int, edt.split("-")))
        sdt = date(sdt[0], sdt[1], sdt[2])
        edt = date(edt[0], edt[1], edt[2])
        slot_names = request.POST.get("slot_name", None)
        slot_names = list(filter(None, slot_names.split(",")))
        print(slot_names)
        doctor = DoctorProfile.objects.get(user=request.user)
        for single_date in rrule(DAILY, dtstart=sdt, until=edt):
            print(single_date)
            for slot in slot_names:
                st = request.POST.get(f"st{slot}", None)
                et = request.POST.get(f"et{slot}", None)
                st = datetime.strptime(st, "%H:%M")
                if et == "":
                    et = None
                else:
                    et = datetime.strptime(et, "%H:%M")
                appt = Appointment.objects.create(doctor=doctor, date=single_date, start_time=st, end_time=et,
                status=AppointmentStatus.OPEN)
        return HttpResponseRedirect("/appointments/?status=0&message=0")
```

k. Send Notifications

```
def send_noti(request):
    if request.method == "GET":
        if is_user_patient(request.user):
            return HttpResponse("Unauthorized")
        context = {}
        return render(request, "send_noti.html", context)
    if request.method == "POST":
        date = request.POST.get("date")
        doctor = DoctorProfile.objects.get(user=request.user)
        appointments = Appointment.objects.filter(
            date=date, doctor=doctor, status=AppointmentStatus.BOOKED
        )
        for appointment in appointments:
            print(appointment)
            send_patient_email(appointment, request, "9")
        return HttpResponseRedirect("/appointments/?status=1&message=9")
```

6. Templates

Django provides a convenient way to generate dynamic HTML pages by using its template system. A template consists of static parts of the desired HTML output as well as some special syntax using the django templating language describing how dynamic content will be inserted. They are rendered by the server and sent to the browser to be shown. We have used templates to render all our pages for the app.

appointment_management > users > templates > patient_profile.html > ...

```
{% extends 'layout.html' %}
{%load static%}
{% block title %}Patient's Profile{% endblock %}

{% block content %}
<link rel="stylesheet" href="{%static 'css/register.css'%}">
<br>
<div class="container register-form">···
</div>
<style>···
</style>
<div class=" main container mt-5 mb-5">
<div class="d-flex justify-content-between mb-3"> <span>Appointment History</span>···
</div>
{% for apt in appointments reversed %}
<div class="card mt-5 border-5 pt-2 active pb-0 px-3">
  <div class="card-body ">
    <div class="row">
      <div class="col-12 ">
        <h4 class="card-title ">
          <b>{{apt.date}} </b>
          {{apt.start_time |date:'f A'}}
          {%if apt.end_time %}
          - {{apt.end_time|date:'f A'}}
          {%endif%}
        </h4>
      </div>
      <div class="col">···
      </div>
    </div>
  </div>
  <div class="card-footer bg-white px-0 ">···
  </div>
</div>
{% empty %}
<div class="card">
  <div class="card-body">
    No Appointments registered for this Patient.
  </div>
</div>
{% endfor %}
```

We use extend tag to put this template within the layout.html base template.

Blocks are reusable sections of the page that can be used by different pages for different content.

The static tag is used to load static file urls dynamically.

We have used a for loop to render the repeating data.

We have also used if else statements within the template to conditionally render the data.

7. URLs

The URL mapping will redirect requests from the web to the project's URLs to app URLs and then to the respective view function.

Base URLs:

```
urlpatterns = [
    path('accounts/', include('django.contrib.auth.urls')),
    path('admin/', admin.site.urls,name='admin'),
    path('users/', include('users.urls')),
    path('', views.home,name='home'),
    path('appointments/', include('appointments.urls')),
    path('sw.js', TemplateView.as_view(template_name='sw.js', content_type='application/x-javascript')),
    url(r'^favicon\.ico$',RedirectView.as_view(url='/static/images/favicon.ico')),
] + static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
```

User URLS:

appointment_management > users > urls.py > ...

```python
from django.urls import path
from . import views
from django.contrib.auth.views import LogoutView
from django.conf.urls import url

urlpatterns =[
    path('login/', views.loginpage,name='login'),
    path('register/', views.register,name='register'),
    path('register/doctor', views.doctor_register,name='doctor_register'),
    path('register/patient', views.patient_register,name='patient_register'),
    path('activate/<slug:uidb64>/<slug:token>/', views.activate, name='activate'),
    path('profile/', views.profile,name='profile'),
    path('patient_profile/<int:id>/',views.patient_profile,name='patient_profile'),
    path('logout', LogoutView.as_view(next_page='loggedout'), name='logout'),
    path('loggedout', views.logout_next, name='loggedout'),
]
```

Appointment URLS:

appointment_management > appointments > urls.py > ...

```python
from django.urls import path
from . import views
from django.contrib.auth.views import LogoutView
from django.conf.urls import url

urlpatterns =[
    path('',views.AppointmentsList.as_view(),name='appointments'),
    path('add/', views.add_appointment,name='add_appointment'),
    path('add/multiple/',views.bulk_apt_create,name='multiple_appointments'),
    path('edit/<int:id>/', views.edit_appointment,name='edit_appointment'),
    path('delete/<int:id>/', views.delete_appointment,name='delete_appointment'),
    path('cancel/<int:id>/', views.cancel_appointment,name='cancel_appointment'),
    path('complete/<int:id>/', views.complete_appointment,name='complete_appointment'),
    path('checkin/<int:id>/', views.checkin_appointment,name='checkin_appointment'),
    path('checkout/<int:id>/', views.cancelCheckIn_appointment,name='cancelCheckIn_appointment'),
    path('book/<int:id>',views.book_appointment,name='book_appointment'),
    path('send_noti/',views.send_noti,name='send_notifications'),
]
```

8. Model Forms

Model forms can be used to generate template forms directly from the models to perform Create and Update operations. We have used model forms to create and edit appointments.

appointment_management > appointments > forms.py > patientAppointmentForm

```python
from django import forms
from .models import Appointment

# creating a form
class appointmentForm(forms.ModelForm):
    required_css_class = 'required'
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
    # create meta class
    class Meta:
        # specify model to be used
        model = Appointment
        # specify fields to be used
        fields = ["status","date","start_time","end_time","notes","patient","apt_type","billing_amount","payment_status"]

class patientAppointmentForm(forms.ModelForm):
    required_css_class = 'required'
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
    # create meta class
    class Meta:
        # specify model to be used
        model = Appointment
        # specify fields to be used
        fields = ["notes","apt_type"]

```

9. Static Files Configuration

Websites generally need to serve additional files such as images, JavaScript, or CSS. In Django, we refer to these files as “static files”. We have configured the static files folder in the settings.py file of the project.

10. Custom Validators

We have customized password validators in our project to make sure the passwords created during registration are up to the standards.

appointment_management > appointment_management > validators.py > ...

```python
import re
from django.core.exceptions import ValidationError
from django.utils.translation import ugettext as _

class UppercaseValidator(object):
    def validate(self, password, user=None):
        if not re.findall('[A-Z]', password):
            raise ValidationError(_("The password must contain at least 1 uppercase letter, A-Z."),code='password_no_upper',)
    def get_help_text(self):
        return _("Your password must contain at least 1 uppercase letter, A-Z.")

class LowercaseValidator(object):
    def validate(self, password, user=None):
        if not re.findall('[a-z]', password):
            raise ValidationError(_("The password must contain at least 1 lowercase letter, a-z."),code='password_no_lower',)
    def get_help_text(self):
        return _("Your password must contain at least 1 lowercase letter, a-z.")

class SymbolValidator(object):
    def validate(self, password, user=None):
        if not re.findall('[()[\]{}|\\`~!@#$%^&*_\-+=;:\'",<>./?]', password):
            raise ValidationError(_("The password must contain at least 1 symbol: " +"()[]{}|\`~!@#$%^&*_-+=;:'\",<>./?"),
            code='password_no_symbol',)
    def get_help_text(self):
        return _("Your password must contain at least 1 symbol: " +"()[]{}|\`~!@#$%^&*_-+=;:'\",<>./?")

class NumberValidator(object):
    def __init__(self, min_digits=1):
        self.min_digits = min_digits
    def validate(self, password, user=None):
        if not len(re.findall('\d', password)) >= self.min_digits:
            raise ValidationError(_("The password must contain at least %(min_digits)d digit(s), 0-9."),code='password_no_number',
            params={'min_digits': self.min_digits},)
    def get_help_text(self):
        return _("Your password must contain at least %(min_digits)d digit(s), 0-9." % {'min_digits': self.min_digits})

```

11. Template Context Processors

Context processors are used to provide additional data to the template that can be used to view files or generate logic. We have used customized context processors to check the user type and generate the template accordingly. We have also used a context processor to modify the admin template.

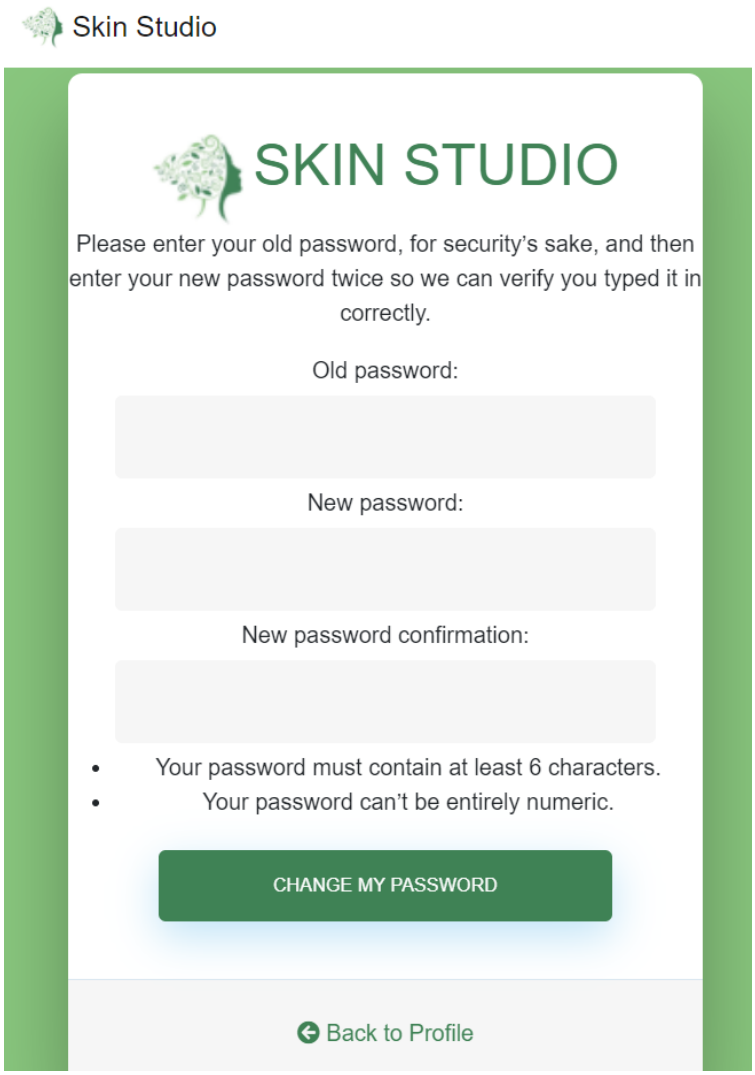
```python
def user_processor(request):
    data = {'user': getattr(request, 'user', None)}
    data['is_doctor'] = False
    data['is_patient'] = False
    if data['user']:
        if request.user.is_authenticated:
                for group in request.user.groups.all():
                        if group.name == 'doctor':
                                data['is_doctor'] = True
                                break
                        elif group.name == 'patient':
                                data['is_patient'] = True
                                break
    return data

def admin_header_processor(request):
    context = {}
    context['index_title'] =  "Welcome to Skin Studio"
    context['site_header'] = 'Skin Studio'
    context['site_name'] = 'Skin Studio'
    context['site_title'] = "Skin Studio - Admin Portal"
    return context
```

```python
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates'),
        ],
        'APP_DIRS': True,
        'OPTIONS': {
            # 'loaders': [
            #     'django.template.loaders.cached.Loader',
            # ],
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
                'appointment_management.context_processor.webpush_processor',
                'appointment_management.context_processor.user_processor',
                'appointment_management.context_processor.admin_header_processor',
            ],
        },
    },
]
```

12. Overwriting Django Default Templates

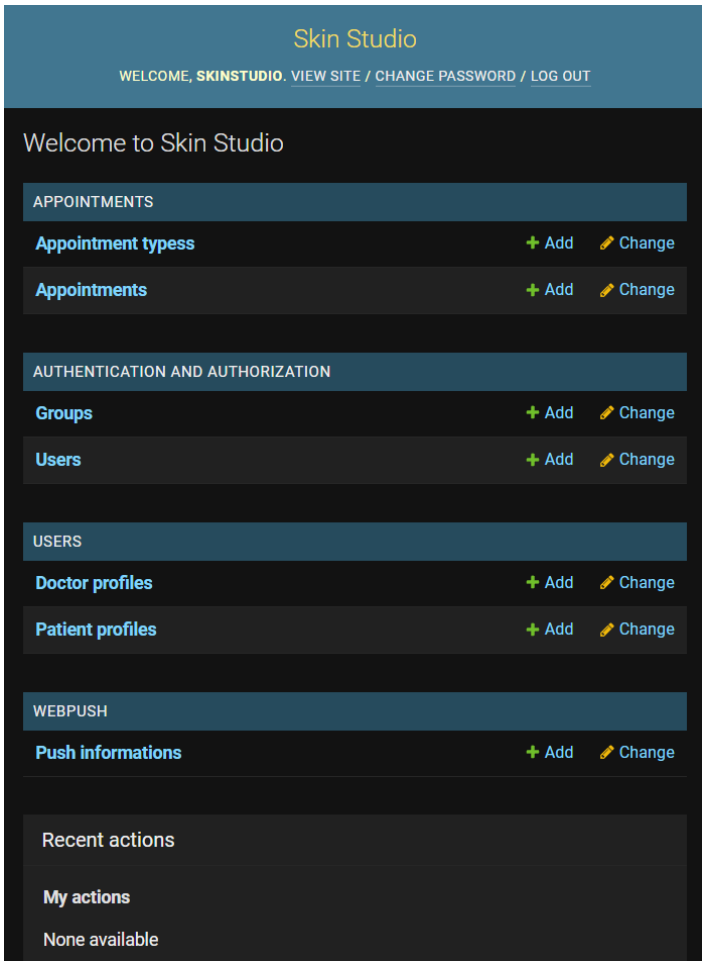
In the main project URLs, we have modified the admin site to reflect the name of our project. We have also customized the django default templates to look similar to our login page template.

```
admin.site.site_header = "Skin Studio"
admin.site.site_title = "Skin Studio - Admin Portal"
admin.site.index_title = "Welcome to Skin Studio"
admin.site.login_template = "login.html"
auth_views.PasswordResetView.template_name = 'password_reset_form.html'
auth_views.PasswordResetView.email_template_name = 'password_reset_email.html'
auth_views.PasswordResetDoneView.template_name = 'password_reset_done.html'
auth_views.PasswordResetConfirmView.template_name = 'password_reset_confirm.html'
auth_views.PasswordResetCompleteView.template_name = 'password_reset_complete.html'
auth_views.PasswordChangeView.template_name = 'password_change_form.html'
auth_views.PasswordChangeDoneView.template_name = 'password_change_done.html'
```



13. Modifying the Admin Site

We have modified the admin template to reflect the name of the business and be personalized to the user's needs by overwriting the default templates and by using context processors. We have also registered our models to the admin template so that the admin can easily manage all data of the site.



14. Notification Emails using SMTP

We have setup django SMTP mail sending using Gmail in order to send appointment notification emails and other transactional emails to the users.

```
# Email Settings

EMAIL_USE_TLS = True
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587
EMAIL_HOST_USER = 'skinstudiobyrasnakapoor@gmail.com'
```

```
def send_patient_email(appointment,request, status):
    try:
        current_site = get_current_site(request)
        mail_subject = status_text_patient[status]['head']
        message = render_to_string('email_apt_pt.html', {
            'appointment': appointment,
            'site': current_site,
            'form': appointmentForm(instance=appointment),
            'text': status_text_patient[status]['body'],
        })
        to_email = [appointment.patient.user.email]
        email = EmailMessage(
            mail_subject, message, to=to_email
        )
        email.content_subtype="html"
        email.send()
        print("sent")
    except Exception as e:
        print(e.message)
```

15. Cryptographic Token Generation for Password Reset

We have used the ‘six’ library to create hashed tokens that are used to verify email addresses, activate the account, and for password reset links.

appointment_management > users > tokens.py > ...

```python
from django.contrib.auth.tokens import PasswordResetTokenGenerator
import six
class TokenGenerator(PasswordResetTokenGenerator):
    def _make_hash_value(self, user, timestamp):
        return (
            six.text_type(user.pk) + six.text_type(timestamp) +
            six.text_type(user.is_active)
        )
account_activation_token = TokenGenerator()

```

```python
def activate(request, uidb64, token):
    try:
        uid = force_text(urlsafe_base64_decode(uidb64))
        user = User.objects.get(pk=uid)
    except(TypeError, ValueError, OverflowError, User.DoesNotExist):
        user = None
    if user is not None and account_activation_token.check_token(user, token):
        user.is_active = True
        user.save()
        return render(request,'login.html',{"msg":"Account activated successfully"})
    else:
        return HttpResponse('Activation link is invalid!')
```

Word Count: 1263